Interpretable Machine Learning Models via Maximum Boolean Satisfiability

Hao Hu

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *14/12/2022* **par :**

Hao Hu

Interpretable Machine Learning Models via Maximum Boolean Satisfiability

---

JURY

Hélène Fargier	Directrice de Recherche	Présidente du jury
Frédéric Koriche	Professeur des Universités	Rapporteur
Chu-Min Li	Professeur des Universités	Rapporteur
Djamal Habet	Professeur des Universités	Examinateur
Marie-José Huguet	Professeure des Universités	Directrice de thèse
Mohamed Siala	Maître de Conférences	Directeur de thèse

École doctorale et spécialité :

MITT : Informatique

Unité de Recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS)

Directeur(s) de Thèse :

Marie-José Huguet et *Mohamed Siala*

Rapporteurs :

Chu-Min Li et *Frédéric Koriche*

世界上只有一种英雄主义，那就是在认清生活真相后，还依然热爱它

(罗曼 • 罗兰 — 米开朗基罗传, 1907)

There is only one heroism in the world: to see the world as it is, and to love it.

(Romain Rolland — Life of Michelangelo, 1907)

Il n'y a qu'un héroïsme au monde: c'est de voir le monde tel qu'il est, et de l'aimer.

(Romain Rolland — Vie de Michel-Ange, 1907)

Acknowledgments

As the last part to write for a thesis, the part of acknowledgments is naturally considered as the easiest one. However, as its special milestone property, personally, it is not. It seems like lots of events happened just in yesterday: the failure of my college entrance examination, the first touch of Computer Science, the acquaintance with my fiancée, the first arrival day in France, the decision for pursuing the CS Ph.D, the COVID pandemic, the thesis defense... Life is not a rehearsal, but it does leave lots of accidental but fantastic memory, with lots of initial strangers.

At first, my sincere thanks to my doctoral supervisors: Professor Marie-José Huguet and Associate Professor Mohamed Siala. I will never forget the period of working together with them on the research of interpretable Machine Learning models. The serious and diligent teaching and researching attitude of Marie-Jo influence me forever. As a foreign student, I am grateful for their understanding and caring in my daily life, especially during the period of the COVID pandemic.

My special thanks to CNRS Researcher Emmanuel Hebrard, who provided me with the research internship chance in LAAS-CNRS. I appreciate his permission to let me explore my research interest freely, which motivates the research topic of my doctoral thesis.

Meanwhile, I would like to express my great gratitude to all my jury members: Professor Chu-Min Li, Professor Frédéric Koriche, Professor Hélène Fargier, and Professor Djamal Habet. Thank you all for reading my manuscript. Especially to Chu-Min and Frédéric, I am grateful for their valuable comments and physical presence in the defense.

I would like to thank all of my colleagues at ROC group, who are all nice and gentle to me. Moreover, I appreciate the French Ministry of Higher Education, Research and Innovation (MESRI) for offering me the 3-year doctoral contract, and LAAS-CNRS for the reception.

I would like to thank lots of my previous friends and teachers. RuiZhe Cao, we have lots of shared tourist experiences in Europe. I do miss those periods we spent in the Alpes, and I do hope you work well in Hangzhou. Associate Professor Tao Xu, I appreciate your help for my Ph.D application and my academic post at Northwestern Polytechnical University (NPU). I am looking forward to working with you in the future.

Additionally, I would like to thank lots of friends met in Toulouse. Dr. Zhen-Hang Wu, I am grateful for lots of discussions with you and your suggestions like a “brother”. I do hope you work well in Michelin Shanghai. Dr. YuXiao Mao, I am grateful for lots of discussions in research with you, which help me release the stress in research. I do hope you work well in Bordeaux. Dr. Tong Chen, during a special period, we are the only two Chinese in LAAS-CNRS, I am grateful for lots of support in daily life, and your cute cat. I do hope you can find a perfect Post-doc post very soon. XingHan Liu, a crazy but fake Arsenal fan, I am grateful for your support to solve my financial distress. All the best to your defense at the end of this year, and your badminton skill. Also thanks a lot to others: FangKai

Xue, GuangJie Zhang, Chao Ren, ShuLi Wang, Lei Ren, TaiWei Xu, Gang Huang, etc, And those friends playing badminton together in ENAC.

Especially, I would like to express my gratitude to all of my Chinese students at INSA Toulouse. Thank you for providing me with the chance to be a teaching assistant, which makes me feel not only the hardness but also the happiness in teaching. Those teaching experiences will help me a lot in being a true lecturer in the future. I do hope all of you pass all exams and graduate with no problem within the next several years.

Furthermore, I would like to thank Northwestern Polytechnical University, where I got my undergraduate education and my future academic position. Besides, I would like to thank the city of Toulouse, "la ville rose". In my view, it is the most livable city in Europe because of its inclusive society and warm people.

Finally, also most importantly, I express my extreme gratitude to my parents and my fiancée ZiChen Geng. After beginning my undergraduate education, my parents always respect my choice and do their best to support me mentally and financially for my further education. If the no reason support from my parents is due to blood, the support from my fiancée could be only the reason of love. We fell in love with each other in 2016 and just spent 11 months together before I went to France. Although sometimes you complained about the difficulty of separation, you always support my decision. I was majorly absent in your life during the last 5 years, the only thing I can do is fill your future life from now on, until the end of my life.

Hao Hu,
Toulouse, February 14, 2023.

Abstract

Interpretable Machine Learning models receive growing interest due to the increasing concerns in understanding the reasoning behind some crucial decisions made by modern Artificial Intelligent systems. Due to their structure, especially with small sizes, these interpretable models are inherently understandable for humans. Compared to classical heuristic methods to learn these models, recent exact methods offer more compact models or better prediction quality. In this thesis, we propose two novel exact methods via Maximum Boolean Satisfiability (MaxSAT) to learn optimal interpretable machine learning models.

Our contribution starts with an original MaxSAT-based exact method to learn optimal decision trees. This method optimizes the empirical accuracy to avoid overfitting, and also enriches the constraints to restrict the tree depth. Additionally, we integrate this MaxSAT-based method in AdaBoost, which is a classical Boosting method to improve the generalization performance. The experimental results show competitive prediction quality of this MaxSAT-based method compared to state-of-the-art heuristic and other exact methods. Additionally, clear improvements in prediction performance are observed after the integration in AdaBoost. Our second contribution is an original MaxSAT-based exact method to optimize binary decision diagrams. We introduce an initial Boolean Satisfiability (SAT) encoding to model binary decision diagrams in limited depth with perfect empirical accuracy. Next, we present how to adapt the SAT-based model into MaxSAT approach. Finally, we present a pre-processing for selecting some important features to increase the scalability of our MaxSAT-based method to optimize binary decision diagrams. The experimental results show clear advances of our MaxSAT-based method in prediction quality, compared to state-of-the-art heuristic methods. We also observe a huge shrink in encoding size and model size in comparison between our approach and state-of-the-art exact method without losing the prediction performance. In addition, great reductions in encoding size are displayed after the application of pre-processing, which boosts the scalability.

Keywords: Interpretable Machine Learning, Maximum Boolean Satisfiability, Decision Trees, Decision Diagrams.

Résumé

Les modèles d'apprentissage interprétables reçoivent un intérêt croissant en raison de l'augmentation des préoccupations pour comprendre le raisonnement menant aux décisions cruciales prises par les systèmes modernes d'intelligence artificielle. En raison de leur structure, en particulier pour des petites tailles, ces modèles interprétables sont intrinsèquement compréhensibles pour les humains. Par rapport aux méthodes heuristiques classiques pour apprendre ces modèles, les méthodes exactes récentes offrent des modèles plus compacts ou atteignent une meilleure qualité de prédiction. Dans cette thèse, nous proposons deux nouvelles méthodes exactes basées sur la Satisfiabilité Booléenne Maximale (MaxSAT) pour apprendre des modèles d'apprentissage interprétables optimaux.

Notre contribution commence par une méthode exacte originale basée sur MaxSAT pour apprendre des arbres de décision optimaux. Cette méthode optimise la précision empirique pour éviter le surapprentissage et prend également en compte des contraintes pour restreindre la profondeur de l'arbre. De plus, nous intégrons cette méthode basée sur MaxSAT à la méthode AdaBoost, qui est une méthode standard de Boosting pour améliorer les performances de généralisation. Les résultats expérimentaux montrent une qualité de prédiction compétitive de cette méthode basée sur MaxSAT par rapport à des méthodes heuristiques et exactes de l'état de l'art. En plus, des améliorations des performances de prédiction sont observées après intégration dans AdaBoost. Notre deuxième contribution est une méthode exacte originale basée sur MaxSAT pour optimiser les diagrammes de décision binaire. Nous introduisons tout d'abord un encodage de Satisfiabilité Booléenne (SAT) pour modéliser des diagrammes de décision binaire de profondeur limitée avec une parfaite précision. Puis, nous présentons comment adapter le modèle en Satisfiabilité Booléenne Maximale. Finalement, nous présentons un pré-traitement pour la sélection de certaines caractéristiques importantes afin d'augmenter le passage à l'échelle de notre méthode MaxSAT pour optimiser les diagrammes de décision binaire. Les résultats expérimentaux montrent des avancées de notre méthode MaxSAT sur la qualité de prédiction, par rapport aux méthodes heuristiques. Nous observons également une réduction importante sur la taille d'encodage et la taille du modèle dans les comparaisons entre notre approche et une méthode exacte de l'état de l'art, sans perdre en performance de prédiction. De plus, une grande réduction sur la taille d'encodage est mise en évidence après application du pré-traitement, ce qui renforce le passage à l'échelle.

Mots-clés : Apprentissage Interprétable, Satisfiabilité Booléenne Maximale, Arbres de Décision, Diagrammes de Décision.

Contents

Introduction

In the last decade, Machine Learning, received a great success in solving many real world problems. These successes have increased the develpment of eXplainable Artificial Intelligence (XAI), especially for high stakes decision systems. XAI aims to produce AI systems that can be understood by human. The goal is to raise the trust that humans can have in AI systems, by understanding the process leading to a given decision or prediction. The field of XAI covers a broad spectrum of research. In this thesis, we are interested in the interpretability of Machine Learning models. In the literature, two main approaches exist to increase the interpretability of Machine Learning models. The first one, called *post hoc* approach, focuses on *black-box* models, such as neural networks or deep learning, and considers a posteriori explanations [Guidotti *et al.* 2018]. One can produce explanations on the output of the black-box model to detail the reason of a given prediction or a black-box inspection to explain how a given black-box works. In the second approach, called *transparency-by-design*, the goal is to produce Machine Learning models that can be understood by humans, based on their simple structure. For instance, *Decision Trees*, *Decision Sets*, or *Decision Rules* are considered as interpretable by-design models when they have small size. Some drawbacks of the black-box explanation approach, that can provide misleading or false explanations, have been highlighted in [Rudin 2019, Laugel *et al.* 2019]. For crucial applications, where decisions may impact individual, such drawbacks raise the need of inherently interpretable Machine Learning models. In [Rudin *et al.* 2021], ten challenges for the development of inherently interpretable Machine Learning models are detailed. The first challeng concerns how to efficiently compute optimal and sparse Machine Learning models. This thesis is in-line with this challenge.

There are numerous heuristic methods for interpretable machine learning models. Although these classical heuristic methods have reduced computation time, the interpretable machine learning models built are often huge in size, making difficult to understand how the model works. To insure that machine learning models found are truly "interpretable", recently, there is growing interest in exact methods for those models. Compared to heuristic approaches, exact methods offer the promise of optimality, for instance in *model size*, *model depth*, or *accuracy*. In this context, combinatorial optimisation methods, such as *Constraint Programming*, *Mixed Integer Linear Programming*, *Boolean Satisfiability (SAT)*, and *Dynamic Programming* have successfully applied for learning optimal interpretable machine learning models. These declarative approaches are particularly interesting since they offer certain flexibility to handle additional requirements when learning a model.

In this thesis, we focus on *Maximum Boolean Satisfiability (MaxSAT)* approach, where MaxSAT is an optimisation version of SAT, to learn optimal interpretable models. A disadvantage of recent SAT-based exact methods is the promise of perfect empirical accuracy for a given model size, or model depth, which is risk in over-

fitting. However, MaxSAT-based exact methods could avoid this disadvantage by optimizing empirical accuracy. Moreover, MaxSAT has a weighted extension, where weights of clauses could naturally approximate the data distribution of datasets used in machine learning. In addition, this strength makes MaxSAT-based exact methods easy to be adapted in Boosting methods.

Thesis Overview

We offer an overview of the thesis, which contains three chapters. Chapter 1 presents a technical background in Boolean Satisfiability, including Boolean Satisfiability (SAT) and its variant Maximum Boolean Satisfiability (MaxSAT). Then, we present important notions in Machine Learning, including interpretable models and ensemble methods. Finally, Chapter 1 provides a literature review of recent related works in interpretable Machine Learning models, including classical heuristic methods and recent exact methods. Chapter 2 presents our contributions for learning optimal decision trees via MaxSAT and its integration in AdaBoost. Chapter 3 presents our contributions in optimizing binary decision diagrams via MaxSAT. We give a summary of the contributions made in this thesis.

1. **Learning optimal decision trees via MaxSAT and its integration in AdaBoost**
   As a very popular machine learning model, decision tree benefits from its inherent interpretability, and the wide range of efficient heuristic methods to compute it. However, due to the explosion in tree size and depth, decision trees found by classical heuristic methods suffers the difficulty in interpretability. This weakness motivates the exact methods to learn optimal decision trees with guarantees of mathematical optimality in some metrics, like *tree size*, *tree depth*, and *accuracy*. Some exact methods are SAT-based, where [Bessiere *et al.* 2009, Narodytska *et al.* 2018] optimize the tree size, and [Avellaneda 2020, Janota & Morgado 2020] optimize the tree depth. However, all of them must subject to the constraint that the decision tree is perfectly accurate on the training set, which is often criticized as it may entail overfitting.

   To offset this drawback, we firstly introduce a MaxSAT approach to learn optimal decision trees by optimizing the accuracy, which is the adaption of the previous SAT approach [Narodytska *et al.* 2018]. At first, we introduce the details of previous SAT encoding, and, we show how to transform the SAT encoding to MaxSAT to optimize the accuracy. Then, we propose new constraints to limit the depths. Next, to improve the prediction quality, we integrate the MaxSAT approach proposed in AdaBoost by adjusting the weights of soft clauses.

   Based on large experimental results, we observe the overfitting phenomenon of previous SAT approach. Moreover, we observe competitive prediction per-

formance of the proposed MaxSAT approach compared to state-of-the-art heuristic and exact methods. Additionally, we perceive clear improvements in prediction quality after the integration in AdaBoost.

2. **Optimizing binary decision diagrams with MaxSAT**
   By providing compact representations for Boolean functions, binary decision diagrams are viewed as interpretable in binary classification. Compared to decision trees, *ordered reduced* binary decision diagrams could avoid the *replication* problem and the *fragmentation* problem effectively [Oliver 1992, Kohavi 1994], which are two major flaws suffered by decision trees. To the best of our knowledge, the only exact method to learn optimal binary decision diagrams is [Cabodi *et al.* 2021], whose target are binary decision diagrams with the smallest sizes that classify all examples correctly. However, this target leads to two drawbacks. The first one is the possible overfitting due to the perfect accuracy. The second one is the lack of restrictions in depth, making it possible that the binary decision diagrams learnt are small in size but high in depth.

   To avoid these disadvantages, we propose a MaxSAT-based approach to learn binary decision diagrams limited in depths with the best empirical accuracy. At first, we introduce an initial SAT-based model to encode binary decision diagrams of given depth with perfect accuracy. Then, we lift the SAT-based model into MaxSAT to optimize the accuracy. In addition, as the complexity of MaxSAT encoding relates strongly to the corresponding SAT encoding, we propose two other SAT-based models for the same objective but with tighter encoding sizes. Finally, to increase the scalability of the proposed MaxSAT approach, we present an hybrid version that selecting a subset of important features by heuristic method, then applying the MaxSAT approach proposed.

   Our experimental evaluations show the comparison between our MaxSAT approach for optimal binary decision diagrams with state-of-the-art heuristic and exact methods. Firstly, compared to heuristic method, our MaxSAT approach shows clear advantages in prediction quality. Then, compared to exact method (the MaxSAT approach of learning optimal decision trees), our MaxSAT approach shows considerable shrink in encoding sizes and model sizes. Meanwhile, our MaxSAT approach displays competitive prediction performance. Finally, the hybrid version is easier in reporting optimality than the original one, but still remains competitive in prediction quality.

Publications

The works presented in this thesis were published in two international conferences. These publications are detailed in Chapter 2 and Chapter 3.

- ***Learning Optimal Decision Trees with MaxSAT and its integration in AdaBoost***. Hao Hu, Mohamed Siala, Emmanuel Hébrard, Marie-José

Huguet. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, page 1170–1176*. [Hu *et al.* 2020]

- ***Optimizing Binary Decision Diagrams with MaxSAT for classification***. Hao Hu, Marie-José Huguet, Mohamed Siala. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, page 3767–3775*. [Hu *et al.* 2022]

We are honored that the MaxSAT formulas of learning optimal decision trees are selected as benchmarks in *MaxSAT Evaluation* 2021 [Bacchus *et al.* 2021a] and 2022 [Bacchus *et al.* 2022]. The descriptions of benchmarks are detailed in the following document, given in Appendix A.

- ***Description of Benchmarks on Learning Optimal Decision Trees and Boosted Trees***. Hao Hu, Emmanuel Hébrard, Marie-José Huguet, Mohamed Siala. In *MaxSAT Evaluation 2021, page 39-40*. [Hu *et al.* 2021]

We have also presented our work on the optimization of binary decision diagrams with MaxSAT, at two francophone conferences: *JFPC 2022 (Journées Francophones de Programmation par Contraintes)* and *CNIA 2022 (Conférence Nationale en Intelligence Artificielle)*.

Chapter 1

Formal Background & State-Of-The-Art

Contents

In this chapter, we present technical background and state-of-the-art methods for Boolean Satisfiability and for Machine Learning. Section 1.1 introduces the Boolean Satisfiability (SAT) and the Maximum Boolean Satisfiability (MaxSAT) problems. Section 1.2 presents supervised Machine Learning, and focuses on interpretable machine learning, and ensemble methods. Section 1.3 provides a literature review of recent related works in interpretable machine learning models. Specifically, it contains the state-of-the-art methods on Decision Trees and Decision Graphs.

1.1 SAT and MaxSAT Problems

The **Boolean Satisfiability** problem (SAT) aims to determine whether a Boolean formula is satisfiable or not. The SAT problem has a key role in computer science, in particular because it is the first problem proven to be NP-complete. In this section, we describe formally some related notions in propositional logic by following the standard terminology from [Biere *et al.* 2021]. Furthermore, we describe the Maximum Boolean Satisfiability problem (MaxSAT), considered in this thesis.

1.1.1 Boolean Satisfiability

An *atom* x is a propositional (i.e., Boolean) variable. A *literal* p is either an atom x, called positive literal, or its negation $\neg$x, called negative literal. A literal p is *true* iff p is positive and its atom is assigned to the value 1, or p is negative and its atom is assigned to the value 0. Otherwise, the literal p is *false*, that is $\neg p$ is *true*.

A *clause* c is a disjunction of literals $(p_1 \vee \cdots \vee p_k)$. We suppose that all literals in a clause are pairwise distinct, and that literals p and $\neg p$ do not appear in the same clause. A clause c is *satisfied* if at least one literal p appearing in the clause $(p \in c)$ is *true*. Conversely, if none of literals appearing in the clause c is *true*, then c is *unsatisfied*.

A proposition formula represented by conjunctions of clauses $c_1 \wedge \cdots \wedge c_n$ is said to be in *Conjunctive Normal Form (CNF)*. A CNF formula corresponds to a SAT instance. The SAT problem consists in determining the satisfiability of a CNF formula. The goal is to find an assignment for all literals appearing in the CNF formula that satisfies all clauses in the formula (the formula is *satisfied* or "SAT" in short); or to report the failure of finding such assignment (the formula is *unsatisfied* or "UNSAT" in short).

There are several complete algorithms to solve the SAT problem. One of the oldest method is the *Davis-Putnam-Logemann-Loveland (DPLL)* algorithm [Davis *et al.* 1962]. The state-of-the-art method applied in most modern SAT solvers is the *Conflict Driven Clause Learning (CDCL)* algorithm [Silva & Sakallah 1996, Silva & Sakallah 1999, Moskewicz *et al.* 2001, Eén & Sörensson 2003].

In the DPLL algorithm, the search space is explored via search tree where every node corresponds to a decision (branching variable selection and value assignment) restricting the search space to a smaller problem. The search tree is explored in a Depth-First Search (DFS) scheme. The backtrack to the last node happens when an "UNSAT" is detected. Then the last decision is reversed and the exploration is resumed. This kind of backtracking is called *chronological backtracking*. During the exploration, two steps are executed at each node of the search tree: *search* and *propagation*. The *search* step relates to the branching variable selection and value assignment by heuristic method to explore the search structure. Some classical heuristics include pure random selection, and maximum occurrences on clauses of minimum size (MOM) [Zabih & McAllester 1988, Silva 1999]. And the *propagation* step relates to the pruning of dead-end branches. The DPLL algorithm uses one

type of propagation called *Unit-propagation (UP)*, which is triggered in two possible conditions. The first condition is whenever a clause c has only one unassigned literal p (c is called as a unit clause), the UP enforces p in c to be *true* as it is the only way to make c satisfied. The second condition is when a clause c is unsatisfied by the assigned literals, the UP return an "UNSAT" of the formula directly.

We describe the principal idea of DPLL algorithm in Algorithm 1.

Algorithm 1: The DPLL Algorithm.

Input: A Set of clauses $\mathcal{C} = \{c_1, \ldots, c_n\}$.

Process *DPLL($\mathcal{C}$)*:
- **while** *there is a unit clause c has literal p unassigned* **do**
  - `// Delete the clause c from C due to the unit propagation.`
  - $\mathcal{C} = UnitPropagation(p, c)$
- **if** $\mathcal{C} = \emptyset$ **then**
  - `// All clauses are satisfied, current assignment of literals is a satisfied assignment.`
  - **return** *SAT*.
- **if** $\mathcal{C}$ *has unsatisfied clause* **then**
  - **return** *UNSAT*.
- `// Select a decision literal by heuristic.`
- $p = SelectDecisionLiteral(\mathcal{C})$.
- `// Explore the positive branch.`
- **if** *DPLL*($\mathcal{C} \wedge \{p\}$) $=$ *SAT* **then**
  - **return** *SAT*.
- **else**
  - `// Explore the negative branch.`
  - **return** DPLL($\mathcal{C} \wedge \{\neg p\}$)

Output: *SAT* with a satisfied assignment, or *UNSAT*.

Inspired by DPLL, the major progress of CDCL concerns the *conflict analysis* when an "UNSAT" is detected. The conflict analysis generates *learning clause* based on the *implication graph* to explain the reason of leading to the failure. In addition, CDCL uses an alternative backtracking scheme called *non-chronological backtracking* (also known as the term *backjump*) to avoid making the same mistake again. Meanwhile, a number of new techniques are involved in CDCL. Some representative ones include an heuristic in branching variable selection called *Variable State Independent Decaying Sum (VSIDS)* [Moskewicz *et al.* 2001], the usage of *lazy structure* for the representation of formulas [Moskewicz *et al.* 2001], and the *periodically restarting* backtrack search [Gomes *et al.* 1998]. For details of CDCL algorithm, we refer readers to [Marques-Silva *et al.* 2021]. The representative CDCL solvers include **Chaff** [Moskewicz *et al.* 2001], **MiniSAT** [Eén & Sörensson 2003], and **Glucose** [Audemard & Simon 2009, Audemard & Simon 2018].

To evaluate the SAT solvers, from 2002, the *International Conference on Theory and Applications of Satisfiability Testing (SAT)* organizes annually the *SAT*

Competition[1].

1.1.2 Maximum Boolean Satisfiability

The Maximum Boolean Satisfiability problem (MaxSAT) consists in finding an assignment of all literals that maximizes the number of satisfied clauses.

A MaxSAT instance is represented by a *weighted CNF formula, that is* conjunctions of *weighted clauses*. A *weighted clause* is a pair (c_i, w_i), where c_i is a clause, and w_i is a positive number indicating its weight. In addition, three variants of MaxSAT problems are mainly studied in the literature:

- ***Weighted MaxSAT***: The weighted MaxSAT problem deals with a weighted CNF formula. The objective is to find an assignment that maximizes the *sum of weights* of satisfied clauses.
- ***Partial MaxSAT***: The partial MaxSAT problem deals with a CNF formula, in which all clauses are divided into two sets: *soft* (or *relaxed*) clauses, and *hard* (or *non-relaxable*) clauses. The objective is to find an assignment that maximizes the *number of satisfied soft clauses* and meanwhile satisfy *all hard clauses*.
- ***Weighted Partial MaxSAT***: The weighted partial MaxSAT is the combination of weighted MaxSAT and partial MaxSAT. It deals with a weighted CNF formula, where soft clauses are weighted. The objective is to find an assignment that satisfies all *hard clauses*, and maximizes the *sum of weights of satisfied soft clauses*.

One can note that the MaxSAT problem could be defined as a weighted MaxSAT problem by considering that all clauses share the same weights, and as Partial MaxSAT problem by declaring all clauses as soft clauses.

The objective of MaxSAT problem is to maximize the number of satisfied clauses, or equivalently to minimize the number of unsatisfied clauses (or MinUNSAT). Similarly, for weighted MaxSAT, the equivalent objective is to minimize the sum of weights of unsatisfied clauses; for partial MaxSAT, the equivalent objective is to minimize the number of unsatisfied soft clauses and meanwhile satisfy all hard clauses declared; and for weighted partial MaxSAT, the equivalent objective is to minimize the sum of weights of satisfied soft clauses and meanwhile satisfy all hard clauses declared.

In this thesis we focus on exact method to to solve the MaxSAT problem and its variants. In exact methods the goal is to compute the *optimal* solution and to prove the optimality. These methods could be roughly divided into two main groups: *Branch-and-Bound (BnB) algorithms* [Li & Manyà 2021], which directly tackle MaxSAT with a bounding procedure; and *SAT-based algorithms* [Bacchus *et al.* 2021b], which transform MaxSAT into a sequence of SAT instances and call a

[1] Learn more details from `http://www.satcompetition.org/`.

modern SAT solver to solve them. In practice, we refer to the MinUNSAT problem where upper bound is greater than or equal to the minimum number of unsatisfied clauses, lower bound is smaller than or equal to the minimum number of unsatisfied clauses. The optimality is reported when the upper bound equals to the lower bound.

Branch-and-Bound MaxSAT solvers implement the branch-and-bound scheme and incorporate a look-ahead procedure that detects inconsistent subsets of soft clauses by applying unit propagation and computes a lower bound. They also apply some inference rules at each node of the search tree. Some representative BnB solvers are **MaxSatz** [Li *et al.* 2007], **MiniMaxSat** [Heras *et al.* 2008], and **Akmaxsat** [Kügel 2010]. Especially, a recent solver **MaxCDCL** [Li *et al.* 2021] integrates the clause learning mechanism in the branch-and-bound, which significantly accelerates the speed of BnB solvers.

SAT-based MaxSAT solvers are based on a reformulation of the MaxSAT problem in a sequence of SAT problems, and consider three types of strategies for exploring the sequence: *linear search*, *core-guided*, and *minimum hitting-set-based* (MHS-based). We introduce them separately.

Linear search solvers use the upper bound approach of the MinUNSAT problem, and iteratively query a SAT solver for a better solution than the current best one. The optimality is reported when no such better solution could be found, and the current best one is the optimal solution. The representative linear search solvers include **SAT4J-MaxSAT** [Berre & Parrain 2010], **QMaxSAT** [Koshimura *et al.* 2012], and **Open-WBO** [Martins *et al.* 2014].

Core-guided and MHS-based solvers use the lower bound approach of the MinUNSAT problem. At first, they consider the input instance as SAT instance, and obtain an unsatisfied subset of soft clauses (called *core*) by using a SAT solver. This core is associated to a given lower bound on the number of unsatisfied clauses. Then, the solvers relax this core and solve the relaxed instance to identify another core. This process is repeated until a satisfiable instance is derived indicating the optimality is reached. The difference between core-guided and MHS-guided solvers is that core-guided solvers relax a core using cardinality constraints, while MHS-guided solvers minimizes the number of different clauses from the core by solving a minimum hitting set instance with an integer programming solver. The representative core-guided solvers include **MSU1.2** [Marques-Silva & Manquinho 2008], **Open-WBO** [Martins *et al.* 2014], and **RC2** [Ignatiev *et al.* 2019]. The representative MHS-based solvers include **MHS** [Saikko *et al.* 2016], and **MaxHS** [Davies & Bacchus 2011, Davies & Bacchus 2013].

Moreover, there is an hybrid version called *core-boosted linear search*, which combines the linear search and the core-guided approaches. At first the MinUNSAT problem is reformulated with a core-guided solver to produce a lower bound with limited time. Then, the problem is solved with a linear search solver. The exchange of information from the core-guided phase and the linear phase tightens the gap between the lower bound and the upper bound. As a result, this hybrid approach could be more effective than either a pure linear search or a pure

core-guided approach. The representative core-boost linear search solver is **Loandra** [Berg *et al.* 2019]. The default time out for core-guided phase of Loandra is 30 seconds.

To evaluate the MaxSAT solvers, from 2006, the *International Conference on Theory and Applications of Satisfiability Testing (SAT)* organizes annually the *MaxSAT Evaluation*[2]. Each year, the evaluation collects new *MaxSAT benchmarks*, and new open-source *MaxSAT solvers*.

1.2 Machine Learning

As an important part of Artificial Intelligence (AI), **Machine Learning** (ML) could be viewed as a set of algorithms that construct good **models** from **datasets**. Globally, a "model" is a *mapping* that maps inputs to predictions. A "dataset" is generally a set of *feature vectors*, where each feature vector is a description of an object using a set of *features*. The number of features of a dataset is called *dimension*. In some cases, features are also called as *attributes*, a feature vector is also called as an *instance*, or an *example*. The process of generating "models" from "datasets" is called **learning** or **training**. The "models" are also called as *learners*, or *hypotheses*.

Based on the existence of *labels* for examples in the dataset, Machine Learning could be roughly divided as two major categories: **Supervised Learning** and **Unsupervised Learning**. Supervised Learning needs label information, where labels reflect the explicit distribution of the datasets, so that the model learnt could make good predictions for unseen examples. However, the goal of Unsupervised Learning is to extract implicit information from the datasets, making the labels are not necessary. For example, *Clustering* is a typical problem of Unsupervised Learning, which aims to separate "close" examples of a dataset into several *clusters*. In this thesis, we mainly consider Supervised Learning problems.

The mathematical notations used in this section majorly follow [Zhou 2021, Zhou 2012]. We also refer readers who are not familiar with Machine Learning to some other classical books, like [Mitchell 1997, Hastie *et al.* 2009]. For more general in Artificial Intelligence, we refer readers to [Russell & Norvig 2020].

1.2.1 Basic Knowledge

1.2.1.1 Classification and Regression

Supervised Learning could be separated into two types of problems: **Classification** for making *discrete* predictions, and **Regression** for making *continual* predictions. Especially, if the predictions are limited in two different discrete classes, we call this classification problem as **Binary Classification**. Normally, one class is called positive class, and the other is called negative class.

[2]Learn more details from https://maxsat-evaluations.github.io/.

A mathematical description for Supervised Learning is as follow. We note a labelled dataset (or a set of examples) containing $\boldsymbol{m}$ examples as $\boldsymbol{\mathcal{E} = \{e_1, e_2, \ldots, e_m\}}$. Each example, like the $\boldsymbol{i}$-th one $\boldsymbol{e_i}$, is a pair of $\boldsymbol{(x_i, y_i)}$, where $\boldsymbol{x_i} = (\boldsymbol{x_{i1}}; \boldsymbol{x_{i2}}; \ldots; \boldsymbol{x_{id}})$ is a feature vector of $\boldsymbol{d}$ features, the $\boldsymbol{x_{ij}}$ indicates the value of j-th feature for $\boldsymbol{x_i}$. The value $\boldsymbol{d}$ is the dimension, the corresponding *feature space* is noted as $\mathcal{X}$. We denote by $\boldsymbol{\mathcal{F} = \{f_1, \ldots, f_d\}}$ the set of features of the dataset. The value $\boldsymbol{y_i}$ is the label of $\boldsymbol{e_i}$, it could be a discrete class for classification, or a continual value for regression. We note $\mathcal{Y}$ as the ensemble of all possible labels, or *label space*. The (unknown) data distribution, denoted by $\boldsymbol{\mathcal{D}}$, as a *distribution* over the feature space $\mathcal{X}$, where the feature vectors $\boldsymbol{x_i}$ from $\mathcal{E}$ are *independently and identically distributed* from $\mathcal{D}$. That is, the feature vectors $\boldsymbol{x_i}$ share the same implicit data distribution $\mathcal{D}$, and do not influence each other. The difference between data distributions can be reflected in the different probabilities of the feature vectors. In practice, this difference is implied in the different weights of feature vectors in the dataset. In general, if not specially mentioned, the feature vectors in a dataset share the same weight, indicating they share the same probability.

The target of Supervised Learning is to find a mapping (i.e., a model) $\boldsymbol{\phi}$ from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$, by using the labelled dataset $\mathcal{E}$. We note $\phi : \mathcal{X} \mapsto \mathcal{Y}$, the $\phi_{(\boldsymbol{x_i})}$ is the prediction made for $\boldsymbol{x_i}$. The *learning algorithm* that provides the model is denoted by $\boldsymbol{\mathcal{L}}$. We also note $\mathcal{L}(\mathcal{E}, \mathcal{D})$ as the model learnt by the algorithm $\mathcal{L}$ from the dataset $\mathcal{E}$ drawn on the distribution $\mathcal{D}$.

For Binary Classification, normally we let $\mathcal{Y} = \{0, 1\}$. When $|\mathcal{Y}| > 2$, we call it as **Multi-class Classification**. When $\mathcal{Y} \subseteq \mathbb{R}$, it is Regression. In this thesis, we mainly consider the Classification problems, especially the Binary Classification problem.

1.2.1.2 Evaluation Measures

After a model ϕ is learnt from a labelled dataset $\mathcal{E} = \{(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_m}, y_m)\}$, for example $(\boldsymbol{x_i}, y_i)$, the prediction $\phi_{(\boldsymbol{x_i})} = y_i$ indicates the model predicts $\boldsymbol{x_i}$ correctly. Otherwise, the prediction is incorrect (or wrong).

We introduce two widely used measures for evaluating the prediction performance of a classification model: the ***accuracy***, and the ***error rate***. The accuracy is the percentage of examples correctly predicted in the whole dataset. In reverse, the error rate is the percentage of examples wrongly predicted. The accuracy is defined as follow using the notation at the beginning:

$$acc(\phi; \mathcal{E}) = \frac{1}{m} \sum_{i=1}^{m} count(\phi_{(\boldsymbol{x_i})} = y_i) \tag{1.1}$$

where $count(\cdot)$ is an indicator function, the value is 1 when $\cdot$ is *True*, or 0 when it is *False*. The error rate could be calculated as $\boldsymbol{err(\phi; \mathcal{E}) = 1 - acc(\phi; \mathcal{E})}$.

More generally, we call the differences between the predictions and the labels as error. We call the labelled dataset used for learning a model as ***training set***, the error of the model learnt on training set as ***training error***, or ***empirical error***.

The error of the model on new unseen examples is called ***generalization error***.

Apparently, we hope the model learnt has small generalization error as it reflects the general prediction performance. However, as we have no idea for the unseen examples, we could only minimize the empirical error. But frequently, a model learnt having small empirical error performs badly in unseen examples. This phenomenon is called ***overfitting***. One major reason of overfitting is the model learnt is so "strong" in the training set so that it considers some specific properties in the training set as general properties. The opposite phenomenon of overfitting is called ***underfitting***, indicating the model does not learn enough general properties from the training set. Compared to overfitting, underfitting is easy to avoid by reducing the empirical error. Overfitting is impossible to avoid entirely, we could only try to "reduce" its possibility.

As it is impossible to list all unseen examples to get the generalization error, we use some unseen examples in reasonable size as the ***testing set***, and the ***testing error*** as an approximation of generalization error. We assume that all examples in the testing set has the independently identical distribution to the training set. In addition, to get "closer" approximation to the generalization error, all examples in the training set are not recommended to appear in the testing set. However, in most cases, there is only one labelled dataset $\mathcal{E}$ but it needs make training and testing together. We introduce three widely used methods for splitting the training set and testing set: the *hold-out* method, the *cross-validation* method, and the *bootstrapping* method.

Hold-Out Method

The **Hold-Out** method directly separates the labelled dataset $\mathcal{E}$ into two exclusive subsets, the one is chosen as training set $\boldsymbol{\mathcal{E}_{train}}$, the other is chosen as testing set $\boldsymbol{\mathcal{E}_{test}}$, where $\mathcal{E}_{train} \cup \mathcal{E}_{test} = \mathcal{E}$, and $\mathcal{E}_{train} \cap \mathcal{E}_{test} = \emptyset$.

There are some attention points for the hold-out method. At first, we should choose a reasonable ratio for splitting the training and testing set, so that the testing set split is in *sufficient* size. In general, to guarantee the fidelity of the evaluation, the testing set split should contain at least 30 examples. Normally, the split ratio for training set varies from 2/3 to 4/5 [Mitchell 1997, Zhou 2021]. Moreover, we should also try to preserve the same distribution of training and testing sets. A common way is called *stratified sampling* that keeps the percentage of examples of all classes in the training and testing sets similar, or equal. But sometimes, *pure random selection* is also used as its effectiveness. Finally, to make the evaluation more stable, generally the splitting process is repeated several times, and the final training (testing) error is the average value of all training (testing) errors of each splitting process.

Sometimes in practice, the hold-out method separates the dataset into three exclusive subsets, the training set, the testing set, and the *validation set* $\boldsymbol{\mathcal{E}_{val}}$. The validation set is used during the training process, as a "substitute" of testing set

to estimate the performance of the learning model. The use of validation set could somehow avoid the underfitting, and reduce the possibility of overfitting.

Cross-Validation Method

The **Cross-Validation** method [Kohavi 1995] at first separates the labelled dataset $\mathcal{E}$ into k exclusive subsets with similar (or same) sizes. That is $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \cdots \cup \mathcal{E}_k$, where for $i \neq j$, it has $\mathcal{E}_i \cap \mathcal{E}_j = \emptyset$. For each subset $\mathcal{E}_i$, the method choose the left $k-1$ subsets as the training set, and $\mathcal{E}_i$ as the testing set. Then, the method generates k pairs of training-testing sets for the evaluation. The final training (testing) error is the average value of the training (testing) errors in k times.

As the cross-validation method strongly relied on the value of k, it is also called as ***k*-fold cross-validation**. The value of k is widely chosen as 5, or 10, considering the balance of computational time. In addition, similar to the hold-out method, to make the evaluation more stable, sometimes, cross-validation is also repeated several times with different random seeds to get the average values.

Cross-validation method is one of the most popular validation method. In this thesis, 5-fold cross-validation is widely used in the experiments.

Bootstrapping Method

The **Bootstrapping** method [Efron & Tibshirani 1993] is based on the *bootstrap sampling*, which is the *sampling with replacement*. In detail, for a labelled dataset $\mathcal{E}$ of size m, the method uses the m times of sampling with replacement to generate a new labelled dataset $\mathcal{E}'$ of size m. For example, if an example e_i from $\mathcal{E}$ is chosen in one time, the method copies the example into $\mathcal{E}'$, and then puts this example back to $\mathcal{E}$ to let it possibly chosen in the future.

Apparently, some examples in the original dataset $\mathcal{E}$ are appeared multiple times in $\mathcal{E}'$, and some examples never. The probability that an example is never chosen within m times is $(1 - \frac{1}{m})^m$, we have

$$\lim_{m \to \infty} (1 - \frac{1}{m})^m = \frac{1}{e} \approx 0.368 \tag{1.2}$$

It indicates nearly a third of examples in $\mathcal{E}$ are never chosen in $\mathcal{E}'$. Therefore, we could use the new dataset $\mathcal{E}'$ as the training set, and the set of left examples $\mathcal{E} \setminus \mathcal{E}'$ as the testing set. This approach is called as ***out-of-bag estimate***.

The bootstrapping method is useful for those datasets in small sizes. In addition, as it could generates multiple different training set, it is also widely used in *Ensemble Learning*, like *Bagging*, *Random Forest*. However, when the dataset is in sufficient size, the hold-out method and the cross-validation method are more recommended, as the bootstrapping method changes the distribution of the dataset.

1.2.2 Interpretable Machine Learning Models

In the last decade, with the great advance in the computational resources, booming effective accurate Machine Learning algorithms are proposed. Especially in the field of **Neural Networks**, like the ***AlexNet*** [Krizhevsky *et al.* 2012], the ***Residual-Net*** [He *et al.* 2016], the ***GAN*** [Goodfellow *et al.* 2014], the ***Transformer*** [Vaswani *et al.* 2017], the ***Informer*** [Zhou *et al.* 2021], etc; and the field of **Ensemble Learning**, like the ***XGBoost*** [Chen & Guestrin 2016], the ***Lightgbm*** [Ke *et al.* 2017], the ***Deep Forest*** [Zhou & Feng 2017], etc. These algorithms have big successes in many important sub-fields of Machine Learning, like *Computer Vision*, *Natural Language Processing*, *Video Processing*, etc.

However, the models learnt via these algorithms are called as "***black-box models***" as they lack of ***interpretability***. There are different (non-mathematical) definitions of interpretability [Miller 2019, Kim *et al.* 2016], the core idea is that, with better interpretability, human could more easily understand why certain decisions or predictions are made by the model learnt.

Recently, there are explosion of works in Machine Learning for better interpretability. They could be classified as two major categories [Molnar 2022]: ***Intrinsic***, or ***Post hoc***. The intrinsic interpretability refers to the ML models that are considered to be interpretable due to their simple structure, like short decision trees. The post hoc interpretability refers to the application of interpretation methods after the black-box models learnt. It creates a second model to explain the black-box model trained. For example, extracting a decision tree for explaining the predictions made by a trained neural network. The methods of post hoc are also called as ***explainable ML***. The author of [Rudin 2019] shows some weaknesses of the explainable ML, and declares the intrinsic models as **Interpretable Machine Learning Models**. We follow this definition.

In this thesis, we only focus on Interpretable Machine Learning Models for binary classification. We introduce some widely-used interpretable ML models as follow, like *Decision Trees*, *Binary Decision Diagrams*, and other popular models, including *Decision List* and *Decision Set*.

1.2.2.1 Decision Trees

Decision Tree (DT) is one of the most popular ML model in supervised learning. The decision process of decision tree is similar as human: make the decision by a series of judgements. This simply logical decision process makes the decision tree intrinsically interpretable.

In general, a decision tree topology is shown in Figure 1.1, which contains a *root node*, several *branching nodes*, and *leaf nodes*. For each branching node (or the root node), a feature is selected as the test for the judgement. Depending on the problem, the leaf node corresponds to a class for classification problem, or to a real value for regression problem. The number of subtrees for a branching node (or the root node), equals to the number of different values of the feature selected. In addition, each subtree indicates a value case of the feature selected. Each path

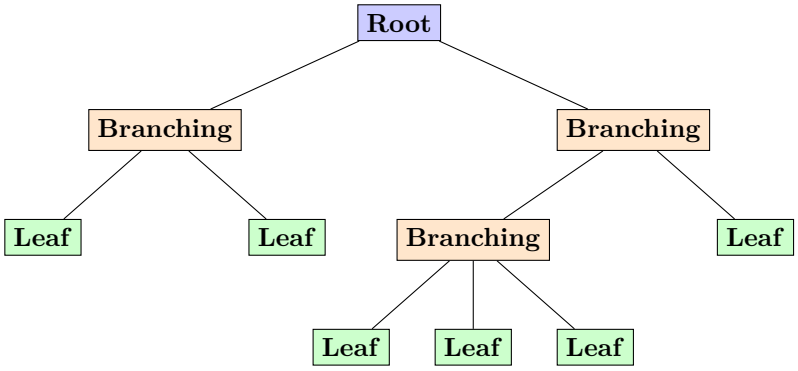


Figure 1.1: A general Decision Tree topology

from the root node to a leaf node corresponds to a series of judgements. To make the prediction for an unseen example, finding the corresponding path from the root node to a leaf node based its values for the series of tests, and the prediction is the value associated to the leaf node.

An illustrating example of decision tree for classification is shown in Figure 1.2, learnt from a small dataset shown in Table 1.1. The small dataset concerning a family go out to play based on the weather, comes from [Quinlan 1993]. It is easy to check that the decision tree in Figure 1.2 classifies all examples correctly.

Outlook	**Temp**($>26°$**C?**)	**Humidity**($>75\%$**?**)	**Windy?**	**Play?**
sunny	false	false	true	Yes
sunny	true	true	true	No
sunny	true	true	false	No
sunny	false	true	false	No
sunny	false	false	false	Yes
overcast	false	true	true	Yes
overcast	true	true	false	Yes
overcast	false	false	true	Yes
overcast	true	false	false	Yes
rain	false	true	true	No
rain	false	false	true	No
rain	false	true	false	Yes

Table 1.1: A small dataset from [Quinlan 1993]

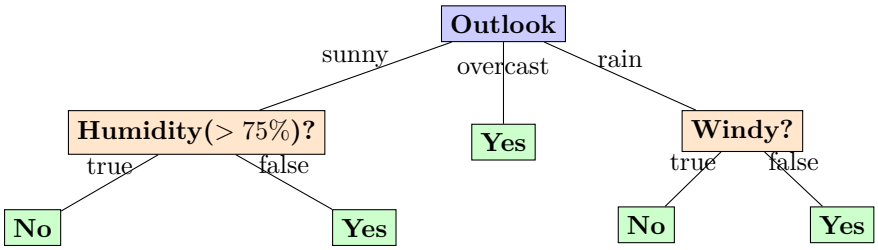


Figure 1.2: An illustrating example of Decision Tree

The traditional way to learn decision trees is *top-down induction* based on the divide and conquer methodology. Briefly, each node corresponds to a subset of dataset, where the root node corresponds to the entire dataset. For branching nodes (or the root node), the corresponding subset is split into different pieces for different subtrees based on the feature associated. The leaf nodes end the split, when all examples in the subset have the same class, or the subset contains no examples, or all features in the datasets are used. Detailed algorithm is shown in the Algorithm 2.

Algorithm 2: The traditional Decision Tree algorithm by top-down induction.

Input: A labelled Dataset $\mathcal{E} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_m}, y_m)\}$;
The feature set of dataset $\mathcal{F} = \{f_1, \ldots, f_d\}$;

1 **Process** *DTGenerate*($\mathcal{E}$, $\mathcal{F}$)**:**
2 *Generate a tree node* ***node*** *for* $\mathcal{E}$.
// When all examples have same class, end the split
3 **if** *all examples in* $\mathcal{E}$ *has same class* c **then**
4 ***node*** *is marked as leaf node with class* c.
5 **return**.
// When there is no candidate feature, end the split
6 **if** $\mathcal{F} = \emptyset$ **then**
7 **return**.
8 *Choose the best split feature* f_* *from* $\mathcal{F}$.
9 **for** *each possible value* f_*^v *of* f_* **do**
10 *Generate a subtree for* ***node***.
11 $\mathcal{E}^v \leftarrow$ *the subset of examples in* $\mathcal{E}$ *with value* f_*^v *of feature* f_*.
12 **if** $\mathcal{E}^v = \emptyset$ **then**
// When no examples lead to this value
13 *The subtree is marked as leaf node with majority class in* $\mathcal{E}$.
14 **return**.
15 **else**
16 *Use the* ***node*** *from* $DTGenerate(\mathcal{E}^v, \mathcal{F} \setminus f_*)$ *as the subtree.*

Output: *A Decision Tree with* ***node*** *as the root.*

There are various famous heuristic decision tree algorithms, like **C4.5** [Quinlan 1993], **CART***(Classification and Regression Tree)* [Breiman *et al.* 1984], etc. The differences between them is the heuristic of choosing the best split feature (line 8 in Algorithm 2). We show details of the heuristic used in *C4.5* and *CART* in the section 1.3.1.1 .

There are also variants of the traditional decision tree algorithm. For example, unlike using a single feature to make the split, **Multivariate Decision Trees** [Murthy *et al.* 1993, Brodley & Utgoff 1995] proposed using a combination of features to achieve more complex topology. And some variants [Utgoff 1988, Utgoff *et al.* 1997] proposed applying incremental learning in the decision trees by

reconstruct partly the topology for new-coming data not learn a new model, to reduce the training time.

1.2.2.2 Binary Decision Diagrams

Binary Decision Diagram (BDD) is another interpretable ML model in supervised learning. Especially, it could only used in binary classification with dataset full of binary features. The decision process of binary decision diagram is same as the decision tree by a series of judgements.

As their compact representations for Boolean functions, binary decision diagrams are widely studied in hardware design, model checking, and knowledge representation [Akers 1978, Moret 1982, Bryant 1986, Knuth 2009]. Considering a sequence of Boolean variables $[\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}]$, a binary decision diagram is a *rooted*, *directed*, *acyclic* graph. It contains two types of vertices. A *terminal vertex* v is associated to a binary value: $value(v) \in \{0, 1\}$. A *nonterminal vertex* v is associated to a Boolean variable x_i and has exact two children: $left(v)$, $right(v)$. Its children are vertices too, and $index(v) \in \{1, \ldots, n\}$ is the index of the Boolean variable associated to v.

To guarantee a unique binary decision diagram for a given Boolean function, two restrictions are widely assumed: *ordered* and *reduced*. The restriction "ordered" indicates for any nonterminal vertex v, it has $index(v) < index(left(v))$ and $index(v) < index(right(v))$. The restriction "reduced" indicates the graph contains no nonterminal vertex v with $left(v) = right(v)$, nor does it contains distinct nonterminal vertices having isomorphic rooted sub-graphs. The Boolean function represented by the binary decision diagram can be recursively obtained with the Shannon expansion process [Shannon 1938]. That is, for an ordered reduced binary decision diagram defined in the sequence of Boolean variables $[\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}]$ having v as the root node, the Boolean function g_v is:

1. If v is a terminal vertex:
   $g_v = value(v)$.

2. If v is a nonterminal vertex with $index(v) = i$:

$$g_v(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}) = \neg \mathrm{x}_\mathrm{i} \cdot g_{left(v)}(\mathrm{x}_1, \ldots, \mathrm{x}_{\mathrm{i}-1}, \mathrm{x}_{\mathrm{i}+1}, \ldots, \mathrm{x}_\mathrm{n}) + \mathrm{x}_\mathrm{i} \cdot g_{right(v)}(\mathrm{x}_1, \ldots, \mathrm{x}_{\mathrm{i}-1}, \mathrm{x}_{\mathrm{i}+1}, \ldots, \mathrm{x}_\mathrm{n}).$$

An example of binary decision diagram is shown in Figure 1.3, where *dashed*(*solid*) line of each vertex indicates the left(right) child. The Boolean function represented is $g(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) = (\mathrm{x}_1 \vee \mathrm{x}_2) \wedge (\mathrm{x}_1 \vee \mathrm{x}_3) \wedge (\mathrm{x}_2 \vee \mathrm{x}_3)$.

We define the *depth* of the binary decision diagram as the number of nonterminal vertices of longest path from the root to a terminal node. It is clear that length of the sequence of Boolean variables is equal or greater than the depth. Therefore,

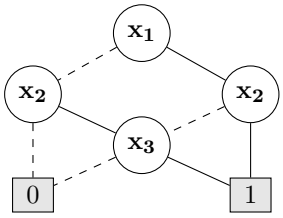


Figure 1.3: An illustrating example of Binary Decision Diagram of depth 3.

the length of the sequence of Boolean variables is equivalent to the *maximum depth* of the binary decision diagram.

To make the binary classification, the sequence of Boolean variables used in the binary decision diagram is changed as a sequence of binary features with the same length. The nonterminal vertices are associated to binary features, and the terminal vertices are associated to binary classes. As there are exact two terminal vertices, they are also called as *sink nodes*. There are several ways to transform the multi-value features of the datasets into binary features. A general method we applied in this thesis is called "one-hot encoding", which replaces a multi-value feature with several new binary features. Each new binary feature indicates does the original multi-value feature equals to a value. Table 1.2 shows the transformed dataset by the one-hot encoding in the Table 1.1. To simplify the representation, the value *true* ($false$) in the feature, and Yes (No) in the class, are transformed into 1 (0).

f_1	f_2	f_3	f_4	f_5	f_6	**Play?**
1	0	0	0	0	1	1
1	0	0	1	1	1	0
1	0	0	1	1	0	0
1	0	0	0	1	0	0
1	0	0	0	0	0	1
0	1	0	0	1	1	1
0	1	0	1	1	0	1
0	1	0	0	0	1	1
0	1	0	1	0	0	1
0	0	1	0	1	1	0
0	0	1	0	0	1	0
0	0	1	0	1	0	1

f_*	**Feature Implied**
f_1	**Outlook = sunny?**
f_2	**Outlook = overcast?**
f_3	**Outlook = rain?**
f_4	**Temp(>26°C?)**
f_5	**Humidity(>** 75%**?)**
f_6	**Windy?**
0	**false** or **No**
1	**true** or **Yes**

Table 1.2: The transformed binary dataset by one-hot encoding from Table 1.1.

The binary decision diagram of maximum depth 4 classifying all examples in Table 1.2 correctly is shown in Figure 1.4. In addition, the binary decision tree corresponding to this binary decision diagram is shown in Figure 1.5. It is clear that the size (number of vertices) of the binary decision diagram is smaller than the size of the corresponding binary decision tree.

Compared to the decision tree, the binary decision diagram could avoid the *replication* problem and the *fragmentation* problem effectively [Oliver 1992, Ko-

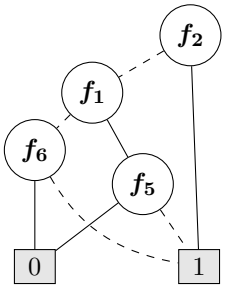


Figure 1.4: The Binary Decision Diagram of max depth 4 classifying all examples.

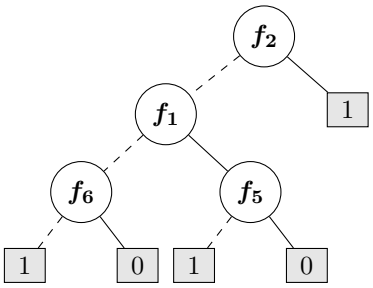


Figure 1.5: The Decision Tree for the dataset transformed in Table 1.2.

havi 1994], which are two major flaws suffered by the decision tree [Matheus & Rendell 1989, Pagallo & Haussler 1990, Rokach & Maimon 2014]. The replication problem appears when two identical subtrees are in the decision tree. The fragmentation problem appears when only few examples are associated to leave nodes. The restriction "reduced" of binary decision diagram could avoid the replication problem entirely, indicating generally the binary decision diagram has more compact topology than the corresponding decision tree in size. To avoid the fragmentation problem, [Kohavi & Li 1995] proposed a post-process of merging compatible subtrees. We show details in the section 1.3.2.

In addition, binary decision diagrams are extended for multi-classification, known as *decision graphs*. There are some heuristic algorithms are proposed to learn a decision graph from a dataset. We show details of some heuristic in the section 1.3.2.

1.2.2.3 Some Other Models

Except the decision tree, binary decision diagram, there are also some other Interpretable ML models. Here we present two of them, the *decision list*, and the *decision set* model.

Decision List was firstly introduced in [Rivest 1987]. The model contains a list of distinct *rules*, by following a ***if - then - else if -*** ... ***- else*** relationship. Each rule is in the format of "$\boldsymbol{\pi_i \Rightarrow k_i}$", where $k_i \in \mathcal{Y}$, corresponding to a conditional statement: "if the predicate π_i is satisfied for an example, then the class predicted is k_i". The size of decision list is the number of the distinct rules it contains.

The Figure 1.6 shows an example of decision list for the small dataset in Table 1.2. The model is comprehensive for human as its logical processes.

IF	$f_2 = 1$	**THEN**	**Play=1**
ELSE IF	$f_1 = 0 \wedge f_6 = 0$	**THEN**	**Play=0**
ELSE IF	$f_1 = 1 \wedge f_5 = 0$	**THEN**	**Play=0**
		ELSE	**Play=1**

Figure 1.6: The *Decision List* for the small dataset in Table 1.2.

Decision Set is another rule-based Interpretable ML model, which was first appeared in [Rivest 1987] as an *unordered* variant of decision list. That is, unlike the decision list relates the rules by the logic ***if - then - else if -*** ... ***- else***, the rules in the decision set are all independent. The Figure 1.7 shows the decision set for the small dataset in Table 1.2.

IF	$f_2 = 1$	**THEN**	**Play=1**
IF	$f_2 = 0 \wedge f_1 = 0 \wedge f_6 = 0$	**THEN**	**Play=0**
IF	$f_2 = 0 \wedge f_1 = 0 \wedge f_6 = 1$	**THEN**	**Play=1**
IF	$f_2 = 0 \wedge f_1 = 1 \wedge f_5 = 0$	**THEN**	**Play=0**
IF	$f_2 = 0 \wedge f_1 = 1 \wedge f_5 = 1$	**THEN**	**Play=1**

Figure 1.7: The *Decision Set* for the small dataset in Table 1.2.

It is easy to transform the decision tree or the binary decision diagram into the decision set, as each path from the root to the leaf node relates to a rule in the decision set. As the rules are unordered, some rules may overlap. That is, for an example, there are may be multiple rules are satisfied. In this case, there are two choices, the one is to apply a tie-break rule to pick the class [Lakkaraju *et al.* 2016], the other is to declare an overlap [Ignatiev *et al.* 2018]. For those examples that none of the rules is satisfied, the decision set could apply the default rule for the prediction [Lakkaraju *et al.* 2016].

1.2.3 Ensemble Methods

Ensemble Methods are a set of algorithms in supervised learning, aim to train multiple learners and combine them to make predictions [Zhou 2012]. A common architecture of Ensemble Methods is shown in Figure 1.8. The multiple learners are called **base learners**, and each one is learnt from training data by a base learning algorithm, like decision tree, neural network, etc. In general, an ensemble method uses a single base learning algorithm to generate base learners, which is called ***homogeneous ensemble***. Otherwise, when an ensemble method uses multiple base learning algorithms, it is called ***heterogeneous ensemble***. In this thesis, we mainly introduce the homogeneous ensembles.

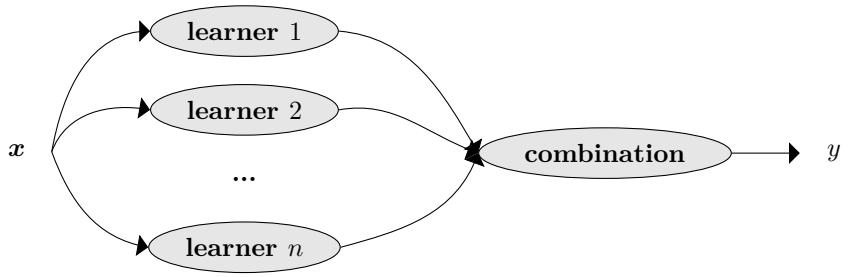


Figure 1.8: A common architecture of Ensemble Methods. [Zhou 2012]

1.2.3.1 The Combination Methods

Ensemble Methods could improve the generalization performance of the base learners. In particular, Ensemble Methods could boost *weak learners* to *strong learners* with good generalizations, even those weak learners are just slightly better than random guess. The key reason of this improvement due to the combination methods. [Dietterich 2000a] attributed three benefits from the combination:

- *Statistical Issue*: It is often that the training data is limited compared to the feature space. And there may be several models (hypotheses) with same accuracy on the training data. The combination of the hypotheses could reduce the risk of wrongly choosing the best one.
- *Computational Issue*: It is often that many learning algorithms are stuck in local optima. The combination of the hypotheses could reduce the risk of leading to a wrong local minimum.
- *Representation Issue*: The combination of the hypotheses may be possible to expand the space of representable functions, and form a more accurate approximation to the true unknown hypothesis.

For numerical outputs, *Averaging* is the most popular combination method; For discrete outputs, *Voting* is the most widely used combination method. In this thesis, we use voting since we mainly consider the classification problem. There are mainly three different voting methods: the *majority voting*, the *plurality voting*, and the *weighted voting*. We introduce a mathematical description for Ensemble Methods before presenting them.

In general, for a multi-classification task with the label space $\mathcal{Y} = \{c_1, \ldots, c_N\}$, we consider an ensemble containing $\boldsymbol{\mathcal{T}}$ base learners. And, we note the i-th base learner as $\boldsymbol{\phi_i}$, the ensemble model as $\boldsymbol{\Phi}$. For a given feature vector $\boldsymbol{x}$, $\phi_i(\boldsymbol{x})$ is the prediction made by the i-th base learner, and $\Phi(\boldsymbol{x})$ is the prediction made by the ensemble after the combination. In addition, $\phi_i^j(\boldsymbol{x})$ is 1 if $\phi_i(\boldsymbol{x}) = c_j$, otherwise 0.

The majority voting chooses the class received more than half votes (each base learner has one vote), or rejects if none of class received more than half votes.

$$\Phi(\boldsymbol{x}) = \begin{cases} c_j, & if \quad \sum_{i=1}^{\mathcal{T}} \phi_i^j(\boldsymbol{x}) > \lfloor \mathcal{T}/2 + 1 \rfloor \\ \text{reject}, & otherwise \end{cases} \tag{1.3}$$

The plurality voting is less strict than the majority voting as it chooses the most voted class. There is no rejection option in plurality voting.

$$\Phi(\boldsymbol{x}) = c_{\arg\max_j \sum_{i=1}^{\mathcal{T}} \phi_i^j(\boldsymbol{x})} \tag{1.4}$$

The weighted voting is a weighted version of plurality voting. Different base learners have different weights, and the weighted voting finds the class that has the

highest highest weight. It is reasonable as it should give more power to stronger base learners in the voting. The weight of ϕ_i is denoted as $\boldsymbol{w_i}$.

$$\Phi(\boldsymbol{x}) = c_{\arg\max_j \sum_{i=1}^{\mathcal{T}} w_i \phi_i^j(\boldsymbol{x})} \tag{1.5}$$

In general, the weights are positive and normalized ($w_i \geq 0$ and $\sum_{i=1}^{\mathcal{T}} w_i = 1$) to realise the weighted averaging.

We shortly explain why the combination could boost weak learners to strong learners. Assuming the output of all base learners are *independent*, and each one makes a correct classification at the probability p. Therefore, the probability of the ensemble that using majority voting ($\boldsymbol{p_{mv}}$) is as follow, by guaranteeing at least $\lfloor \mathcal{T}/2 + 1 \rfloor$ base learners make correct classifications [Hansen & Salamon 1990]:

$$p_{mv} = \sum_{k=\lfloor \mathcal{T}/2+1 \rfloor}^{\mathcal{T}} \binom{\mathcal{T}}{k} p^k (1-p)^{\mathcal{T}-k} \tag{1.6}$$

From [Lam & Suen 1997], when $p > 0.5$, p_{mv} is monotonically increasing in $\mathcal{T}$, and $\lim_{\mathcal{T}\to\infty} p_{mv} = 1$; when $p < 0.5$, p_{mv} is monotonically decreasing in $\mathcal{T}$, and $\lim_{\mathcal{T}\to\infty} p_{mv} = 0$; when $p = 0.5$, $p_{mv} = 0.5$ for any $\mathcal{T}$.

In practice, the assume that all base learners are independent is not possible. In general, the base learners are highly related as they are trained on the same problem. Therefore, to generate an ensemble with good prediction performance, it needs not only accurate base learners, but also *diverse* learners. In fact, one core research of the Ensemble Methods is, how to find a good trade-off between the accuracy and the diversity of the base learners [Zhou 2012, Brown *et al.* 2005, Tang *et al.* 2006].

There are two major paradigms of Ensemble Methods: The one is to generate base learners *sequentially*, the other is to generate base learners *in parallel*. The ***Boosting*** methods are the representative of the former one, and the ***Bagging*** methods and the ***Random Forest*** are the representative of the later one.

[Dietterich 2000b] provides an experimental comparison between *Bagging*, *Boosting*, and *Random Forest* based on decision trees. And [Rokach 2016] provides a literature review of recent ensemble methods based on decision trees.

1.2.3.2 Boosting

Boosting methods are a family of algorithms generating base learners sequentially. The general mechanism of Boosting is simple: at first, train a base learner with the given dataset. Then, adjust the data distribution of the given dataset by the prediction of the base learner, that is, increase the weights of examples wrongly predicted so that they have more attentions in the next iterations. Next, use the adjusted dataset to train a new base learner. Repeat this process until it arrives the number of iterations preset, and finally combine those base learners. The Algorithm 3 shows this general mechanism [Zhou 2012].

Algorithm 3: The general mechanism of Boosting methods.

Input: A labelled Dataset $\mathcal{E}$ in distribution $\mathcal{D}$;
A base learning algorithm $\mathcal{L}$;
The number of iterations $\mathcal{T}$.

$\mathcal{D}_1 = \mathcal{D}$. // Initialize the data distribution
for $t = 1, \ldots, \mathcal{T}$ **do**
  $\phi_t = \mathcal{L}(\mathcal{E}; \mathcal{D}_t)$. // Train a base learner from $\mathcal{E}$ under distribution $\mathcal{D}_t$
  $\varepsilon_t = err(\phi_t; \mathcal{E})$. // Calculate the error rate of ϕ_t
  $\mathcal{D}_{t+1} = Adjust\_Distribution(\mathcal{D}_t, \varepsilon_t)$. // Adjust the data distribution

Output: $\Phi(\boldsymbol{x}) = Combine(\{\phi_1(\boldsymbol{x}), \ldots, \phi_{\mathcal{T}}(\boldsymbol{x})\})$.

There are lots of Boosting methods, some famous of them are **AdaBoost**(*Adaptive Boosting*) [Freund & Schapire 1997, Friedman *et al.* 2000], **GBDT** (*Gradient-Boosted Decision Trees*) [Friedman 2001, Friedman 2002], **XGBooost**(*eXtreme Gradient Boosting*) [Chen & Guestrin 2016], etc. We introduce the *AdaBoost* in detail as it is the representative Boosting method.

Considering the binary classification on classes $\{-1, +1\}$, the AdaBoost from [Freund & Schapire 1997] is shown in Algorithm 4, where $sign(\cdot)$ is an sign function, the value is $+1$ when $\cdot > 0$, or -1 when $\cdot \leq 0$.

Algorithm 4: The AdaBoost Algorithm.

Input: A labelled Dataset $\mathcal{E} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_m}, y_m)\}$;
A base learning algorithm $\mathcal{L}$;
The number of iterations $\mathcal{T}$.

$\mathcal{D}_1 = 1/m$. // Initialize the distribution with same weight
for $t = 1, \ldots, \mathcal{T}$ **do**
  $\phi_t = \mathcal{L}(\mathcal{E}; \mathcal{D}_t)$. // Train the base learner in t-th round
  $\varepsilon_t = err(\phi_t; \mathcal{E})$. // Evaluate the error rate of ϕ_t
  // End the iteration if ϕ_t is weaker than random guess
  if $\varepsilon_t > 0.5$ **then**
    break.
  $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$. // α_t is the weight of ϕ_t in final combination
  // Increase the weights of examples wrongly predicted
  $\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t}{z_t} \cdot \exp\left(\alpha_t \cdot count(\phi_t(\boldsymbol{x_i}) \neq y_i)\right)$. // z_t is a normalization factor for the distribution

Output: $\Phi(\boldsymbol{x}) = sign\left(\sum_{t=1}^{\mathcal{T}} \alpha_t \phi_t(\boldsymbol{x})\right)$.

The *AdaBoost* algorithm uses additive weighted voting for the final prediction, where the base learners with better prediction performances gain higher weights in the final voting. In addition, in the *AdaBoost* algorithm, the base learner trained in t-th iteration will influence the base learner in the next iteration by updating the data distribution. That is, the algorithm increases the weights of examples wrongly predicted by the current base learner, making those examples have more chance to

be correctly predicted in future iterations.

1.2.3.3 Bagging and Random Forest

As introduced before, the diversity between base learners in an ensemble method is important to achieve good prediction performance. For a given labelled dataset, a practical way to generate diverse base learners is to train base learners by different diverse subsets sampled from the dataset. Meanwhile, each subset should contain sufficient examples to avoid training base learner with poor prediction performance.

Bagging (*Bootstrap AGGregatING*) [Breiman 1996] adopts the bootstrap sampling [Efron & Tibshirani 1993] to generate different diverse subsets with duplication for training base learners. In detail, when a labelled dataset $\mathcal{E}$ containing m given examples, a subset containing m examples with duplication will be generated by sampling with replacement. This process is repeated $\mathcal{T}$ times to generate $\mathcal{T}$ different subsets. The *Bagging* algorithm uses those $\mathcal{T}$ subsets to train $\mathcal{T}$ base learners. Then, in classification, the algorithm combines those base learners by plurality voting. The *Bagging* is shown in the Algorithm 5.

Algorithm 5: The Bagging Algorithm.

Input: A labelled Dataset $\mathcal{E} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_m}, y_m)\}$;
A base learning algorithm $\mathcal{L}$;
The number of iterations $\mathcal{T}$.

1 **for** $t = 1, \ldots, \mathcal{T}$ **do**
  `// $\mathcal{D}_{bs}$ is the bootstrap distribution, indicating the subset of $\mathcal{E}$`
2 $\phi_t = \mathcal{L}(\mathcal{E}; \mathcal{D}_{bs})$.

Output: $\Phi(\boldsymbol{x}) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{\mathcal{T}} count(\phi_t(\boldsymbol{x}) = y)$.

Compared with Boosting methods, *Bagging* could train base learners in parallel. Moreover, unlike Boosting methods need be modified for multi-classification, *Bagging* could directly adapted in multi-classification, since the diversity comes from the different subsets generated by bootstrap sampling. There is another advantage of *Bagging* indicated by [Breiman 1996], the out-of-bag examples for estimating the generalization performance.

Random Forest [Breiman 2001] is an extension of *Bagging*. At first, Random Forest uses decision trees as the learning algorithm to generate base learners. Then, to increase the diversity between decision trees generated, the algorithm involves randomness in the feature selection. That is, during the construction of a base decision tree, in each split, unlike traditional methods using the whole features as candidate set, the algorithm randomly chooses a subset of features to choose the best feature for making the splits. The algorithm to generate a random base decision tree in the Random Forest follows the procedures of Algorithm 2, but instead of choosing the best split feature from the whole feature set of the dataset (the line

8), it uses a subset containing K features selected randomly from the whole feature set. The K is a preset parameter that imports the randomness.

The term *forest* was first introduced by [Ho 1995]. There are also lots of variants of generating random trees with different measures of randomness, some popular are ***Extremely Randomized Trees*** [Geurts *et al.* 2006], ***Rotation Forest*** [Rodríguez *et al.* 2006], etc.

1.3 Related Works in Interpretable ML Models

In this section, we provide a literature review of several methods for computing two Interpretable ML models: decision trees and decision graphs. For each ML model, we present the traditional heuristic methods and the recent exact methods. Traditional heuristic methods still obtain great interest due to their scalability. The exact methods for Interpretable ML models offer guarantee of mathematical optimality, for instance on model size or prediction error, and received lots of interest in recent papers. In this section, we focus on those exact methods using *combinatorial optimization* approaches.

1.3.1 Related Methods for Decision Trees

In this section, we introduce standard heuristic algorithms and several exact combinatorial optimization approaches.

1.3.1.1 Traditional Heuristic Algorithms

We present here some traditional top-down heuristic algorithms for decision trees. The core difference between them is the heuristic of choosing the best feature to split the dataset.

ID3 and C4.5 Algorithms

The methods **C4.5** [Quinlan 1993] and **ID3** [Quinlan 1986] are classical heuristic algorithms to learn decision trees. They are based on top-down induction, as shown in Algorithm 2. The heuristic used in these methods to choose the best split feature (line 8 in Algorithm 2) is based on the *Gain Ratio*. We explain this concept step by step.

At first, we introduce the concept of *information entropy* [Shannon 1948], which is based on the fraction of the different labels of the examples. Considering a labelled dataset $\mathcal{E}$, we note the *rate* (in percent) of examples with class c_j ($j = 1, \ldots, |\mathcal{Y}|$) as per_j, then the information entropy for $\mathcal{E}$ is defined as follow.

$$Ent(\mathcal{E}) = -\sum_{j=1}^{|\mathcal{Y}|} per_j \log_2 per_j \tag{1.7}$$

The information entropy $Ent(\mathcal{E})$ is used to measure the *purity* of a given example set. When $Ent(\mathcal{E})$ has smaller value, the dataset $\mathcal{E}$ is purer, indicating more

examples share same labels. For example, when all examples in $\mathcal{E}$ have same class, then $per_1 = 1$, leading $Ent(\mathcal{E}) = 0$. Otherwise, the information entropy is positive.

From Algorithm 2, in general, we hope that the subsets of examples become purer after the split by the feature selected. Therefore, assuming the feature selected to make the split is f_*, which has V different possible values $\{f_*^1, \ldots, f_*^V\}$. As in Algorithm 2, we use $\mathcal{E}^v$ to denote the subset of examples in $\mathcal{E}$ having the value f_*^v for the feature f_*. The *information gain* of using feature f_* as the split for dataset $\mathcal{E}$ is defined as follow.

$$Gain(\mathcal{E}, f_*) = Ent(\mathcal{E}) - \sum_{v=1}^{V} \frac{|\mathcal{E}^v|}{|\mathcal{E}|} Ent(\mathcal{E}^v) \tag{1.8}$$

The information gain uses the factor $|\mathcal{E}^v|/|\mathcal{E}|$ to reflect that the subsets split with more examples have more influences. In general, the bigger value of information gain indicates the bigger improvement in the purity when using f_* as the split feature. Therefore, using the feature with the biggest value in information gain is an useful heuristic to choose the best split feature f, which is $f = \arg\max_{f \in \mathcal{F}} Gain(\mathcal{E}, f)$.

The *ID3* algorithm [Quinlan 1986] applies this heuristic to select the best feature. However, the information gain prefers those features with more different values, which could easily cause the problem of overfitting. To reduce this disadvantage, the *C4.5* algorithm proposes the concept called *Gain Ratio*, which is a variant of the heuristic information gain. It is defined as follow.

$$\begin{aligned} Gain\_ratio(\mathcal{E}, f_*) &= \frac{Gain(\mathcal{E}, f_*)}{IV(f_*)} \\ IV(f_*) &= -\sum_{v=1}^{V} \frac{|\mathcal{E}^v|}{|\mathcal{E}|} \log_2 \frac{|\mathcal{E}^v|}{|\mathcal{E}|} \end{aligned} \tag{1.9}$$

where the $IV(f_*)$ is called the *intrinsic value* of the feature f_*. When f_* has more values, the value $IV(f_*)$ is generally bigger. The *C4.5* algorithm choose the best split feature f, that $f = \arg\max_{f \in \mathcal{F}} Gain\_ratio(\mathcal{E}, f)$.

We show how the heuristic is applied for the toy dataset $\mathcal{E}$ given in Table 1.2. The information entropy of $\mathcal{E}$ is:

$$Ent(\mathcal{E}) = -\sum_{j=0}^{1} per_j \log_2 per_j = -(\frac{5}{12} \log_2 \frac{5}{12} + \frac{7}{12} \log_2 \frac{7}{12}) = 0.9799$$

The information gain of each possible feature for the dataset $\mathcal{E}$ are as follow.

$$\begin{array}{lll} Gain(\mathcal{E}, f_1) = 0.0718 & Gain(\mathcal{E}, f_2) = 0.3436 & Gain(\mathcal{E}, f_3) = 0.0616 \\ Gain(\mathcal{E}, f_4) = 0.0102 & Gain(\mathcal{E}, f_5) = 0.1043 & Gain(\mathcal{E}, f_6) = 0.0207 \end{array}$$

Therefore, the first best feature to split is f_2. Repeat this process as the *ID3*

algorithm propose, the decision tree found is shown in Figure 1.9.

As the dataset in Table 1.2 contains only binary features, there are no great influences of using the factor of intrinsic values. The decision tree found by using the Gain Ratio proposed in the *C4.5* algorithm is same as the *ID3* algorithm, as the Figure 1.9 shown.

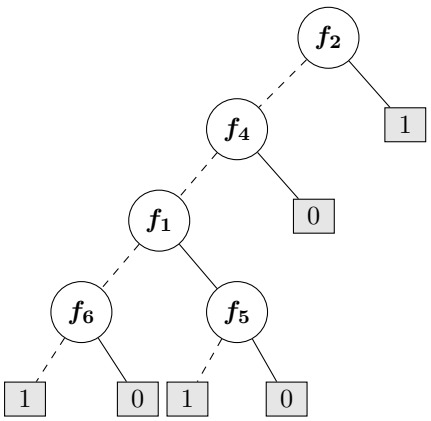


Figure 1.9: The Decision Tree for the dataset in Table 1.2 by *ID3* and *C4.5*.

w	f_1	f_2	f_3	f_4	f_5	f_6	**Play?**
1	1	0	0	0	0	1	1
3	1	0	0	1	1	1	0
2	1	0	0	1	1	0	0
3	1	0	0	0	1	0	0
1	1	0	0	0	0	0	1
1	0	1	0	0	1	1	1
2	0	1	0	1	1	0	1
2	0	1	0	0	0	1	1
2	0	1	0	1	0	0	1
3	0	0	1	0	1	1	0
2	0	0	1	0	0	1	0
1	0	0	1	0	1	0	1

Table 1.3: A weighted binary dataset from Table 1.2.

The *ID3* and *C4.5* algorithm could be easily applied with different data distribution. The idea is when calculating the information entropy, the per_j does not represent the percentage of examples with class c_j, but the percentage of the sum of weights of examples for this class. We consider the toy example $\mathcal{E}$ with different weights for each example. The new weighted dataset, denoted $\mathcal{E}_w$, is shown in Table 1.3. Therefore, the per_0^w and per_1^w of $\mathcal{E}_w$ are calculated as follow:

$$per_0^w = \frac{3+2+3+3+2}{23} = \frac{13}{23}$$
$$per_1^w = \frac{1+1+1+2+2+2+1}{23} = \frac{10}{23}$$

The information entropy related is

$$Ent(\mathcal{E}_w) = -\sum_{j=0}^{1} per_j^w \log_2 per_j^w = -(\frac{13}{23}\log_2\frac{13}{23} + \frac{10}{23}\log_2\frac{10}{23}) = 0.9877$$

The decision tree found by the heuristic information gain (*ID3*) is shown in the left one of Figure 1.10, the heuristic gain ratio (*C4.5*) is shown in the right one.

CART (Classification and Regression Tree) Algorithm

The **CART** (Classification and Regression Tree) method [Breiman *et al.* 1984] is another classical heuristic algorithm to learn a decision tree. The heuristic used to

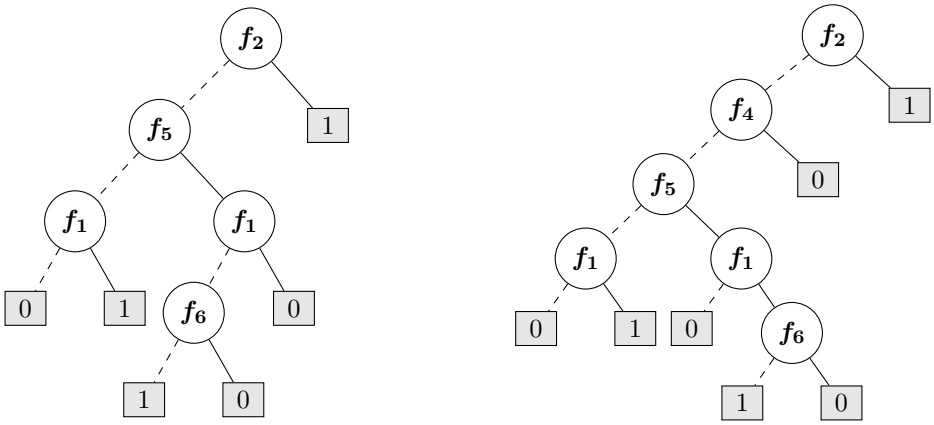


Figure 1.10: The Decision Tree for the weighted dataset in Table 1.3 by *ID3* (left) and *C4.5* (right).

choose the best split feature is based on the concept of *Gini Index* to measure the *purity* of a given example set.

Firstly, *CART* uses the concept *Gini Value*, which is defined as follow.

$$Gini(\mathcal{E}) = 1 - \sum_{j=1}^{|\mathcal{Y}|} per_j^2 \tag{1.10}$$

As for the information entropy, it is clear that when $Gini(\mathcal{E})$ has smaller values, the dataset $\mathcal{E}$ is purer. The *Gini Index* of using feature f_* as the split for dataset $\mathcal{E}$ is defined as follow by using the Gini Value.

$$Gini\_index(\mathcal{E}, f_*) = \sum_{v=1}^{V} \frac{|\mathcal{E}^v|}{|\mathcal{E}|} Gini(\mathcal{E}^v) \tag{1.11}$$

Similar as the information gain, the Gini Index considers the influence of the different sizes of subsets after split. The *CART* algorithm choose the best split feature f, which is $f = \arg\min_{f \in \mathcal{F}} Gini\_index(\mathcal{E}, f)$.

We consider the toy example $\mathcal{E}$ of Table 1.2. After computing the Gini Index, the best split feature for $\mathcal{E}$ is f_2 as it relates to the smallest value.

$$Gini\_index(\mathcal{E}, f_1) = 0.4095 \qquad Gini\_index(\mathcal{E}, f_2) = 0.2031$$
$$Gini\_index(\mathcal{E}, f_3) = 0.4431 \qquad Gini\_index(\mathcal{E}, f_4) = 0.4911$$
$$Gini\_index(\mathcal{E}, f_5) = 0.3844 \qquad Gini\_index(\mathcal{E}, f_6) = 0.4702$$

Repeat this process, the decision tree found is shown the left one in Figure 1.11. The *CART* method could be easily applied with different data distribution by changing the way of calculating the per_j values. The right one in Figure 1.11 is the decision tree found by *CART* algorithm for the weighted dataset $\mathcal{E}_w$ of Table 1.3.

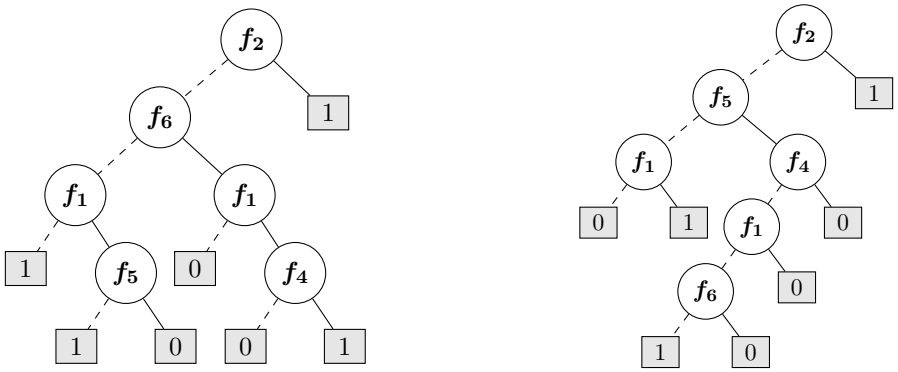


Figure 1.11: The Decision Tree for the toy dataset of Table 1.2 (the weighted dataset of Table 1.3) by *CART* is shown in the left (right) one.

1.3.1.2 Recent Exact Methods

Recently, there are increasing interests in exact methods to find the optimal decision trees, which is known as a NP-Hard problem [Hyafil & Rivest 1976]. In particular, it is essential to specify the goal of each exact method as each one needs an objective to optimize, indicating the metric of the optimal decision tree. The metrics widely used are *tree size*, *tree depth*, and *accuracy* leading to the three following goals.

- ***Goal 1 (tree size)**: For a given set of examples $\mathcal{E}$, find the Decision Tree with the smallest size (number of nodes) classifying all examples of $\mathcal{E}$ correctly.*
- ***Goal 2 (tree depth)**: For a given set of examples $\mathcal{E}$, find the Decision Tree with the smallest depth classifying all examples of $\mathcal{E}$ correctly.*
- ***Goal 3 (accuracy)**: For a given set of examples $\mathcal{E}$, find the Decision Tree restricted in topology (generally in depth) that maximizes the number of examples of $\mathcal{E}$ correctly classified.*

In this literature review, we focus on recent exact methods based on combinatorial optimization, including *Boolean Satisfiability*, *Maximum Boolean Satisfiability*, *Constraint Programming (CP)*, *Mixed Integer Programming (MIP)*, and *Dynamic Programming (DP)*. We separate this review on recent exact methods into two parts chronologically by our contribution [Hu *et al.* 2020]. In each part, we review those exact methods by the categories of combinatorial optimization method.

Depending on the different goals, ***SAT-based*** exact methods are divided into two families. At first, [Bessiere *et al.* 2009, Narodytska *et al.* 2018] consider the *Goal 1*, aiming to optimize the tree size. Then, [Avellaneda 2020, Janota & Morgado 2020] share the *Goal 2*, purposing to optimize the tree depth. In general, those methods deal with the *binary classification* and consider binarized datasets (with only *binary features*). The common process of SAT-based methods to find optimal decision trees in smallest tree sizes (respectively, tree depths) with perfect empirical accuracy, is based on a sequence of SAT queries. Each SAT query answers the existence of a

decision tree in given tree size (respectively, tree depth) with perfect accuracy. The sequence of SAT queries starts from the query of tree size (respectively, tree depth) of an upper bound received by heuristic methods, continues with the queries of decreasing tree sizes (respectively, tree depth), until the answer is unsatisfied, and the last decision tree found is the optimal one.

To reach the *Goal 1*, [Bessiere *et al.* 2009] proposes the fist SAT encoding for decision tree of fixed tree size. At first, the encoding captures the relationship between tree nodes and features by propositional variables. Then, it realise the perfect accuracy by ensuring that each pair of examples with opposite classes would not be lead to the same leaf node. This pioneering work suffers greatly from its formulation size, making it unrealistic to work for trees with more than fifteen nodes. After that, [Narodytska *et al.* 2018] proposes a new SAT encoding for decision tree of fixed tree size, which significantly reduces the encoding size. The reduction accounts majorly for the way of realising the perfect accuracy, where the new SAT encoding ensures all leaf nodes reject all examples with the opposite classes. The details of this encoding will be introduced in Section 2.2, as it plays an essential role in our contribution.

A motivation of the *Goal 2* is that compared to tree size, the metric tree depth could not only control the tree structure, but also provide better understanding by limiting the number of judgements. Additionally, it could avoid producing decision trees with small sizes but with high depths. [Avellaneda 2020] and [Janota & Morgado 2020] are two SAT encodings to model decision tree in given depth, which are proposed almost at same time. In [Avellaneda 2020], the author encodes a *full complete binary decision tree* in fixed depth. This pre-assumed tree structure benefits from its preset parent-child relationship. Therefore, this encoding reduces formulation size by eliminating variables and constraints to describe the connections between nodes. Analogously, in [Janota & Morgado 2020], the authors propose a SAT encoding for explicit paths of decision trees, which also contains no constraints to describe tree topology. This encoding restricts tree depth by the number of steps in each explicit path, meanwhile, tree size by the number of explicit paths.

The only ***CP-based*** exact method [Verhaeghe *et al.* 2020] focuses on the *Goal 3*, aiming to optimize empirical accuracy for depth-restricted decision trees. This method explores the left subtrees and right subtrees of a node in decision tree independently with a form of AND/OR search tree. In addition, to avoid searching equivalent subtrees, this method applies a caching system to store the optimal subtrees already found so that the search space could be reduced.

The ***MIP-based*** exact methods include *OCT* [Bertsimas & Dunn 2017], and *BinOCT* [Verwer & Zhang 2017, Verwer & Zhang 2019], where all of them focus on the *Goal 3* to optimize the misclassification. In [Bertsimas & Dunn 2017], the authors consider to learn *optimal classification trees (OCT)*, where the decision trees are assumed as multi-variate. The resulting decision trees were shown more accurate than heuristic trees. The main limitation of this method is the scalability, which is able to handle datasets with a few thousands examples. In [Verwer & Zhang 2017], the authors propose similar MIP approach to learn optimal classification trees, but

as well to learn optimal regression trees. The reductions in formulation size are achieved in *BinOCT* [Verwer & Zhang 2019] by applying binary linear program to learn optimal decision trees. Therefore, this method compute solutions much faster than the previous MIP methods.

The ***DP-based*** exact methods include *DL8* [Nijssen & Fromont 2007], *DL8.5* [Aglin *et al.* 2020], and *OSDT* [Hu *et al.* 2019], where all of them also focus on the *Goal 3* that minimize the empirical prediction error. In *DL8* [Nijssen & Fromont 2007], the algorithm extracts optimal decision trees by encapsulating the itemset lattices. The dynamic programming nature arises because once the split feature is determined, the optimal solutions of the left subtree and the right subtree are independent. To enforce the depth constraint in *DL8*, *DL8.5* [Aglin *et al.* 2020] is introduced. This algorithm uses a branch-and-bound search with caching to safely enumerate trees under the depth constraint. Both *DL8* and *DL8.5* have the drawback in the need of massive memory to deal with datasets in small sizes. A slightly different direction is to learn *Optimal Sparse Decision Tree (OSDT)* proposed in [Hu *et al.* 2019], where each node in the tree is considered as equal to some numbers of misclassifications. The aim of this method is to reach a balance between misclassification and number of nodes.

In our contribution [Hu *et al.* 2020], which is detailed in Chapter 2, we propose an original MaxSAT-based encoding extending the SAT encoding of [Narodytska *et al.* 2018], to reach the *Goal 3*, purposing to optimize the accuracy. Then, there are some new ***SAT*** or ***MaxSAT*** exact methods with different goals. For the *Goal 1*, [Alos *et al.* 2021] follows the principal encoding in [Narodytska *et al.* 2018], but directly finds the optimal decision tree in tree size via MaxSAT approach. Meanwhile, this method also proposes to generate multiple different decision trees in optimal size to avoid overfitting. However, slight improvements in generalization performance are observed from its experimental results. For the *Goal 2*, [Schidler & Szeider 2021] proposes a SAT-based local improvement method to optimize tree depth. The brief idea is to improve an heuristic decision tree iteratively, which replaces its subtrees with the subtrees of smaller depths with perfect empirical accuracy found by the SAT-based exact method. In addition, [Shati *et al.* 2021] proposes another novel SAT-based exact method to optimize tree depth which treats numeric features directly without binarized transformation. For the *Goal 3*, [Shati *et al.* 2021] extends their own SAT encoding for decision tree restricted in depth into MaxSAT formulation to optimize accuracy. The technique to generate soft clauses shares the same idea as our contribution. Next, a novel ***DP-based*** exact method named *MurTree* [Demirovic *et al.* 2022] is proposed to reach the *Goal 3*. Compared to previous DP-based methods, this method additionally introduces constraints on both depth and number of nodes to improve the scalability of decision tree optimization.

1.3.2 Related Methods for Decision Graphs

In this section, we introduce standard heuristic algorithms and several exact combinatorial optimization approaches for decision graphs.

1.3.2.1 Decision graphs

Decision graphs, also called decision diagrams, are proposed in [Oliver 1992] and are still studied in Machine Learning [Zhu & Shoaran 2021] and in Combinatorial Optimization [Bergman *et al.* 2016]. This model is introduced to face some limitations of decision trees, especially the replication and the fragmentation problem. Decision graph is a layered directed acyclic graph with a single node at the first layer (the root note) and one or several nodes at the last layer (sink nodes or leaves, for instance the labels of the dataset in machine learning context). Each internal node is associated to a variable (for instance a feature of the dataset). An arc from a node of a given layer to a node of a next level is valuated by the assignment of the variable associated to the originating node.

A standard variant of decision graph is binary decision diagram, where each internal node has only two successors). They are widely studied in hardware design and model checking [Bryant 1986], and are introduced with details in Section 1.2.2.2.. To sum up, binary decision diagrams allow to represent Boolean function and are adapted for binary classification on binary dataset. In addition, they are assumed as reduced and ordered, which are described in Section 1.2.2.2.

The **Oblivious Read-Once Decision Graph** or *OODG*, introduced in [Kohavi 1994], is a variant of decision graph with additional properties.

- The "*read-once*" property indicates that each feature is selected at most once along any path from the root to a category node.
- The "*levelled*" property indicates that the nodes are partitioned into a sequence of pairwise disjoint sets, representing the level. The outgoing edges from each level terminate at the next level.
- The "*oblivious*" property extends the "levelled" property by setting that all nodes at a given level are associated to the same feature.

From the definition of OODG, we can highlight some similarities and differences between it and ordered reduced binary decision diagram. For the similarities, OODG and binary decision diagram share the same topology. In detail, the category nodes and the branching nodes in the OODG model relate to the terminal vertices (leaves) and the non-terminal vertices in binary decision diagram; the combination of the "*read-once*" property and "*oblivious*" properties for OODG are the same as the "*ordered*" restriction for binary decision diagram. For the differences, the OODG is not limited in binary classification, as the leaves nodes in OODG could be associated to multiple values. In addition, the OODG considers multi-values

features by allowing multiple outgoing edges for a branching node. However, especially for binary classification for binary datasets, the OODG could be considered as *equivalent* to the binary decision diagram.

Algorithm 6: The algorithm to grow the Oblivious Decision Tree (ODT) in limited depth.

Input: A labelled Dataset $\mathcal{E} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_m}, y_m)\}$;
The feature set of dataset $\mathcal{F} = \{f_1, \ldots, f_d\}$;
The preset depth d.

```
// S is the set of sets of examples
```
1 $\mathcal{S} = \{\mathcal{E}\}$
2 *Create a tree node* ***node*** *related to* $\mathcal{E}$ *as the root.*
3 **while** $d \geq 0$ **do**
4 **if** $\mathcal{F} = \emptyset$ **then**
5 $d = 0$. **goto** line 8.
6 *Choose the best split feature* f_* *from* $\mathcal{F}$ *for* $\mathcal{S}$.
7 $\mathcal{S}' = \{\}$
8 **for** *each* $s \in \mathcal{S}$ **do**
9 **if** $s = \emptyset$ **then**
 `// The case of node captures no example`
10 *Label the node related to* s *as "unknown".*
11 **else if** $d = 0$ **then**
 `// The case of category node`
12 *Label the node related to* s *with the majority class of* s.
13 **else**
 `// The case of normal branching node`
14 *Label the node related to* s *with* f_*.
15 **for** *each possible value* f_*^v *of* f_* **do**
16 $s^v \leftarrow$ *the example subset of* s *with the value* f_*^v *of feature* f_*.
17 $\mathcal{S}' = \mathcal{S}' \cup \{s^v\}$.
18 *Create a node for* s^v *and link to the node related to* s *with* f_*^v.
19 $\mathcal{S} = \mathcal{S}'$. $\mathcal{F} = \mathcal{F} \setminus f_*$. $d = d - 1$.

Output: *An Oblivious Decision Tree with* ***node*** *as the root.*

1.3.2.2 Heuristic methods for OODG

To build an OODG for the classification, two heuristic methods: *bottom-up* induction [Kohavi 1994] and *top-down* induction [Kohavi & Li 1995], are proposed. The objective of the bottom-up induction is to build an OODG with no classification error from the bottom to the top. The heuristic is to choose the feature leading to the "*narrowest*" level in the number of branching nodes. In reverse, the top-down induction of OODG contains two critical phases. At first, growing an oblivious decision tree (ODT) by using the *mutual information* heuristic. Then, merging

the *isomorphic* and *compatible* subtrees in the ODT from top to down to build the OODG. Unlike the bottom-up induction, the top-down approach could control the depth of the OODG built as a preset parameter. We introduce the top-down induction in detail.

The first step of the top-down induction is to grow an oblivious decision tree (ODT) with heuristic. This process is similar to the traditional decision tree. The difference between them is the process of the traditional decision tree is *recursive*, but the process of the top-down induction for ODT is *iterative*. The Algorithm 6 shows how to grow an ODT in limited depth. The heuristic to choose the best split feature for a level (line 6 in Algorithm 6), is based on the *mutual information*.

To explain the concept of mutual information, we introduce the concept of *conditional entropy* [Cover & Thomas 2001] at first. Similar to the information entropy, conditional entropy is also used to measure the *amount of information* of a given example set after a sequence of features is selected. Considering a labelled dataset $\mathcal{E}$, the conditional entropy for $\mathcal{E}$ after the sequence of features $[f_1, \ldots, f_l]$ is selected is defined as follow:

$$\begin{aligned} H(\mathcal{E}|f_1, \ldots, f_l) &= -\sum_{f_1^* \in f_1, \ldots, f_l^* \in f_l, c_j \in \mathcal{Y}} \Delta \\ \Delta &= per(c_j, f_1^*, \ldots, f_l^*) \log_2 per(c_j|f_1^*, \ldots, f_l^*) \end{aligned} \tag{1.12}$$

where f_l^* indicates a general case of all possible values for f_l. The $per(c_j, f_1^*, \ldots, f_l^*)$ indicates the percentage of examples with label c_j that satisfy the assignments $f_1 = f_1^*, \ldots, f_l = f_l^*$ for the whole dataset. The $per(c_j|f_1^*, \ldots, f_l^*)$ indicates the percentage of examples with label c_j for all examples that satisfy the assignments $f_1 = f_1^*, \ldots, f_l = f_l^*$. Similar to the information entropy, the conditional entropy has *smaller* value when the sequence of features chosen making $\mathcal{E}$ *purer*. Given a new feature f_*, the *mutual information* is defined as the *difference* between the original conditional entropy and the updated conditional entropy with f_*:

$$I(\mathcal{E}; f_*|f_1, \ldots, f_l) = H(\mathcal{E}|f_1, \ldots, f_l) - H(\mathcal{E}|f_1, \ldots, f_l, f_*) \tag{1.13}$$

To choose the first feature (when $l = 0$), the mutual information is defined with the help of information entropy:

$$I(\mathcal{E}; f_*) = Ent(\mathcal{E}) - H(\mathcal{E}|f_*) \tag{1.14}$$

In general, the *bigger* value of mutual information indicates the *bigger* improvement in the purity when adding f_* into the existing sequence of features. Therefore, the heuristic to choose the best feature f into the existing feature sequence, which is $f = \arg\max_{f \in \mathcal{F}} I(\mathcal{E}; f|f_1, \ldots, f_l)$ in mathematical format.

Therefore, based on the heuristic of the mutual information, when the preset depth is 3, the sequence of feature of size 3 selected for the small dataset in Table 1.2 is $[f_2, f_4, f_1]$. The ODT of the depth 3 is shown in Figure 1.12, where the leaf nodes labelled "u" are the "*unknown*" nodes indicating they capture no example.

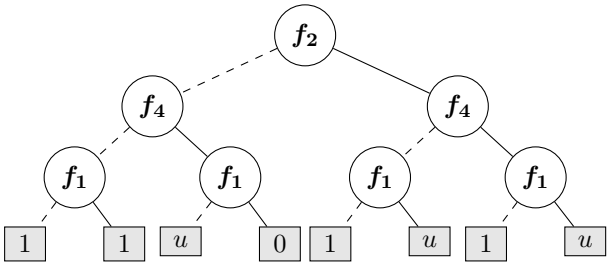


Figure 1.12: The Oblivious Decision Tree (ODT) for the small dataset in Table 1.2.

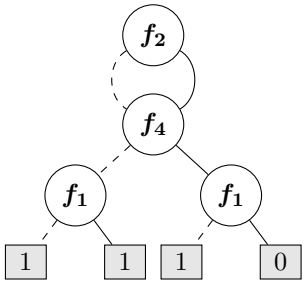


Figure 1.13: The ODT after merging the compatible subtrees of the root.

To build the OODG by merging subtrees of the ODT generated from top to down, we introduce the concept of *isomorphic* and *compatible* subtrees separately. Before judging two subtrees are isomorphic or compatible, there is an assume that the roots of them should be in the same level, indicating they are either branching nodes associated to the same feature, or category nodes.

Two subtrees are *isomorphic* if they are both category nodes with same class, or if the corresponding children are the roots of isomorphic subtrees. For example, in Figure 1.12, the two subtrees of the leftmost branching node associated to f_1 are isomorphic, same as the two subtrees of the rightmost branching node associated to f_4. Merging isomorphic subtrees reduces the size of the model without changing the bias of the original decision structure.

Two subtrees are *compatible* if either at least one root is labelled "*unknown*", or if the corresponding children are the roots of compatible subtrees. As the "*unknown*" nodes capture no example, they could match anything when we judge two subtrees are compatible. For example, in Figure 1.12 the two subtrees of the root are compatible, the ODT after merging them is shown in Figure 1.13. Merging compatible subtrees could help assign classes for those "*unknown*" nodes, which solves the fragmentation problem of the decision tree. This post-process changes the bias by assuming that they are likely to behave the same as the corresponding child in the compatible subtree.

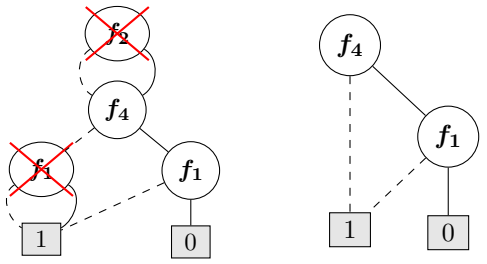


Figure 1.14: The OODG of depth 3 that merges isomorphic and compatible subtrees.

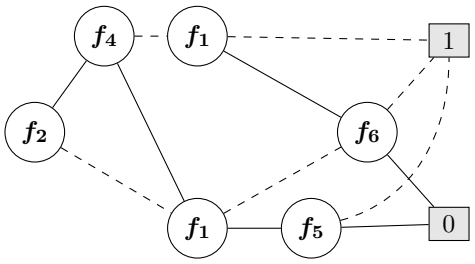


Figure 1.15: The OODG of depth 5 that classifies all examples correctly.

The corresponding OODG after merging isomorphic and compatible subtrees of the ODT in Figure 1.12 is shown as the left one in Figure 1.14. It is clear that

the OODG contains some *constant nodes*, such the node that all edges emanating from it terminate at the same node of the next level. For example, the root and the leftmost branching node associated to f_1 are constant nodes. These constant nodes could be removed as they made useless splits to make the OODG more compact and readable. The OODG after removing all constant nodes is shown in the right one in Figure 1.14. In addition, this OODG of depth 3 does not correctly classify all examples of the small dataset. The OODG with the smallest depth that well classifies all examples is shown in the Figure 1.15, which is in depth 5, with $[f_2, f_4, f_1, f_5, f_6]$ as the sequence of features selected.

1.3.2.3 Recent Exact Methods

As the binary decision diagrams are not widely studied in Machine Learning, there are not many exact methods for this model. We present our MaxSAT-based approach [Hu *et al.* 2022] in Chapter 3. To the best of our knowledge, there are two other recent exact combinatorial optimization approaches for optimal binary decision diagram or its variant. The first one is based on Boolean Satisfiability [Cabodi *et al.* 2021], which is before our contribution. The other one is based on Mixed Integer Linear Programming [Florio *et al.* 2022]. We introduce separately the principal ideas of these methods.

In [Cabodi *et al.* 2021], the authors proposed the first SAT-based approach to find the optimal binary decision diagrams. In detail, the binary decision diagrams found are *ordered* and *reduced*. The goal is similar as the one proposed in [Narodytska *et al.* 2018], shown as follow:

- ***Goal*** *: For a given set of examples $\mathcal{E}$, find the Binary Decision Diagram in the smallest size (number of nodes) classifying all examples in $\mathcal{E}$ correctly.*

The principal idea to achieve this objective is to apply an iterative approach. In each step of the iterative approach, a SAT problem is asked for the existence of a binary decision diagram of a given size N that classifying all examples in $\mathcal{E}$ correctly. In [Cabodi *et al.* 2021], the authors proposed the starting lower bound of the size is 3, it would double the size until the SAT response is found. Then to find the optimal size, a binary search would be performed between the identified range.

The SAT model of a binary decision diagram in a given size contains two major categories of constraints: The constraints to form a *valid binary decision diagram* in given size; and the constraints to let the model *classify all examples correctly.*

Briefly speaking, to form a valid binary decision diagram in given size, the constraints consists of two parts as follow:

- Constraints to describe the *parent-children relationship* between nodes, including for each node (except the root) has at least 1 parent; for the two terminal nodes must not be a parent of the other; for each node (except the terminal nodes) has two identical children (the *reduced* restriction).

- Constraints to relates the *features and nodes*, including for each node is associated to exactly one feature; for preserving the global feature ordering; for relating the global feature ordering with the nodes by the parent-children relationship (the *ordered* restriction).

To let the valid binary decision diagram found classify all examples correctly, the constraints ensure that each example is predicted correctly along one path in the binary decision diagram. We refer readers to [Cabodi *et al.* 2021] for details.

As the first exact combinatorial method for the binary decision diagram, this SAT-based approach aims to find the optimal model in the smallest size classifying all examples correctly. However, there are some weakness of this approach. At first, there is no limit in the (*maximum*) *depth* in the proposed SAT model. It possibly leads to the binary decision diagram found is small in size by deep in depth, which decreases the interpretability. Then, classifying all examples correctly could easily raise the overfitting, which affect the generalization performance. Finally, from the experiments of the paper, the proposed SAT-based approach suffers a lot in the scalability problem.

In [Florio *et al.* 2022], the authors proposed the first MILP-based approach to learn *optimal decision diagrams* (ODDs). Compared to the binary decision diagrams, the decision diagrams found by this approach are more general in the skeleton. At first, the decision diagrams are not *ordered*. Then, the topology of the decision diagrams found is limited by a *preset skeleton*, which is defined by a sequence of maximum number of nodes in each layer. For example, $(1-2-4-8)$, $(1-2-4-4-4)$, $(1-2-3-3-3-3-3)$ are some different skeletons proposed in the experiments of the paper. The decision diagrams found are decided by the *activated nodes* and their *mutual connections*. Moreover, the decision diagrams are designed as *multi-variate*. That is, a linear combination of multiple features could be associated to the non-terminal nodes in the decision diagrams. The decision diagrams could also be *single-variate*, too. The final difference is the decision diagrams found could handle *multi-classification* by arranging a dedicated terminal node for each class.

Unlike the SAT-based approach, the MILP-based approach optimizes the combination of *accuracy* and an additional *regularization term* in the *model size*, which is shown as follow:

- ***Goal**: For a given set of examples $\mathcal{E}$ and a preset skeleton, find the Decision Diagram ϕ with the best value in the objective function as follow:*

$$\min\left(err(\phi;\mathcal{E}) + \alpha\|\phi\|\right)$$

, where $\|\phi\|$ is the size of the Decision Diagram found (the number of activated nodes), and α is a regularization parameter to control the penalty of model size.

Briefly speaking, the MILP-based approach firstly introduces the *flow variables* to represent the trajectory of the examples within the diagram. Then, it connects

these flow variables to the design variables that defines the valid topology of decision diagram, and those variables making the splits. We refer readers to [Florio *et al.* 2022] for details in the mathematical formulation.

As the first MILP-based approach for optimal decision diagrams, this approach has a general objective in ML by considering the regularization term. From the experiments, the authors show the fact that compared to the optimal decision tree, the optimal decision diagram has a much more *balanced data fragmentation*.

Chapter 2

Learning Optimal Decision Trees via MaxSAT

Contents

In this chapter, we present our contribution to learn optimal decision trees via MaxSAT approach. This chapter is an extended version of the paper [Hu *et al.* 2020]. It is divided into six sections. Section 2.1 describes the motivation of this work, and the target problem. Section 2.2 introduces the details of the SAT encoding from [Narodytska *et al.* 2018], which is the essential basic of our research. Section 2.3 introduces the details of the proposed MaxSAT model, and some experimental results. Section 2.5 shows how we adapt our MaxSAT formulation to the well-known AdaBoost Algorithm for better performance. Section 2.6 synthesizes the results obtained by different MaxSAT solvers of the *MaxSAT Evaluation 2021 and 2022*, used in our MaxSAT formulations. Section 2.7 provides a brief summary this chapter.

2.1 Motivation and Problem Description

As a very popular machine learning model, decision tree majorly benefits from its interpretability, and the wide range of efficient methods to compute it. Several classic greedy heuristic methods are introduced in Section 1.3.1.1. Those methods typically build the tree from the top to the bottom, by splitting the datasets with the features selected by different heuristics, like the highest information gain. However, those heuristic methods suffer from the difficulty in interpretability, due to the explosion in tree size and depth. In addition, a complex decision tree leads to the overfitting problem. Therefore, a simpler (e.g smaller) decision tree is not only better in interpretability, but is also often more accurate on unknown data.

Recently, several exact methods to learn optimal decision trees are proposed to offer guarantees of mathematical optimality. Section 1.3.1.2 provides the literature review of some combinatorial optimization approaches for optimal decision trees. As mentioned in Section 1.3.1.2, the metrics widely used for optimal decision trees are *tree size*, *tree depth*, and *accuracy*. Unlike other combinatorial optimization approaches (Constraint Programming, Mixed Integer Linear Programming, Dynamic Programming), there are no exact methods based on Boolean Satisfiability (or its variants) to optimize the accuracy for the decision tree before our research. In fact, the previous SAT-based exact methods [Bessiere *et al.* 2009, Narodytska *et al.* 2018] optimize the tree size, and, [Avellaneda 2020, Janota & Morgado 2020] optimize the tree depth. All of them must subject to the constraint that the decision tree is perfectly accurate on the training set, which is often criticized as it may entail overfitting.

Corresponding to the *Goal 1* (tree size) described in Section 1.3.1.2, the decision problem solved by the SAT model of [Narodytska *et al.* 2018] is:

- $P_{dt}(\mathcal{E}, N)$*: Given a set of examples* $\mathcal{E}$*, is there a valid binary decision tree (each internal node has exactly two children) of size* N *that classifies correctly all examples in* $\mathcal{E}$*?*

The SAT approach finds the decision tree with the smallest size by a linear search of this decision problem. The initial tree size is provided by a heuristic decision tree method, like ITI [Utgoff *et al.* 1997]. Then, the tree size decreases until the answer is unsatisfiable, and, the last decision tree is the optimal one. In contrast, the optimisation problem considered in our research corresponding to the *Goal 3* (accuracy), is:

- $P^{*}_{dt}(\mathcal{E}, H)$*: Given a set of examples* $\mathcal{E}$*, find a valid binary decision tree with maximum/exact depth* H *that maximises the number of examples in* $\mathcal{E}$ *that are correctly classified.*

Moreover, the previous SAT-based methods have the limit in scalability. The principal reasons are the constraint of perfect accuracy and the iterative SAT queries

to check the smallest size/depth. Although solving the MaxSAT formula to optimality is of course harder than solving the corresponding SAT formula, the MaxSAT-based approach does not need the iterative processes. In addition, the use of incomplete MaxSAT solver could return the best result within given reasonable time, even the optimality is not reported.

Finally, the MaxSAT approach can be naturally integrated in AdaBoost to improve the prediction performance. The idea is to update the data distributions by changing the weights of corresponding soft clauses. This technique still improves the scalability, as the individual trees of the ensemble can be smaller.

As our MaxSAT model extends the SAT model in [Narodytska *et al.* 2018], we introduce the details of the SAT model in the next section. To simplify the notation, we assume that the set of examples $\mathcal{E}$ is binary containing M examples, and K binary features.

Additionally, for all possible uses of cardinality constraints in this chapter, we model the cardinality constraints by the sequential counters encodings proposed in [Sinz 2005].

2.2 Details of Previous SAT Encoding

In this section, we present the SAT encoding previously proposed in [Narodytska *et al.* 2018] for solving the decision problem $P_{dt}(\mathcal{E}, N)$, that is to find a valid binary decision tree of size N that classifies correctly all the examples of the dataset $\mathcal{E}$. The SAT encoding contains three parts of constraints as follow:

- ***Part 1:*** *Constraints to encode a valid binary tree of size N.*
- ***Part 2:*** *Constraints to map features (respectively, classes) to internal nodes (respectively, leaf nodes).*
- ***Part 3:*** *Constraints to classify correctly all examples in $\mathcal{E}$.*

We show the details of the constraints in different parts separately.

2.2.1 Encoding a Valid Binary Tree of Given Size

As the encoding considers a valid binary tree, where each internal node has two children, therefore the size N must be an odd number.

To represent a binary tree of given size, the encoding uses the numbering of nodes, which is assumed in the *breadth-first order* from left to right. Namely, the root node of the tree is numbered as 1. Moreover, for a node i, the number of its two children ranges from $i+1$ to $\min(2i+1, N)$. In addition, the number of the left child and the one of the right child are consecutive numbers.

To model whether a node i is an internal node or a leaf node, the encoding applies a propositional variable v_i, where v_i is $true$ (respectively, $false$) indicates the node i is a leaf node (respectively, internal node). For the child-parent relationship between

Var	Description of variables
v_i	**1** iff node i is a leaf node, and **0** otherwise. $\forall i \in \{1, N\}$
l_{ij}	**1** iff node j is the left child of node i, and **0** otherwise. $\forall i \in \{1, N\}$, and $\forall j \in LR(i)$, where $LR(i) = even([i+1, \min(2i, N-1)])$
r_{ij}	**1** iff node j is the right child of node i, and **0** otherwise. $\forall i \in \{1, N\}$, and $\forall j \in RR(i)$, where $RR(i) = odd([i+2, \min(2i+1, N)])$
p_{ji}	**1** iff node i is the parent of node j, and **0** otherwise. $\forall i \in \{1, N-1\}$, and $\forall j \in \{2, N\}$

Table 2.1: Description of propositional variables concerning to tree topology in [Narodytska *et al.* 2018].

the node i and the node j, three sets of propositional variables, l_{ij}, r_{ij}, and p_{ji}, are proposed. The definitions of these variables sets are shown in Table 2.1. Note that l_{ij} and r_{ij} are defined for even/odd indices as a left/right child must be an even/odd node. These shortcuts are defined as $j \in LR(i)$ and $j \in RR(i)$ in Table 2.1.

Example 1 *To encode a valid binary tree with* 5 *nodes* ($N = 5$)*, fours sets of variables are introduced, which are* $\{v_1, \ldots, v_5\}$*,* $\{l_{12}, l_{24}, l_{34}\}$*,* $\{r_{13}, r_{25}, r_{35}\}$*, and* $\{p_{21}, p_{31}, p_{32}, p_{42}, p_{43}, p_{52}, p_{53}, p_{54}\}$*.*

With the help of the proposed sets of variables, encoding a valid binary tree of given size needs several constraints describing the topology. At first, the root must not be a leaf node as we encode a valid tree.

$$(\neg v_1) \tag{2.1}$$

Then, a leaf node has no children, which is for $i = 1, \ldots, N-2$:

$$v_i \rightarrow \neg l_{ij}, \quad j \in LR(i) \tag{2.2}$$

Next, the left child and the right child of the node i are numbered consecutively, which is for $i = 1, \ldots, N-2$:

$$l_{ij} \leftrightarrow r_{ij+1}, \quad j \in LR(i) \tag{2.3}$$

Moreover, an internal node must have exact one left and one right child, which is for $i = 1, \ldots, N-2$:

$$\neg v_i \rightarrow \Big(\sum_{j \in LR(i)} l_{ij} = 1 \Big) \tag{2.4}$$

Additionally, when node i is a parent then it must have two child, which is for $i = 1, \ldots, N-2$:

$$\begin{aligned} p_{ji} &\leftrightarrow l_{ij}, \quad j \in LR(i) \\ p_{ji} &\leftrightarrow r_{ij}, \quad j \in RR(i) \end{aligned} \tag{2.5}$$

Finally, to ensure that the topology must be a tree, except the root, all nodes must have exact one parent, which is for $j = 2, \ldots, N$:

$$\sum_{i=\lfloor \frac{j}{2} \rfloor}^{\min(j-1,N)} p_{ji} = 1 \tag{2.6}$$

Example 2 *We continue with the Example 1. Based on Constraints 2.1, 2.2, 2.3, and 2.4), the following constraints are generated to encode the valid binary tree structure:*

$$\neg v_1; \quad v_1 \rightarrow \neg l_{12}; \quad v_2 \rightarrow \neg l_{24}; \quad v_3 \rightarrow \neg l_{34}$$
$$l_{12} \leftrightarrow r_{13}; \quad l_{24} \leftrightarrow r_{25}; \quad l_{34} \leftrightarrow r_{35}$$
$$\neg v_1 \rightarrow (l_{12} = 1); \quad \neg v_2 \rightarrow (l_{24} = 1); \quad \neg v_3 \rightarrow (l_{34} = 1)$$

Then, the parent-child relations are encoded by Constraints 2.5 and 2.6 as follow:

$$p_{21} \leftrightarrow l_{12}; \quad p_{42} \leftrightarrow l_{24}; \quad p_{43} \rightarrow l_{34}$$
$$p_{31} \leftrightarrow r_{13}; \quad p_{52} \leftrightarrow r_{25}; \quad p_{53} \rightarrow r_{35}$$
$$p_{21} = 1; \quad p_{31} + p_{32} = 1; \quad p_{42} + p_{43} = 1; \quad p_{52} + p_{53} + p_{54} = 1$$

Solving this simple example, the encoding allows only two valid binary trees. The first one has the node 1 *and* 2 *as internal nodes (the left one of Figure 2.1). The second one has the node* 1 *and* 3 *as internal nodes (the right one of Figure 2.1).*

Figure 2.1: The two valid binary trees of size 5.

2.2.2 Mapping Features and Classes to Nodes

Given a valid binary tree topology of a given size, it is essential to map features to internal nodes and to map classes to leaf nodes. Three additional variables are needed to capture these constraints. At first, a propositional variable a_{rj} is introduced to relates each binary feature f_r to each node j. Then, another propositional variable c_j is used to indicate if the class associated to leaf node j is positive or negative. Moreover, to avoid the duplication of feature f_r in any paths from the root until the node j, u_{rj} is proposed to store this selection information. The definitions of these variables sets are shown in Table 2.2.

With the use of these propositional variables, the constraints to map features (classes) to internal nodes (leaf nodes) are shown as follow. For an internal node,

Var	Description of variables
a_{rj}	**1** iff feature f_r is assigned to node j, **0** otherwise. $\forall r \in \{1, K\}, \forall j \in \{1, N\}$
u_{rj}	**1** iff feature f_r is being selected before or in node j, **0** otherwise. $\forall r \in \{1, K\}, \forall j \in \{1, N\}$
c_j	**1** iff class of leaf node j is **1**, **0** otherwise. $\forall j \in \{1, N\}$

Table 2.2: Description of propositional variables for features and classes mapping in [Narodytska *et al.* 2018].

exactly one feature is assigned, which is for $j = 1, \ldots, N$:

$$\neg v_j \rightarrow \big(\sum_{r=1}^{K} a_{rj} = 1\big) \tag{2.7}$$

In reverse, if node j is a leaf node, no feature should be used, which is for $j = 1, \ldots, N$:

$$v_j \rightarrow \big(\sum_{r=1}^{K} a_{rj} = 0\big) \tag{2.8}$$

Then, to judge the feature f_r is being selected before or in node j, which is to avoid the duplication of feature in any paths from the root, we consider the following constraint, with $r = 1, \ldots, K$, $j = 1, \ldots, N$:

$$\begin{gathered} \bigwedge_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} (u_{ri} \wedge p_{ji} \rightarrow \neg a_{rj}) \\ u_{rj} \leftrightarrow \big(a_{rj} \vee \bigvee_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} (u_{ri} \wedge p_{ji})\big) \end{gathered} \tag{2.9}$$

The first part indicates for all possible paths to node j, if the feature f_r is selected before or in its parent, it must not being assigned at the node j. The second part describes that there are two cases for the feature f_r to be selected before or in node j: the first one is that the feature f_r is assigned to node j, the other one is that the feature f_r is selected before or in the parent of node j.

2.2.3 Classifying All Examples Correctly

Considering a valid binary decision tree of given size, solving the decision problem $P_{dt}(\mathcal{E}, N)$ needs constraints for classifying all examples correctly. To ensure the accuracy is perfect, the idea is that all positive examples must not lead to negative leaf nodes. Similarly, all negative examples must not lead to positive leaf nodes. In other words, a positive (resp. negative) leaf node rejects all negative (resp. positive)

examples.

To remember the selection of a feature f_r and its value along the path from the root to the node j, two variables d^0_{rj} and d^1_{rj} are introduced. Concretely, any example having $f_r = 0$ (resp. $f_r = 1$) will be rejected by the node j or by one of its ancestors iff $d^0_{rj} = 1$ (resp. $d^1_{rj} = 1$). The definitions of these two variables sets are shown in Table 2.3.

Var	Description of variables
d^0_{rj}	**1** iff node j, or one of its ancestor, rejects any example having feature $f_r = 0$, $\forall r \in \{1, K\}, \forall j \in \{1, N\}$
d^1_{rj}	**1** iff node j, or one of its ancestor, rejects any example having feature $f_r = 1$, $\forall r \in \{1, K\}, \forall j \in \{1, N\}$

Table 2.3: Description of propositional variables concerning to classification in [Narodytska *et al.* 2018].

Considering the constraints for the feature selection at first, there is no feature selected before the root, therefore, with $r = 1, \ldots, K$:

$$d^0_{r1} = 0, \quad d^1_{r1} = 0 \tag{2.10}$$

Then, to obtain the selection of a feature $f_r = 0$ along the path from the root to node j, with $j = 1, \ldots, N$, $r = 1, \ldots, K$:

$$d^0_{rj} \leftrightarrow \Big(\bigvee_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} ((p_{ji} \wedge d^0_{ri}) \vee (a_{ri} \wedge r_{ij})) \Big) \tag{2.11}$$

This constraint implies two cases. The first one is that one ancestor of the node j already rejected any example having $f_r = 0$, and the second one is that the rejection occurs exactly in the parent of node j. In the decision tree, as the value 0 leads the example to the left child, choosing the right child indicates the rejection.

Analogously, to obtain the selection of a feature $f_r = 1$ along the path from the root to node j, with $j = 1, \ldots, N$, $r = 1, \ldots, K$,

$$d^1_{rj} \leftrightarrow \Big(\bigvee_{i=\lfloor \frac{j}{2} \rfloor}^{j-1} ((p_{ji} \wedge d^1_{ri}) \vee (a_{ri} \wedge l_{ij})) \Big) \tag{2.12}$$

To classify all positive examples correctly, let a positive example e_q (note as $e_q \in \mathcal{E}^+$), and the value of feature f_r for e_q be $\sigma(r, q) \in \{0, 1\}$. For every leaf node j, with $j = 1, \ldots, N$:

$$v_j \wedge \neg c_j \rightarrow \bigvee_{r=1}^{K} d^{\sigma(r,q)}_{rj} \tag{2.13}$$

That is, any positive example must be rejected by the leaf node associated with the negative class.

Similarly, to classify all negative examples correctly, let a negative example e_q (note as $e_q \in \mathcal{E}^-$). For every leaf node j, with $j = 1, \ldots, N$:

$$v_j \wedge c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r,q)} \tag{2.14}$$

That is, any negative example must be rejected by the leaf node associated with the positive class.

Example 3 *Continuing with the Example 2. The Table 2.4 gives a binary dataset from [Narodytska et al. 2018]. Solving the decision problem $P_{dt}(\mathcal{E}, N)$ with $N = 5$ produces the binary decision tree of size 5 given in Figure 2.2. This tree classifies all examples of Table 2.4 correctly. The constraints 2.13 and 2.14 rule out that the second valid binary tree given in Figure 2.1 as it does not ensure perfect classification for this dataset.*

Ex.	L	C	E	S	H
e_1	1	0	1	0	0
e_2	1	0	0	1	0
e_3	0	0	1	0	1
e_4	1	1	0	0	0
e_5	0	0	0	1	1
e_6	1	1	1	1	0
e_7	0	1	1	0	0
e_8	0	0	1	1	1

Table 2.4: A toy dataset from [Narodytska *et al.* 2018].

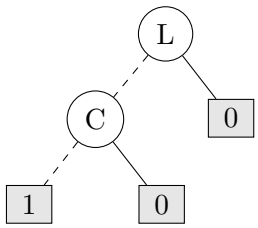


Figure 2.2: The decision tree classifying all examples in Table 2.4.

As mentioned before, we consider a dataset $\mathcal{E}$ containing M examples and K binary features. The encoding size (on the number of literals) of a target decision tree with N nodes is in $\boldsymbol{O(K \times N^2 + M \times N \times K)}$. The term $M \times N$ results from the constraints 2.13 and 2.14, each contains $O(K)$ literals. The term $K \times N^2$ depends. on the remaining constraints. In comparison, the previous model presented in [Bessiere *et al.* 2009], is in $\boldsymbol{O(K \times N^2 \times M^2 + N \times K^2 + K \times N^3)}$, this proposed model is far lighter.

2.3 MaxSAT Model Proposed

In this section, we consider the optimisation problem $P^*_{dt}(\mathcal{E}, H)$, the goal is to find a valid binary decision tree with maximum/exact depth H that maximises the number of examples in $\mathcal{E}$ that are correctly classified. We present MaxSAT model, based on previous SAT model, for this optimization problem. In the experimental evaluation, we first outline the overfitting phenomenon of optimal decision tree with perfect accuracy. Then, we show the comparison between the proposed MaxSAT approach with some state-of-the-art heuristics and exact methods.

The previous SAT encoding for the decision problem $P_{dt}(\mathcal{E}, N)$, considers decision trees of given size N. However, the optimisation problem $P^*_{dt}(\mathcal{E}, H)$ considers the maximum/exact depth H of decision trees. Then, three adaptions are proposed to solve the optimisation problem $P^*_{dt}(\mathcal{E}, H)$ by considering a similar optimisation problem $P^*_{dt}(\mathcal{E}, N)$ defining as follow:

- $P^*_{dt}(\mathcal{E}, N)$*: Given a set of examples* $\mathcal{E}$*, find a valid binary decision tree of size* N *that maximises the number of examples in* $\mathcal{E}$ *that are correctly classified.*

The three adaptions to solve the problem $P^*_{dt}(\mathcal{E}, H)$ are the following:

- ***Adaption 1****: Solve the optimisation problem* $P^*_{dt}(\mathcal{E}, N)$ *via an adapted MaxSAT encoding.*
- ***Adaption 2****: Add new constraints to control the maximum/exact depth* H *of the tree of given size* N*.*
- ***Adaption 3****: Add new constraints to encode the relaxation of the tree size with* N *as an upper bound.*

We present the three adaptions separately in the next paragraphs.

2.3.1 Maximising Examples Correctly Classified

By transforming the previous SAT encoding to MaxSAT encoding, the target of the first adaption is to change the original decision problem $P_{dt}(\mathcal{E}, N)$ to be solved into the optimized version $P^*_{dt}(\mathcal{E}, N)$.

As we introduced before, the previous SAT encoding contains three parts of constraints, where the constraints of classifying correctly all examples matter. Therefore, to realise the first adaption, except the constraints of classifying examples, all constraints (Constraints 2.1- 2.12) are kept as ***hard clauses***. Then, to classify each example, we introduce one Boolean variable b_q for every example $e_q \in \mathcal{E}$ to indicate whether the example e_q is correctly classified or not. The definition of b_q is shown Table 2.6.

Next, we link the b_q variable with the constraints of classifying examples (Constraints 2.13, 2.14) as ***hard clauses***. That is, for every positive example $e_q \in \mathcal{E}^+$, and every leaf node j, with $j = 1, \ldots, N$:

$$b_q \rightarrow (v_j \wedge \neg c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r,q)}) \tag{2.15}$$

And, for every negative example $e_q \in \mathcal{E}^-$, and every leaf node j, with $j = 1, \ldots, N$:

$$b_q \rightarrow (v_j \wedge c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r,q)}) \tag{2.16}$$

Finally, in order to model the objective of maximizing the number of examples that are correctly classified, each literal b_q is declared as a ***soft clause***. Clearly, based on the definition of b_q, the number of satisfied soft clauses is equal to the number of correctly classified examples.

Example 4 *The Figure 2.3 shows the decision tree of size* 9 *found when solving the* $P^*_{dt}(\mathcal{E}, N)$ *optimisation problem on the dataset of Table 2.5. Meanwhile, the smallest decision tree with perfect accuracy via the SAT encoding approach is also in size of* 9.

f_1	f_2	f_3	f_4	f_5	f_6	**Play?**
1	0	0	0	0	1	1
1	0	0	1	1	1	0
1	0	0	1	1	0	0
1	0	0	0	1	0	0
1	0	0	0	0	0	1
0	1	0	0	1	1	1
0	1	0	1	1	0	1
0	1	0	0	0	1	1
0	1	0	1	0	0	1
0	0	1	0	1	1	0
0	0	1	0	0	1	0
0	0	1	0	1	0	1

Table 2.5: A binary dataset (previously introduced in Table 1.2)

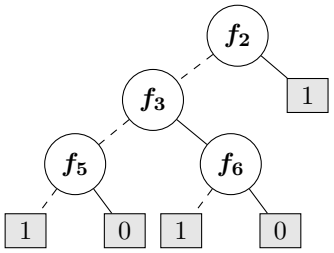


Figure 2.3: An optimal decision tree of size 9 for the dataset in Table 2.5 .

Solving the optimisation problem $P^*_{dt}(\mathcal{E}, N)$ *with other smaller values of* N *produces other optimal decision trees. Figure 2.4 shows two new optimal decision trees in size* 5 *(the left one) and* 7 *(the right one) maximising the accuracy for the same dataset containing* 12 *examples. The decision tree in size* 5 *correctly classifies* 10 *examples, and the other correctly classifies* 11 *examples.*

Figure 2.4: Two optimal decision trees in size 5 and 7 maximising examples for the dataset in Table 2.5.

Example 4 shows that the decision trees in small sizes do not lose a lot in the accuracy. However, compared with the decision trees found via previous SAT model, they could provide better interpretability by profiting their simple topologies.

2.3.2 Controlling Depth for Tree of Given Size

The proposed MaxSAT encoding solves the optimisation problem $P^*_{dt}(\mathcal{E}, N)$, the second adaption aims to control the depth H for the tree of given size N. As a matter of fact, there exist different topologies for a binary decision tree of a given size. For instance, Figure 2.5 shows the two extreme situations of the binary tree topology using the same size $N = 7$: a complete (balanced) binary tree (the left one of depth $H = 2$), and a fully unbalanced binary tree (the right one of depth $H = 3$). Note that we count the depth of binary tree from the root as depth 0.

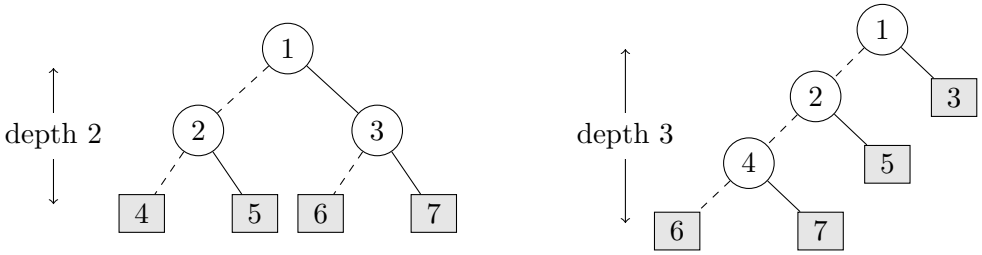


Figure 2.5: Two valid binary trees in size 7 with different depths.

Var	Description of variables
b_q	**1** iff example e_q is correctly classified, **0** otherwise $\forall q \in \{1, M\}$
$depth_{jt}$	**1** iff the node j is in depth t, **0** otherwise $\forall j \in \{1, N\}$, $\forall t \in DS(j)$, where $DS(j) = [\lceil \log(j+1) \rceil - 1, \lceil (j-1)/2 \rceil]$
m_j	**1** iff at least j nodes are used to construct the tree, **0** otherwise $\forall j \in \{3, N\}$

Table 2.6: Description of all new propositional variables for MaxSAT encoding.

In binary tree, the corresponding depth of a given node j varies in an interval reflecting these two extreme situations.

- the upper bound of the depth is associated to the fully unbalanced tree, which is $\lceil (j-1)/2 \rceil$.
- the lower bound of the depth is associated to the complete (balanced) tree, which is $\lceil \log(j+1) \rceil - 1$.

For example, as shown in Figure 2.5, the node 6 and node 7 could be in depth 3 or depth 2. To reflect this property between node and its depth, we introduce a Boolean variable $depth_{jt}$ to indicate the node j is in depth t or not. The definition of $depth_{jt}$ is given in Table 2.6. The depth interval for a node j is defined as $DS(j)$ in this Table.

Example 5 *Considering a valid binary tree with 7 nodes ($N = 7$), from the previous SAT encoding, there are four sets of variables for the tree structure:* $\{v_1, \ldots, v_7\}$,

$\{l_{12}, l_{24}, l_{34}, l_{36}, l_{46}, l_{56}\}$, $\{r_{13}, r_{25}, r_{35}, r_{37}, r_{47}, r_{57}\}$, $\{p_{21}, p_{31}, p_{32}, p_{42}, p_{43}, p_{52}, p_{53}, p_{54}, p_{63}, p_{64}, p_{65}, p_{74}, p_{75}, p_{76}\}$. *The variables controlling the depth are* $\{depth_{10}, depth_{21}, depth_{31}, depth_{42}, depth_{52}, depth_{62}, depth_{63}, depth_{72}, depth_{73}\}$.

In order to control the maximum allowed depth of the tree with a given size, we introduce the following constraints. At first, we notice that the root is always at depth 0:

$$(depth_{10}) \tag{2.17}$$

Then, each node must be at only one depth, which is for $j = 1, \ldots, N$:

$$\sum_{t \in DS(j)} depth_{jt} = 1 \tag{2.18}$$

Next, the children relationship implies the increasing of the depth. In detail, if node i is in depth t, and node j is a child of node i, then node j must be in depth $t + 1$, which is for $i = 1, \ldots, N$:

$$\begin{aligned} depth_{it} \wedge l_{ij} &\rightarrow depth_{j(t+1)}, \quad j \in LR(i) \\ depth_{it} \wedge r_{ij} &\rightarrow depth_{j(t+1)}, \quad j \in RR(i) \end{aligned} \tag{2.19}$$

Finally, to control the tree topology with H as the maximum depth, we set that all possible nodes at depth H must be leaf nodes, which is for $j \in [2H, \min(2^{H+1} - 1, N)]$:

$$depth_{jH} \rightarrow v_j \tag{2.20}$$

The interval of the index of possible nodes in the depth H is also based on the two extreme situations, where $2H$ corresponds to the fully unbalanced situation, and the $\min(2^{H+1} - 1, N)$ corresponds to the complete (balanced) situation.

In addition, the following constraint can be added if H is given as an exact depth instead of an upper bound. The idea is that not only all possible nodes in the depth H must be leaf nodes, but also at least one node is in the depth H:

$$\bigvee_{j=2H}^{\min(2^{H+1}-1,N)} depth_{jH} = 1 \tag{2.21}$$

Example 6 *Continuing with Example 5, the following specific constraints are generated based on Constraints 2.17, 2.18, and 2.19:*

$$depth_{10}; \quad depth_{10} = 1; \quad depth_{21} = 1; \quad depth_{31} = 1; \quad depth_{42} = 1; \quad depth_{52} = 1;$$
$$depth_{62} + depth_{63} = 1; \quad depth_{72} + depth_{73} = 1;$$

$$\begin{aligned} &depth_{10} \wedge l_{12} \rightarrow depth_{21}; \quad depth_{21} \wedge l_{24} \rightarrow depth_{42}; \quad depth_{31} \wedge l_{34} \rightarrow depth_{42}; \\ &depth_{10} \wedge r_{13} \rightarrow depth_{31}; \quad depth_{21} \wedge r_{25} \rightarrow depth_{52}; \quad depth_{31} \wedge r_{35} \rightarrow depth_{52}; \\ &depth_{31} \wedge l_{36} \rightarrow depth_{62}; \quad depth_{42} \wedge l_{46} \rightarrow depth_{63}; \quad depth_{52} \wedge l_{56} \rightarrow depth_{63}; \\ &depth_{31} \wedge r_{37} \rightarrow depth_{72}; \quad depth_{42} \wedge r_{47} \rightarrow depth_{73}; \quad depth_{52} \wedge r_{57} \rightarrow depth_{73}; \end{aligned}$$

Considering the maximum depth is 3*, we have the following constraints:*

$$depth_{63} \rightarrow v_6; \quad depth_{73} \rightarrow v_7$$

Solving this simple example, the encoding allows five different topologies, where one of depth 2 *(the left one in Figure 2.5), and four of depth* 3 *(the right one of Figure 2.5 and three additional trees in Figure 2.6).*

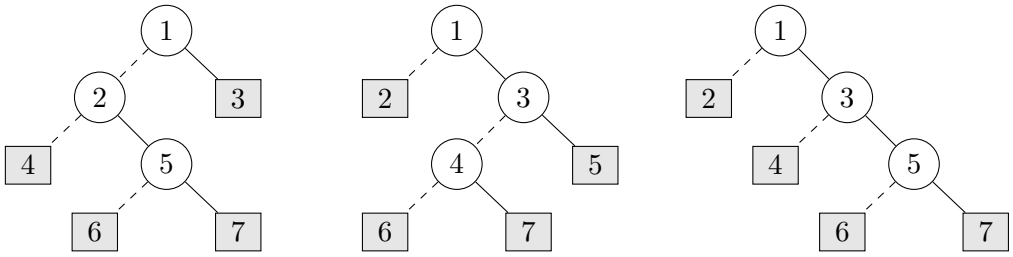


Figure 2.6: The other three valid binary trees in size 7 of depth 3.

If we add the constraints to control the tree topology of exact depth 3*, from Constraint 2.21:*

$$depth_{63} \vee depth_{73} = 1;$$

Then, the topologies allowed by the encoding only correspond to the four trees of depth 3. The tree of depth 2 *is avoided, as the* $depth_{62}$ *and* $depth_{72}$ *must be false.*

To combine the constraints of controlling the depth into the MaxSAT encoding, we simply keep these constraints as ***hard clauses***, as they could be viewed as an extension of structural constraints.

2.3.3 Limiting Tree Size in Given Interval

After the first and second adaption, the MaxSAT encoding could not only solve the optimisation problem $P^*_{dt}(\mathcal{E}, N)$, but also control its maximal or exact depth. However, this encoding could not solve the final problem $P^*_{dt}(\mathcal{E}, H)$ as it models the tree with given size. In this section, we show the final adaption to build a decision tree with an upper bound on the size instead of the exact size.

There is a relationship between the size and the depth in a valid binary tree. That is, when the depth of a valid binary tree is given, the size of the tree is in a corresponding interval. In details, for the maximum depth H, the upper bound of the size is $2^H - 1$. Moreover, if H is set as exact depth, we additionally get the lower bound of the size as $2H + 1$. Recall that the size of a valid binary decision tree can only be an odd number starting from 3. In common, suppose that N is an upper bound of the tree size, we introduce a Boolean variable m_j to indicate that there are at least j ($j \in \{3, 5, \ldots, N\}$) nodes to construct the tree. The definition of m_j is given in Table 2.6. The constraints for controlling the tree size, are then the following:

As at least 3 nodes are necessary to build a valid tree, so we need to enforce variable m_3 to be true. Then, if at least $j+2$ nodes are used to construct the tree, it must use at least j nodes, which is for $j \in [1, N-2]$:

$$m_{j+2} \rightarrow m_j \tag{2.22}$$

Next, we apply the following simple rule to adapt each constraint set as hard clauses: we look at each hard clause $\mathcal{C}$ from the encoding separately, and consider j as the largest node index used in $\mathcal{C}$. We simply replace the original hard clause $\mathcal{C}$ by the two cases:

$$\begin{aligned} m_j &\rightarrow \mathcal{C}, \qquad \text{if } j \text{ is odd;} \\ m_{j+1} &\rightarrow \mathcal{C}, \qquad \text{if } j \text{ is even;} \end{aligned} \tag{2.23}$$

That is, if j is odd (respectively even) and at least j (respectively $j+1$) nodes are used, then the hard clause $\mathcal{C}$ is held. The use of Constraints 2.22 and 2.23 allows the tree size is restricted by an upper bound, which is applicable to control the maximum tree depth. In addition, to set a lower bound N_l to limit tree size in a given interval, we simply enforce m_{N_l} as true indicating at least N_l nodes are used. This adaption could help control the exact tree depth.

Example 7 *Considering we control the maximum depth of the target decision trees between* 2 *and* 3*, then, the corresponding intervals to limit the tree size are* $[3, 7]$ *and* $[3, 15]$*. In addition, for exact depth* $H = 3$*, we simply change the interval as* $[7, 15]$*, which increases the lower bound.*

For the toy dataset of Table 2.5, solving the optimisation problem $P^*_{dt}(\mathcal{E}, H)$ *with* $H \leq 2$ *produces the decision tree given in the left part of Figure 2.7. This decision tree maximises the number of examples correctly classified (*11/12*). Solving the optimisation problem with* $H \leq 3$ *gives the decision tree on the right part of Figure 2.5. This tree correctly classifies all examples (*12/12*) of the dataset.*

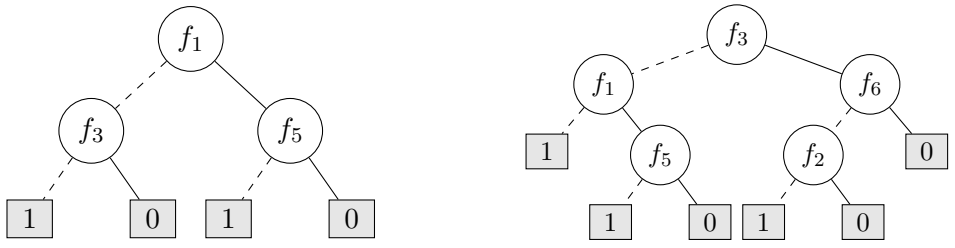


Figure 2.7: The optimal decision trees of maximum depth 2 and 3, that maximise the number of correctly classified examples for the dataset of Table 2.5.

To simplify the notation, we refer to the MaxSAT encoding applying the three adaptions to find the decision tree with *maximum depth* as MaxSAT-DT-max, and with *exact depth* as MaxSAT-DT-exact.

2.4 Experimental Results

In this section, we present our large experimental studies to evaluate our propositions on different levels. The source code (developed in Python) and datasets are available online at `https://gitlab.laas.fr/hhu/maxsat-decision-trees`. The outline of the experiments contains two parts. The first experiment aims to highlight the overfitting behaviour of the decision trees found via the existing SAT approach. In the second experiment, we evaluate the prediction performance between our propositions (MaxSAT-`DT`-max and MaxSAT-`DT`-exact) with the state-of-the-art heuristic and exact methods. Here the heuristic method used is **CART** [Breiman *et al.* 1984], and the exact method is **DL8.5** [Aglin *et al.* 2020]. Both of them are described in Section 1.3.1.

We perform experiments on datasets from CP4IM[1]. The dataset are binarized with the classical one-hot encoding. In Table 2.7, we present the characteristic of these datasets. In detail, the column $\boldsymbol{M}$ indicates the number of examples in the dataset, the column $\boldsymbol{K_{orig}}$ indicates the original number of features, the column $\boldsymbol{K}$ indicates the number of binary features after binarization, and the column ***pos*** indicates the percentage of positive examples in the dataset.

Dataset	M	K_{orig}	K	pos
anneal	812	42	89	0.77
audiology	216	67	146	0.26
australian	653	51	124	0.55
cancer	683	9	89	0.35
car	1728	6	21	0.30
cleveland	296	45	95	0.54
hypothyroid	3247	43	86	0.91
kr-vs-kp	3196	36	73	0.52
lymph	148	27	68	0.55
mushroom	8124	21	112	0.52
tumor	336	15	31	0.24
soybean	630	16	50	0.15
splice-1	3190	60	287	0.52
tic-tac-toe	958	9	27	0.65
vote	435	16	48	0.61

Table 2.7: Detailed information of datasets from CP4IM used in experiments.

We ran all experiments on a cluster using Xeon E5-2695 v3@2.30GHz CPU running xUbuntu 16.04.6LTS.

2.4.1 The Overfitting Phenomenon

The first experiment aims to show the existence of the overfitting phenomenon for the SAT approach of learning optimal decision trees with perfect accuracy

[1] `https://dtai.cs.kuleuven.be/CP4IM/datasets/`

from [Narodytska *et al.* 2018]. However, the tendency of the increase of the training accuracy could not be obtained directly by applying the SAT method, because during the iterative process of decreasing the tree size, all the decision trees found have perfect accuracy. To solve this problem, we learn decision trees with our MaxSAT encoding (just applying the Adaption 1, solving the optimisation problem $P^*_{dt}(\mathcal{E}, N)$), by increasing the tree size starting from 3 until we find the size that classifies correctly all examples in the training set. The final decision tree obtained via the MaxSAT model is in the same size as the optimal decision tree found via the previous SAT approach, as it is the decision tree in the smallest size with perfect accuracy.

Considering the scalability of the SAT method, in this experiment, for each dataset, we use the hold-out method to split the training and testing set. Following the experiment topology in [Narodytska *et al.* 2018], we choose 3 different small ratios $r = \{0.05, 0.1, 0.2\}$ to generate the training set, and the remaining examples are used as the testing set. This process is repeated 10 times with different random seeds to avoid the influence of random seeds. The MaxSAT solver we used is RC2 [Ignatiev *et al.* 2019], which is an effective complete MaxSAT solver. For each training process, the solver is left with no time limit until it finds the optimal solution (in terms of training accuracy).

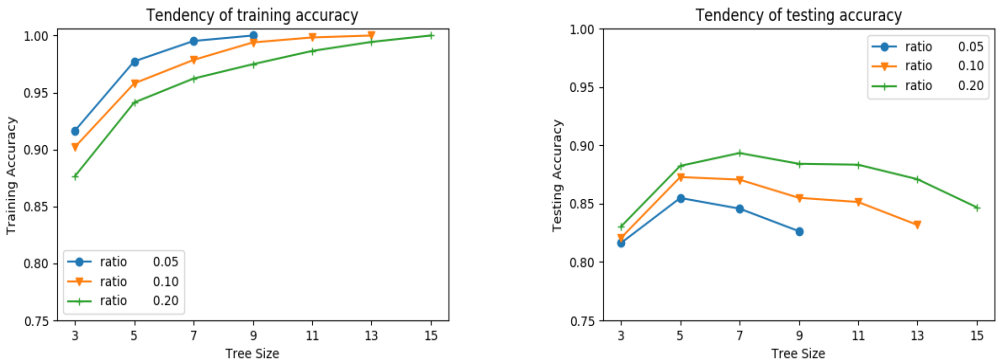


Figure 2.8: The tendency of the training accuracy (left) and testing accuracy (right) with the increase of tree size for the dataset "breast-cancer" in different ratios.

In Figure 2.8, we report the average training accuracy and testing accuracy of the decision trees with the increase of tree size for the dataset "breast-cancer" using different sampling ratios, where the left one indicates the training accuracy, the right one indicates the testing accuracy. We observe clearly the improvement of training accuracy with the growth of tree size, until reaching a perfect classification. However, the testing accuracy shows that the perfect decision tree overfits the training set, since the smaller trees, while less accurate on the training set have better testing accuracy. The results for the other dataset are detailed in Appendix A. This overfitting phenomenon is not remarkable for every dataset, but we almost systematically observe a plateau whereby the testing accuracy stays constant at best while the training accuracy increases.

2.4.2 Comparison with Different Methods

In the second experiment, we compare the prediction of our propositions (MaxSAT-DT-max and MaxSAT-DT-exact) with the CART method as a state-of-the-art heuristic (with the scikit-learn Python library [Pedregosa *et al.* 2011]), and DL8.5 as the state-of-the-art exact method (via its Python package in version 0.0.9). Except for the maximum depth, all parameters are kept to their default values for all models.

As the MaxSAT approach faces less scalability problem, for each dataset, we use stratified sampling to preserve the class distribution with 5-fold cross-validation. This process is repeated 10 times with different random seeds to avoid the influence of random seeds. Unlike the first experiment, the MaxSAT solver we used in this experiment is Loandra [Berg *et al.* 2019], which is the winner of incomplete solver in the *MaxSAT Evaluation 2019*. In fact, a complete MaxSAT solver could not scale well on the datasets we used, as it only return the solution when the optimality is reported. Whereas, the incomplete MaxSAT solver can return the best solution, or report optimality within limited time.

For each experiment, the timeout for training is set to 15 minutes, and the memory limit is set to 16GB. The candidate maximum depths are restricted to $H \in \{2, 3, 4\}$ for CART, DL8.5, and MaxSAT-DT-max. For MaxSAT-DT-exact, the candidate depths are used as *exact depths*.

Table 2.8 reports the average training accuracy of the different methods, and Table 2.9 reports the average testing accuracy. Each row corresponds to 50 runs for a given dataset with a given depth (5-fold cross-validation with 10 different random seeds). The column "**Acc**" stands for accuracy in percent, the column "**Opt**" indicates the percentage of reporting optimality, and the column "**Time**" indicates the run-time in seconds. The value "**MO**" corresponds to a memory out, and the value "**TO**" is a timeout. The best values between different methods are marked in blue. Moreover, for methods MaxSAT-DT-exact and MaxSAT-DT-max, their training and testing accuracy are marked with "$*$" if they are within 3% points of the best.

We do not report exact run time of CART as it takes only few seconds. At first, from the results in Table 2.8 and 2.9, we observe that the methods MaxSAT-DT-exact and MaxSAT-DT-max are competitive with both heuristic and exact methods in prediction performance. Although both MaxSAT-DT approaches could not always report optimality within limited time, it is close to the optimal solution obtained by DL8.5. In addition, we observe that DL8.5 needs massive memory for deep trees, while the two MaxSAT-DT methods do not. Indeed, as a dynamic programming based approach, DL8.5 benefits its effectiveness in run time, but also suffers from the trade-off between time and memory. For instance, for a maximum depth of 5, DL8.5 runs out of memory on 6 datasets, even when lifting the memory limit into 50GB. This scalability problem explains why we consider small depths.

Datasets	H	MaxSAT-DT-exact			MaxSAT-DT-max			CART	DL8.5	
		Acc	Opt	Time	Acc	Opt	Time	Acc	Acc	Time
anneal	2	83.19	86	593.71	83.19	94	530.45	81.48	83.19	0.03
	3	85.06*	0	TO	85.05*	0	TO	81.60	86.26	1.76
	4	86.11*	0	TO	85.79	0	TO	82.68	89.10	73.53
audilogy	2	95.44	100	25.84	95.44	100	25.17	94.91	95.44	0.04
	3	97.87*	0	TO	97.84*	0	TO	97.33	98.07	2.42
	4	94.43	36	706.00	99.43*	34	777.25	98.89	99.90	46.33
australian	2	87.00*	12	866.79	87.00*	12	883.78	86.68	87.01	0.06
	3	87.83*	0	TO	87.79*	0	TO	86.88	89.00	6.96
	4	88.44*	0	TO	88.19*	0	TO	89.04	MO	MO
cancer	2	94.94	100	2.90	94.94	100	3.19	94.42	94.94	0.02
	3	96.66*	20	842.64	96.66*	7	844.62	95.66	96.67	0.71
	4	97.70*	0	TO	97.39*	0	TO	96.91	98.10	20.38
car	2	85.53	100	3.48	85.53	100	3.95	85.53	85.53	0.01
	3	89.24	96	591.65	89.24	74	700.19	88.48	89.24	0.03
	4	91.60*	0	TO	91.47*	0	TO	89.62	92.31	0.31
cleveland	2	80.94*	0	TO	80.95	0	TO	78.20	80.95	0.03
	3	83.95	0	TO	85.00*	0	TO	85.78	87.20	2.99
	4	85.57	0	TO	85.31	0	TO	88.32	92.77	134.72
hypothyrold	2	97.84	100	130.88	97.84	100	131.56	97.84	97.84	0.04
	3	98.13*	0	TO	98.13*	0	TO	98.12	98.14	2.95
	4	98.36*	0	TO	98.35*	0	TO	98.38	98.44	121.92
kr-vs-kp	2	86.92	100	130.88	86.92	100	131.56	77.49	86.92	0.03
	3	93.60*	0	TO	93.61*	0	TO	90.43	93.81	1.60
	4	93.80*	0	TO	94.20*	0	TO	94.09	95.50	67.18
lymph	2	86.07	100	31.32	86.07	100	35.98	84.58	86.07	0.01
	3	91.71*	0	TO	91.78*	0	TO	89.97	92.81	0.44
	4	94.51*	0	TO	94.87*	2	868.11	94.88	99.00	7.75
mushroom	2	96.90	100	89.89	96.90	100	132.94	92.71	96.90	0.09
	3	99.73*	0	TO	99.69*	0	TO	96.55	99.90	4.94
	4	100	100	354.33	99.99*	96	388.80	99.92	100	28.65
tumor	2	83.07	100	17.16	83.07	100	17.74	82.82	83.07	0.01
	3	86.32*	0	TO	86.44*	0	TO	84.81	86.58	0.10
	4	87.45*	0	TO	87.71*	0	TO	87.45	90.22	1.57
soybean	2	91.27	100	8.54	91.27	100	8.56	89.29	91.27	0.01
	3	95.46*	0	TO	95.50*	0	TO	92.23	95.51	0.20
	4	97.14*	0	TO	97.05*	0	TO	94.39	98.02	3.58
splice-1	2	84.13*	0	TO	84.17*	0	TO	84.04	84.31	0.50
	3	84.06	0	TO	85.90	0	TO	91.31	92.98	81.01
	4	86.21	0	TO	83.91	0	TO	95.43	MO	MO
tic-tac-toe	2	71.12	100	99.50	71.12	100	92.00	70.92	71.12	0.01
	3	77.45*	0	TO	76.94*	0	TO	75.70	78.65	0.08
	4	82.05	0	TO	81.33	0	TO	83.82	86.65	1.14
vote	2	96.22	100	3.49	96.22	100	3.61	95.63	96.22	0.01
	3	97.39*	0	TO	97.35*	0	TO	96.91	97.50	0.23
	4	98.50*	0	TO	98.51*	0	TO	98.03	99.18	4.04

Table 2.8: Evaluation of the training accuracy between the MaxSAT-DT-max, MaxSAT-DT-exact, CART, and DL8.5.

Datasets	H	MaxSAT-DT-exact	MaxSAT-DT-max	CART	DL8.5
anneal	2	82.16*	82.05*	81.09	82.31
	3	84.23*	84.29*	81.22	85.26
	4	83.71*	84.50*	80.94	86.42
audilogy	2	94.82*	94.49*	94.56	94.92
	3	93.72*	93.91*	94.16	93.53
	4	94.70	94.08*	94.51	94.50
australian-credit	2	84.72*	84.89*	86.59	84.81
	3	84.92*	85.31*	84.99	85.33
	4	85.00*	85.31*	85.44	MO
breast-cancer	2	93.97	93.92*	93.89	93.88
	3	94.07*	94.45	93.80	94.14
	4	94.05	93.95*	93.79	93.66
car	2	85.53	85.53	85.53	85.53
	3	87.49*	87.49*	87.65	87.49
	4	89.69*	89.84*	87.93	90.69
heart-cleveland	2	71.59*	71.60*	72.71	71.53
	3	76.08	76.09	79.37	75.85
	4	75.11	75.82*	76.55	78.15
hypothyrold	2	97.84	97.84	97.84	97.84
	3	97.83*	97.83*	97.87	97.86
	4	97.98*	98.04*	98.10	97.87
kr-vs-kp	2	86.92	86.92	76.75	86.92
	3	93.57*	93.56*	90.43	93.75
	4	93.69*	94.10*	94.09	95.36
lymph	2	79.16*	79.42*	80.85	79.07
	3	80.95*	80.45*	79.05	81.80
	4	80.53*	81.71*	82.15	80.28
mushroom	2	96.90	96.90	92.71	96.90
	3	99.97	99.66*	96.53	99.90
	4	100	99.98*	99.90	100
primary-tumor	2	79.55*	79.82*	80.15	80.06
	3	82.79*	82.61*	79.83	83.27
	4	82.74*	83.24*	81.72	83.30
soybean	2	91.27	91.27	87.94	91.27
	3	94.29*	94.19*	90.46	94.35
	4	95.30*	96.04*	92.46	96.17
splice-1	2	83.44*	83.22*	84.04	82.80
	3	83.62	85.60	90.96	92.83
	4	85.87	83.18	95.25	MO
tic-tac-toe	2	67.45*	67.51*	68.22	67.59
	3	73.54	73.42*	72.04	72.22
	4	78.15*	78.04*	80.97	80.30
vote	2	94.94*	94.98*	95.44	94.84
	3	94.16*	94.18*	94.67	93.95
	4	94.64	94.25*	94.57	93.61

Table 2.9: Evaluation of the testing accuracy between the MaxSAT-DT-max, MaxSAT-DT-exact, CART, and DL8.5.

2.5 Boosting the Model

In this section, we explain how the MaxSAT-DT approaches are well adapted to implement the classical Boosting method AdaBoost presented in Section 1.2.3.2.

The motivation of this adaption is to improve the generalization performance of MaxSAT-DT approaches. Then, we conduct an experimental evaluation to show the impact of the proposed integration of the MaxSAT-DT approaches in AdaBoost.

2.5.1 Integration in AdaBoost

Before explaining details of the integration of MaxSAT-DT approaches in AdaBoost, we briefly show the natural resemblance between MaxSAT-DT approaches and Boosting methods. As described in Section 1.2.3.2, the core of Boosting methods is to adjust the data distribution by the predictions made in each iteration. Meanwhile, the MaxSAT formulas generated by MaxSAT-DT could approximate the data distribution by the weights of their *soft clauses*. In Section 2.3.1, we present that each *soft clause* indicates whether the corresponding example is correctly classified or not. Originally, we consider that all soft clauses share the same weight, which is equivalently viewed as an average data distribution. In this section, we use the *weighted partial MaxSAT* to allow different weights for the soft clauses, so that the data distribution is approximated.

In details, the MaxSAT formula used to learn the decision tree at the iteration t is *identical* to the one at the previous iteration, except for the *weight* associated to each soft clause b_q. Therefore, to approximate the data distribution $\mathcal{D}_t$ of the iteration t, we associate every soft clause b_q to a positive integer weight w_q^t. We set all weights at the first iteration with the value 1 as initial distribution, indicating the equal importance of each example. Then, the weight for the next iteration w_q^{t+1}, is calculated based on w_q^t in two steps.

Firstly, we update and normalize the weights:

$$\widehat{w_q^{t+1}} = \frac{w_q^t * factor_q^t}{\sum_{q=1}^{M}(w_q^t * factor_q^t)} \tag{2.24}$$

where $factor_q^t$ is an updating factor based on the predictions made by the decision tree ϕ_t learnt in the iteration t for the example $e_q = (\boldsymbol{x_q}, y_q)$:

$$factor_q^t = \begin{cases} \exp(-\alpha_t) & \text{if } \phi_t(\boldsymbol{x_q}) = y_q \\ \exp(\alpha_t) & \text{if } \phi_t(\boldsymbol{x_q}) \neq y_q \end{cases} \tag{2.25}$$

The value $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$ is the weight of ϕ_t in the final voting of AdaBoost, and the value ε_t is the error rate of ϕ_t in the example set (see Algorithm 4 for details).

The second step is to discretize the weight $\widehat{w_q^{t+1}}$ as follow, as the weighted partial MaxSAT could only accept positive integer weights:

$$w_q^{t+1} = round(\frac{\widehat{w_q^{t+1}}}{\min_{q\in[1,\ldots,M]}(\widehat{w_q^{t+1}})}) \tag{2.26}$$

We recall the readers that the final prediction made by AdaBoost is $\Phi(\boldsymbol{x}) =$

$sign(\sum_{t=1}^{\mathcal{T}} \alpha_t \phi_t(\boldsymbol{x}))$, where $\mathcal{T}$ is the number of iterations preset.

2.5.2 Experimental Results

In order to show the influence of the integration of MaxSAT-DT approach in AdaBoost, we make this experiment to compare it with the original MaxSAT-DT approach. In addition, to check the differences between different ensemble methods, we also compare our integration of MaxSAT-DT in Adaboost with the integration of MaxSAT-DT approach in Bagging, and with the AdaBoost based on CART, . To simplify the notation, we refer the integration of MaxSAT-DT approach in AdaBoost as **MaxSAT-DT-adaboost** (or **DT-ada** in short), the integration of MaxSAT-DT in Bagging as **MaxSAT-DT-bagging** (or **DT-bag** in short), and the AdaBoost based in CART as **CART-adaboost** (or **CART-ada** in short).

We fix the the parameters of the decision trees for different methods. That is, for each ensemble method, the decision tree learnt in each iteration shares the same parameters. In particular, we use the decision tree found via MaxSAT-DT-max approach as the base learner in the ensemble methods. The candidate maximum depths are $H \in \{2, 3, 4\}$. The MaxSAT solver used is Loandra, and the timeout is set to 15 minutes. For each ensemble method, we set 21 as the number of learners as it is quite reasonable size, and an odd number could avoid ties in the voting phase.

We selected datasets where MaxSAT-DT approach do not perform well in the previous experiment. Moreover, considering the computational time, we use the hold-out method with the ratio $r = 0.8$ to split the training set, and the rest are used as testing set. This process is also repeated 10 times with different random seeds to avoid the influence of random seeds.

The results are presented in Table 2.10, where the accuracy are shown in percentage, and the best values are marked in blue. Moreover, the accuracy are marked with "$*$" if they are within 3% points of the best. The results clearly show that, compared with MaxSAT-DT-max approach, the MaxSAT-DT-adaboost method does improve the prediction performance (both in training and testing accuracy) for almost all instances. For example, for the dataset "*car*" and "*tic-tac-toe*", the improvement in prediction accuracy is more than 10%. The method MaxSAT-DT-bagging obtains better prediction quality compared to MaxSAT-DT-max approach, reflecting the effectiveness of ensemble methods. However, it presents worse prediction compare to MaxSAT-DT-adaboost. In addition, compared with the boosted trees based on CART, MaxSAT-DT-adaboost is competitive in prediction quality (both in training and testing accuracy).

2.6 Performance of Different MaxSAT Solvers

Fortunately, we are honored that the MaxSAT formulas of MaxSAT-DT-max and MaxSAT-DT-adaboost are selected as benchmarks for the *MaxSAT Evaluation (MSE)*

Datasets	H	Testing accuracy				Training accuracy			
		DT-max	DT-ada	CART-ada	DT-bag	DT-max	DT-ada	CART-ada	DT-bag
anneal	2	80.98	80.98	83.99	80.98	83.20	83.36*	85.21	83.17
	3	82.82	84.90*	86.81	82.82	84.90	88.60	88.26	85.00
	4	82.82	84.82*	87.55	83.30	85.52	90.14*	91.05	85.81
australian	2	87.12	87.20	85.98	89.20	86.56	89.98	93.84	85.89
	3	87.88	87.54*	87.05	88.72	87.33	89.66	93.80	87.03
	4	87.88	87.88*	85.91	89.14	88.48	90.21	99.31	87.03
car	2	84.68	96.53	95.29	84.68	85.75	97.47	96.27	85.75
	3	87.86	95.44*	97.83	89.92	89.15	97.10*	98.84	91.08
	4	90.46	98.36*	98.67	95.02	91.24	98.70*	99.91	95.30
cleveland	2	73.33	78.33*	79.83	76.85	80.93	90.25	90.13	85.22
	3	78.33	83.70	81.17	81.48	83.90	93.97	99.15	88.37
	4	85.0	80.17	77.83	83.89	83.47	96.09	100	90.07
tumor	2	82.35	82.35	81.91	82.03	82.84	86.19*	87.05	83.25
	3	82.65	84.48	80.74	84.31	86.34	90.92	89.96	87.77
	4	80.88	79.14	78.97	84.80	86.57	88.81	93.77	89.59
tic-tac-toe	2	68.91	77.14*	78.08	71.27	70.98	80.39*	82.61	72.37
	3	74.61	94.69	93.01	81.46	76.86	95.96*	96.31	89.46
	4	76.74	94.49*	96.94	85.26	81.31	95.80	100	85.37

Table 2.10: Evaluation of the different Ensemble Methods.

in 2021 [Bacchus *et al.* 2021a] and 2022 [Bacchus *et al.* 2022]. In this section, we present a summary of the performance of different MaxSAT solvers on our benchmarks executed during *MSE 2021* and *MSE 2022*. In MSE, the competition between MaxSAT solvers is organized in three main tracks: complete tracks, incomplete tracks and incremental tracks (new in 2022). Depending on the type of MaxSAT formula, there are two sub-tracks in both complete and incomplete tracks, corresponding to unweighted and weighted formula.

The descriptions of the proposed benchmarks are detailed in [Hu *et al.* 2021] and is given in Appendix B. Our benchmarks (unweighted and weighted formula) are used in both complete and incomplete tracks. Our unweighted instances are produced by the MaxSAT-DT-max approach and our weighted instances are produced by the MaxSAT-DT-adaboost approach.

In the complete tracks of *MSE*, a time out is imposed (300 seconds) for all MaxSAT solvers. The vast majority of MaxSAT solvers could not report solution in this time limit. Meanwhile, the solver logs do not show sufficient valuable information to make provable analysis. Additionally, in *MSE 2022*, our weighted instances are used in the new incremental track. However, similarly to the complete track, the results of the incremental track do not produce sufficient valuable information. Therefore, we focus majorly on the results of the incomplete track to compare different MaxSAT solvers on our unweighted and weighted instances.

2.6.1 Incomplete Unweighted Track

The incomplete unweighted track uses the MaxSAT formulas generated by MaxSAT-DT-max. Overall, the benchmarks we provided contain the formulas for 15 datasets from CP4IM. Considering the number of different benchmarks, the *MaxSAT Evaluation* chooses part of the benchmarks to evaluate the MaxSAT solvers. The benchmarks are randomly chosen for each track. Therefore, we present the summary of evaluations separately by years.

In *MSE 2021*, six MaxSAT solvers are evaluated in the incomplete unweighted track. These solvers are **Exact** [Devriendt 2021], **Loandra** [Berg *et al.* 2019], **SATLike-c** and its variant **SATLike-ck** with control of steps [Lei *et al.* 2021, Cai & Lei 2020], **StableResolve** [Reisch *et al.* 2020], and **TT-Open-WBO-Inc-21**(*"TT-WBO"* in short) [Martins *et al.* 2014, Martins *et al.* 2021]. Table 2.11 shows the evaluation of these MaxSAT solvers with 300 seconds as the time limit. In this table, the benchmarks are identified by the combination of dataset and the maximum depth, which are shown in the column "Dataset/H". The column "Best" indicates the smallest number of unsatisfied soft clauses between the different MaxSAT solvers, where 0 indicates that all soft clauses are satisfied (for example, there is no unsatisfiable clauses for benchmarks "*vote/5*" and "*lymph/6*"). Each cell gives the ratio in percent between the best result and the result of each MaxSAT solver. The value in brackets is the number of unsatisfiable clauses obtained by the considered solver. An higher ratio indicates a better performance of corresponding MaxSAT solver. Two special cases need attention. The first case is when the MaxSAT solver could not obtain a solution in the time limit, the ratio is then set to 0 (for example, it is the case for the solver **Exact** on benchmark *splice-1/5*). The second case is when the best value is 0, the ratio is calculated with (*best_value* +1) / (*corresponding_value* +1). The best MaxSAT solver for each benchmark is marked in blue. In addition, we add a summary row to count, for each MaxSAT solver, the number of benchmarks with the best value.

From Table 2.11, we observe that the MaxSAT solver with the best performance on our benchmarks is **SATLike-c** and its variant **SATLike-ck**, which is a hybrid solver combining the local search algorithm "*SATLike*" and the "*Open-WBO*" MaxSAT solver. Then, **Loandra** and **TT-Open-WBO-Inc-21** rank the second best efficient MaxSAT solvers. Moreover, we observe that, in general, Loandra performs well for benchmarks with small maximum depths (like $3, 4$), but badly for benchmarks with large maximum depths (like $5, 6$).

In *MSE 2022*, three new MaxSAT solvers are evaluated in the incomplete unweighted track. The new MaxSAT solvers are **DT-Hywalk** [Zheng *et al.* 2022b, Zheng *et al.* 2022a], **noSAT-MaxSAT** (*noSAT* in short) [Lübke & Schupp 2022], and **NuWLS-c** [Chu *et al.* 2022]. The comparison of these different MaxSAT solvers with 300 seconds as time limit are shown in Table 2.12. The columns are the same as in Table 2.11. The column "**TT-OpenWBO-Inc-***" (*TT-WBO-** in short) indicates the different variants of the MaxSAT solver **TT-Open-WBO-Inc** [Nadel 2020] (**TT-Open-WBO-Inc-i** indicates the application of *IntelSAT*

Dataset/H	Best	Exact	Loandra	SATlike-c	Satlike-ck	StableResolve	TT-WBO
australian/3	67	0.38 (179)	0.99 (68)	1.00 (67)	0.99 (68)	0.92 (73)	1.00 (67)
tumor/3	40	0.66 (61)	0.98 (41)	1.00 (40)	0.85 (47)	0.67 (60)	0.89 (45)
soybean/3	22	0.47 (48)	1.00 (22)	0.85 (26)	1.00 (22)	0.31 (73)	1.00 (22)
splice-1/3	1176	0.96 (1225)	0.93 (1265)	1.00 (1176)	0.92 (1282)	0.85 (1389)	0.85 (1387)
vote/3	6	0.64 (10)	1.00 (6)	0.88 (7)	0.88 (7)	0.58 (11)	0.88 (7)
anneal/4	90	0.59 (153)	1.00 (90)	0.95 (95)	0.95 (95)	0.75 (120)	0.95 (95)
hypothyroid/4	46	0.10 (475)	0.03 (1612)	1.00 (46)	0.22 (211)	0.46 (102)	0.30 (157)
tumor/4	36	0.64 (57)	1.00 (36)	0.95 (38)	0.90 (40)	0.31 (118)	0.95 (38)
soybean/4	12	0.18 (73)	1.00 (12)	1.00 (12)	0.54 (23)	0.18 (73)	1.00 (12)
tic-tac-toe/4	161	0.55 (296)	0.98 (165)	0.79 (204)	1.00 (161)	0.70 (230)	0.79 (204)
vote/4	2	0.18 (16)	0.60 (4)	1.00 (2)	1.00 (2)	0.25 (11)	0.75 (3)
cancer/5	12	0.15 (87)	0.43 (29)	1.00 (12)	0.65 (19)	0.07 (179)	0.81 (15)
cleveland/5	32	0.25 (132)	0.89 (36)	1.00 (32)	0.83 (39)	0.41 (79)	1.00 (32)
splice-1/5	1122	0.00 (-)	1.00 (1122)	0.90 (1250)	0.88 (1280)	0.49 (2300)	0.87 (1297)
tic-tac-toe/5	134	0.52 (261)	1.00 (134)	0.81 (165)	0.77 (175)	0.58 (230)	0.82 (163)
vote/5	0	0.10 (9)	0.25 (3)	0.33 (2)	0.33 (2)	0.01 (95)	0.25 (3)
anneal/6	94	0.23 (411)	0.19 (500)	1.00 (94)	1.00 (94)	0.56 (170)	1.00 (94)
australian/6	109	0.00 (-)	0.35 (316)	0.88 (124)	1.00 (109)	0.53 (205)	0.86 (127)
car/6	121	0.24 (504)	0.39 (313)	1.00 (121)	0.49 (246)	0.31 (395)	0.91 (133)
cleveland/6	27	0.37 (75)	0.23 (121)	1.00 (27)	0.64 (43)	0.35 (78)	1.00 (27)
lymph/6	0	0.02 (40)	0.08 (12)	0.13 (7)	0.08 (11)	0.01 (74)	0.13 (7)
soybean/6	7	0.03 (237)	0.53 (14)	0.29 (27)	1.00 (7)	0.11 (74)	0.29 (27)
tic-tac-toe/6	164	0.49 (338)	0.57 (287)	0.72 (227)	1.00 (164)	0.46 (360)	0.72 (227)
Best Count	-	0/23	7/23	13/23	8/23	0/23	7/23

Table 2.11: The evaluation of all MaxSAT solvers in incomplete unweighted track of MaxSAT Evaluation 2021.

[Nadel 2022], **TT-Open-WBO-Inc-is** indicates the usage of IntelSAT and tuned for shorter invocations, and **TT-Open-WBO-Inc-g** indicates the usage of *Glucose 4.1* [Audemard & Simon 2018]).

From Table 2.12, we observe that the MaxSAT solver with the best performance in our benchmarks is **Loandra**. The difference between the Loandra solver in *MSE 2022* and the one in *MSE 2021* is the pre-processing step. The latest variant of Loandra employs a recent extension of *MaxPRE* [Korhonen *et al.* 2017], which enables stronger reasoning to improve the upper bound of the number of unsatisfied clauses. Meanwhile, we observe that five benchmarks are used both in *MSE 2021 & MSE 2022*. These benchmarks are marked in cyan. From the comparison of best values found by the solvers in two years, we observe the great improvement of MaxSAT solvers. For example, for benchmark "*splice-1/5*" and "*car/6*", the best values in 2021 are respectively 1122 and 121, but the best values in 2022 are

Dataset/H	Best	DT-Hywalk	Exact	Loandra	noSAT	NuWLS-c	TT-WBO-g	TT-WBO-i	TT-WBO-is
car/3	143	1 (143)	0.583 (246)	1 (143)	0.778 (184)	1 (143)	1 (143)	1 (143)	1 (143)
mushroom/3	7	0.138 (57)	0.006 (1247)	1 (7)	0 (-)	0.138 (57)	1 (7)	1 (7)	1 (7)
tic-tac-toe/3	165	1 (165)	0.722 (229)	0.912 (181)	0.725 (228)	0.912 (181)	0.912 (181)	0.878 (188)	0.878 (188)
anneal/4	83	0.884 (94)	0.764 (109)	1 (83)	0.56 (149)	0.884 (94)	0.884 (94)	0.894 (93)	0.894 (93)
australian/4	59	0.968 (61)	0.385 (155)	1 (59)	0.458 (130)	0.938 (63)	0.923 (64)	0.87 (68)	0.87 (68)
cleveland/4	28	0.879 (32)	0.592 (48)	0.906 (31)	0.302 (95)	0.906 (31)	0.763 (37)	0.784 (36)	0.784 (36)
hypothyroid/4	46	1 (46)	0.228 (205)	1 (46)	0.212 (221)	0.979 (47)	1 (46)	0.959 (48)	1 (46)
lymph/4	3	0.4 (9)	0.182 (21)	0.444 (8)	0.08 (49)	0.667 (5)	0.308 (12)	0.4 (9)	0.4 (9)
splice-1/5	853	0.683 (1250)	0.502 (1700)	0.891 (958)	0 (-)	0.687 (1242)	0.658 (1297)	0.779 (1095)	0.727 (1173)
tumor/5	29	0.938 (31)	0.536 (55)	0.769 (38)	0.469 (63)	1 (29)	0.938 (31)	0.811 (36)	0.811 (36)
cancer/6	5	0.25 (23)	0.1 (59)	0.429 (13)	0.033 (181)	0.545 (10)	0.24 (24)	0.667 (8)	0.667 (8)
car/6	42	1 (42)	0.11 (389)	0.768 (55)	0.086 (501)	0.827 (51)	0.672 (63)	0.782 (54)	0.589 (72)
soybean/6	5	0.857 (6)	0.087 (68)	0.667 (8)	0.07 (85)	0.857 (6)	1 (5)	0.857 (6)	0.857 (6)
Best Count	-	5/13	0/13	7/13	0/13	3/13	4/13	3/13	4/13

Table 2.12: The evaluation of all MaxSAT solvers in incomplete unweighted track of MaxSAT Evaluation 2022.

respectively 853 and 42, which directly shows the progress of MaxSAT solvers.

2.6.2 Incomplete Weighted Track

The incomplete weighted track uses the MaxSAT formulas generated by MaxSAT-DT-adaboost. Considering the number of different benchmarks, each year the *MSE* chooses randomly part of the benchmarks for the incomplete weighted track. We present the summary of evaluations separately by years.

In *MSE 2021*, all MaxSAT solvers evaluated in unweighted track also join the weighted track. Additionally, **Open-WBO-Inc-bmo-complete** (*Open-WBO-c* in short) [Joshi *et al.* 2021] and its variant **Open-WBO-Inc-bmo-satlike** (*Open-WBO-s* in short) are also evaluated in weighted track. Table 2.13 shows the evaluation of different MaxSAT solvers with 300 seconds as time limit. The columns are the same as for the tables shown in unweighted track. From Table 2.13, we observe that **Loandra** is the MaxSAT solver with the best performance, which is far more efficient than others in the weighted track.

In *MSE 2022*, all MaxSAT solvers in unweighted track also join the weighted track. Same as the weighted track in 2021, **Open-WBO-Inc-bmo-complete** and its variant **Open-WBO-Inc-bmo-satlike** are then evaluated. The details of the evaluation are shown in Table 2.14, where the columns are the same as in the previous tables. From Table 2.14, we observe that **Loandra** is also the most efficient solver in the weighted track in 2022. Meanwhile, **DT-Hywalk** shows its competitive performance compared to Loandra for benchmarks in smaller sizes. In addition, we also observe that there are three shared benchmarks used both in *MSE 2021 & MSE 2022*, which are marked in cyan. However, unlike the unweighted track, the results do not show progress in the best values. But, it could also be the cause of the small number of shared benchmarks.

2.7 Summary of Chapter

In this chapter, we firstly introduced details of the previous SAT approach of learning decision trees in the smallest size with perfect accuracy from [Narodytska *et al.* 2018]. Then, we propose the three adaptions to transform the SAT encoding into MaxSAT to find decision trees of depths restricted with best prediction accuracy, aiming to avoid overfitting and increase the generalization performance. Next, we propose the integration of our MaxSAT approach in AdaBoost to improve the prediction performance. Our computational experiments demonstrate at first the competitive prediction quality of our MaxSAT approach comparing with state-of-the-art heuristic and exact methods. Moreover, the progress in generalization performance is observed in the results of the integration of AdaBoost. At the end, we briefly summarized the performance of different MaxSAT solvers in *MaxSAT Evaluation 2021 & 2022* on the formulas generated by our MaxSAT approach.

Dataset/H/Iteration	Best	Exact	Open-WBO-c	Open-WBO-s	Loandra	SATlike-c	Satlike-ck	StableResolve	TT-WBO
anneal/3/2	161	0.78 (206)	0.66 (246)	0.66 (246)	1.00 (161)	0.75 (214)	0.75 (214)	0.75 (214)	0.73 (222)
car/3/6	905	0.72 (1264)	0.73 (1247)	0.80 (1138)	1.00 (905)	0.75 (1211)	0.75 (1211)	0.57 (1590)	0.70 (1291)
cleveland/3/2	64	0.82 (78)	0.76 (85)	0.71 (90)	1.00 (64)	0.86 (75)	0.86 (75)	0.52 (125)	0.90 (71)
tumor/3/2	70	0.79 (89)	0.95 (74)	0.83 (85)	0.93 (75)	1.00 (70)	1.00 (70)	0.71 (99)	1.00 (70)
tic-tac-toe/3/6	496	0.66 (751)	0.87 (573)	0.66 (755)	0.80 (621)	0.93 (536)	0.86 (576)	0.67 (745)	1.00 (496)
anneal/5/3	241	0.56 (428)	0.66 (365)	0.49 (491)	1.00 (241)	0.76 (319)	0.76 (319)	0.65 (370)	0.76 (319)
australian/5/3	196	0.63 (312)	0.57 (343)	0.43 (454)	1.00 (196)	0.72 (273)	0.70 (281)	0.54 (366)	0.65 (300)
australian/5/5	283	0.57 (493)	0.84 (337)	0.55 (511)	1.00 (283)	0.61 (461)	0.68 (417)	0.43 (655)	0.72 (395)
car/5/4	364	0.56 (655)	0.70 (521)	0.30 (1208)	1.00 (364)	0.65 (562)	0.69 (530)	0.33 (1105)	0.76 (478)
car/5/5	446	0.64 (695)	0.61 (733)	0.33 (1339)	1.00 (446)	0.65 (688)	0.58 (773)	0.29 (1536)	0.63 (710)
tumor/5/3	114	0.93 (123)	0.57 (201)	0.55 (209)	1.00 (114)	0.87 (131)	0.87 (131)	0.51 (226)	0.86 (132)
tic-tac-toe/5/4	290	0.61 (474)	1.00 (290)	0.55 (531)	0.80 (365)	0.88 (330)	0.85 (342)	0.48 (611)	0.81 (358)
anneal/6/6	637	0.74 (861)	0.84 (758)	0.61 (1042)	1.00 (637)	0.76 (844)	0.75 (852)	0.64 (991)	0.70 (908)
austrlian/6/6	433	0.74 (583)	0.71 (614)	0.58 (745)	1.00 (433)	0.78 (553)	0.78 (553)	0.61 (709)	0.69 (627)
car/6/3	228	0.33 (694)	0.92 (249)	0.26 (897)	1.00 (228)	0.58 (396)	0.61 (377)	0.29 (789)	0.58 (397)
cleveland/6/3	56	0.52 (108)	0.73 (77)	0.39 (145)	1.00 (56)	0.58 (98)	0.58 (98)	0.43 (132)	0.57 (99)
tumor/6/6	189	0.61 (308)	0.60 (315)	0.45 (424)	1.00 (189)	0.59 (322)	0.58 (329)	0.49 (389)	0.58 (324)
Best Count	-	0/17	1/17	0/17	14/17	1/17	1/17	0/17	2/17

Table 2.13: The evaluation of all MaxSAT solvers in incomplete weighted track of MaxSAT Evaluation 2021.

Dataset/H/Iteration	Best	DT-Hywalk	Exact	Loandra	noSAT	NuWLS-c	Open-WBO-c	Open-WBO-s	TT-WBO-g	TT-WBO-i	TT-WBO-is
anneal/3/2	161	0.750 (215)	0.835 (193)	1.000 (161)	0.753 (214)	0.839 (214)	0.656 (246)	0.651 (248)	0.775 (208)	0.839 (192)	0.757 (213)
australian/3/4	307	0.875 (351)	0.880 (349)	1.000 (307)	0.723 (425)	0.870 (358)	0.665 (462)	0.665 (462)	0.877 (350)	0.870 (353)	0.875 (351)
car/3/2	232	1.000 (232)	0.584 (398)	1.000 (232)	0.546 (426)	0.975 (232)	1.000 (232)	0.696 (334)	0.996 (233)	0.975 (238)	0.975 (238)
tic-tac-toe/3/6	496	1.000 (496)	0.640 (776)	0.799 (621)	0.626 (793)	0.813 (578)	0.866 (573)	0.694 (715)	0.900 (551)	0.813 (610)	0.837 (593)
anneal/4/4	314	0.873 (360)	0.847 (371)	1.000 (314)	0.873 (360)	0.895 (360)	0.778 (404)	0.553 (569)	0.905 (347)	0.895 (351)	0.897 (350)
australian/4/4	319	0.767 (416)	0.764 (418)	1.000 (319)	0.495 (646)	0.816 (420)	0.597 (535)	0.742 (430)	0.767 (416)	0.816 (391)	0.767 (416)
cleveland/4/4	83	0.613 (136)	0.604 (138)	1.000 (83)	0.457 (183)	0.622 (141)	0.651 (128)	0.442 (189)	0.609 (137)	0.622 (134)	0.618 (135)
cleveland/4/5	107	0.603 (178)	0.587 (183)	1.000 (107)	0.527 (204)	0.584 (159)	0.655 (164)	0.464 (232)	0.794 (135)	0.584 (184)	0.584 (184)
anneal/5/4	255	0.645 (396)	0.663 (385)	1.000 (255)	0.574 (445)	0.645 (396)	0.508 (503)	0.345 (741)	0.645 (396)	0.645 (396)	0.645 (396)
australian/5/5	283	0.823 (344)	0.578 (490)	1.000 (283)	0.384 (739)	0.599 (500)	0.840 (337)	0.555 (511)	0.574 (494)	0.599 (473)	0.689 (411)
australian/5/6	288	0.660 (437)	0.606 (476)	1.000 (288)	0.447 (645)	0.607 (456)	0.850 (339)	0.495 (583)	0.660 (437)	0.607 (475)	0.743 (388)
cleveland/5/5	83	0.440 (190)	0.442 (189)	1.000 (83)	0.313 (267)	0.730 (180)	0.583 (143)	0.339 (247)	0.464 (180)	0.730 (114)	0.464 (180)
australian/6/3	207	0.682 (304)	0.698 (297)	1.000 (207)	0.630 (329)	0.756 (291)	0.675 (307)	0.457 (454)	0.762 (272)	0.756 (274)	0.754 (275)
cleveland/6/4	39	0.328 (121)	0.308 (129)	0.667 (59)	0.255 (156)	0.333 (128)	1.000 (39)	0.180 (221)	0.325 (122)	0.333 (119)	0.339 (117)
tumor/6/6	189	0.594 (319)	0.615 (308)	1.000 (189)	0.434 (437)	0.597 (344)	0.601 (315)	0.447 (424)	0.627 (302)	0.597 (317)	0.578 (328)
tumor/4/3	100	0.711 (141)	0.727 (138)	1.000 (100)	0.580 (173)	0.711 (148)	0.716 (140)	0.500 (201)	0.765 (131)	0.711 (141)	0.697 (144)
Best Count	-	2/16	0/16	14/16	0/16	0/16	2/16	0/16	0/16	0/16	0/16

Table 2.14: The evaluation of all MaxSAT solvers in incomplete weighted track of MaxSAT Evaluation 2022.

Chapter 3

Optimizing Binary Decision Diagrams via MaxSAT

Contents

In this chapter, we present our contribution to optimize binary decision diagrams with MaxSAT for classification. This chapter is an extended version of the paper [Hu *et al.* 2022] and contains five sections. Section 3.1 explains the motivation and the target problem. Section 3.2 introduces an essential proposition from [Knuth 2009] relating the binary decision diagram and the truth table for the same Boolean function. Section 3.3 presents the details of the proposed MaxSAT model to learn the optimal binary decision diagrams for classification, and some experimental results. Section 3.5 proposes a simple heuristic pre-processing step to increase the scalability of the proposed MaxSAT model. Section 3.6 briefly summarizes this chapter.

3.1 Motivation and Problem Description

The advantages of binary decision diagrams justify their possible substitution for decision trees in interpretable machine learning, although they fail to gain enough

interest as decision trees. In fact, compared to decision trees, binary decision diagrams could avoid the replication problem and fragmentation problem effectively, which are two flaws of decision trees explained in Section 1.2.2.2.

To the best of our knowledge, [Cabodi *et al.* 2021] is the only exact method of learning optimal binary decision diagrams before our research. This SAT-based exact method extends the core of SAT encoding for decision tree proposed in [Narodytska *et al.* 2018] to learn binary decision diagrams. The target of this approach is to learn optimal binary decision diagrams with the smallest sizes (number of nodes) that correctly classify all examples, which leads to two drawbacks. The first drawback is the possible overfitting due to the perfect accuracy. The other is the lack of restraint in depth of the binary decision diagram learnt, possibly leading the diagram learnt is small in size but high in depth. As the considered binary decision diagrams are *ordered*, this drawback equivalently indicates that this approach could not limit the number of different features used.

To offset these drawbacks, we consider a new target to learn binary decision diagram controlled by depth that optimizes the accuracy. In detail, this target could be described as the following optimisation problem:

- $P^*_{bdd}(\mathcal{E}, H)$*: Given a set of examples* $\mathcal{E}$*, find a binary decision diagram of depth* H *that maximises the number of examples in* $\mathcal{E}$ *that are correctly classified.*

This problem shares same objective with $P^*_{dt}(\mathcal{E}, H)$, the optimisation problem proposed to learn optimal decision trees in chapter 2. Therefore, inspired by the solving methodology of our previous research, we firstly introduce a SAT-based model to find the binary decision diagrams with the smallest number of features classifying all examples correctly, which is described as the following decision problem:

- $P_{bdd}(\mathcal{E}, H)$*: Given a set of examples* $\mathcal{E}$*, is there a binary decision diagram of depth* H *that classifies correctly all examples in* $\mathcal{E}$*?*

Then, we introduce a lifted MaxSAT-based model to solve the optimisation problem $P^*_{bdd}(\mathcal{E}, H)$. An additional motivation of our research is to face some scalability issues highlighted in Chapter 2. In practice, as the binary decision diagrams have smaller sizes than the corresponding decision trees, the MaxSAT formula for optimizing binary decision diagrams is lighter than the one for decision trees with the same objective. The shrink of encoding size could reduce the time to report optimality, or deal with larger datasets within same limited time. Moreover, in order to increase the scalability of our MaxSAT approach, we propose a heuristic extension based on a simple pre-processing step. The details are shown in Section 3.5.

Almost at the same time, [Florio *et al.* 2022] proposes the first MILP-based exact method for learning optimal decision diagrams. However, we consider this method is quite incomparable because of the difference in the topology. In [Florio *et al.* 2022], the decision diagrams are limited by a *preset skeleton*, and are not *ordered*. We refer readers to Section 1.3.1.2 that provides the literature review of the exact methods for optimal (binary) decision diagrams.

Before introducing the details of the proposed SAT & MaxSAT encoding for optimal binary decision diagrams, we present an essential proposition from [Knuth 2009] in the next section. As in Chapter 2, we consider binary datasets $\mathcal{E}$ containing M examples, and K binary features.

Additionally, for all possible uses of cardinality constraints in this chapter, we model the cardinality constraints by the sequential counters encodings proposed in [Sinz 2005].

3.2 An Essential Proposition

In this section, we present an essential proposition from [Knuth 2009], which relates the truth table and the binary decision diagram of the same Boolean function. In other words, this proposition shows how to build the binary decision diagram structure by the corresponding truth table.

Firstly, we introduce the basic idea and notations for a *truth table*. Let g be a Boolean function defined over a sequence of n Boolean variables $[\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}]$. The function g can be represented by a truth table, which is a binary string of size 2^n listing values of all assignments of the n variables. A truth table β of length 2^n is called to be of order n.

Then, we describe the *subtables* of a truth table, which are defined recursively. A truth table β of order $n \geq 1$ can be represented by $\beta_0\beta_1$, where β_0 and β_1 are truth table of order $n-1$, indicating the left and right part of β. Therefore, β_0 and β_1 are called the subtables of β. The subtables of subtables are also considered as subtables, and a table is considered as a subtable of itself. All the subtables are unique without duplication.

Next, we explain the concept of *bead*. A bead of order n ($n \geq 1$) is a truth table β of order n that does not contain identical subtables. More formally, $\beta = \beta_0\beta_1$ is a bead if $\beta_0 \neq \beta_1$. Equivalently speaking, bead is a restricted truth table that avoids identical left and right parts. Especially, 0 and 1 are two special values that happens to be bead. In addition, the *beads* of a Boolean function g are the subtables of its truth table that happens to be bead.

Example 8 *Considering a Boolean function* $g_1(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) = (\mathrm{x}_1 \vee \mathrm{x}_2 \vee \mathrm{x}_3) \wedge (\neg\mathrm{x}_1 \vee \mathrm{x}_2 \vee \neg\mathrm{x}_3) \wedge (\neg\mathrm{x}_1 \vee \neg\mathrm{x}_2 \vee \neg\mathrm{x}_3)$*, the associated truth table* β_{g_1} *is* 01111010 *and is of order* 3 *(its size is* $8 = 2^3$*).*

The subtables of β_{g_1} *are* $\{01111010, 0111, 1010, 01, 11, 10, 0, 1\}$*. Duplicated subtables are eliminated. For instance the subtable* 1010 *of this example, produces only one subtable* 10*.*

In the set of subtables of β_{g_1}*,* 01111010*,* 0111*,* 01*,* 10*,* 0*, and* 1 *are beads;* 1010 *and* 11 *are not beads.*

The proposition that links binary decision diagram and truth table is based on the concept of beads, it is described as follow:

Proposition 1 *All vertices of a binary decision diagram, are in one-to-one correspondence with the beads of the Boolean function g it represents.*

Based on Proposition 1, we can produce the binary decision diagram of a Boolean function, by combining its beads and its sequence of variables.

Example 9 *Continuing with Example 8, the beads of Boolean function $g_1(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$ are $\{01111010, 0111, 01, 10, 0, 1\}$. We can represent all the subtables of the truth table of the function g_1 as illustrated in the left one of Figure 3.1. From Proposition 1, we can remove subtables that are not beads as their corresponding nodes have the left and right child, and produce the binary decision diagram illustrated in the middle one of Figure 3.1. Then, we can replace the beads by vertices associated with the sequence of Boolean variables $[\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3]$. The final binary decision diagram for $g_1(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$ is shown in the right part of Figure 3.1.*

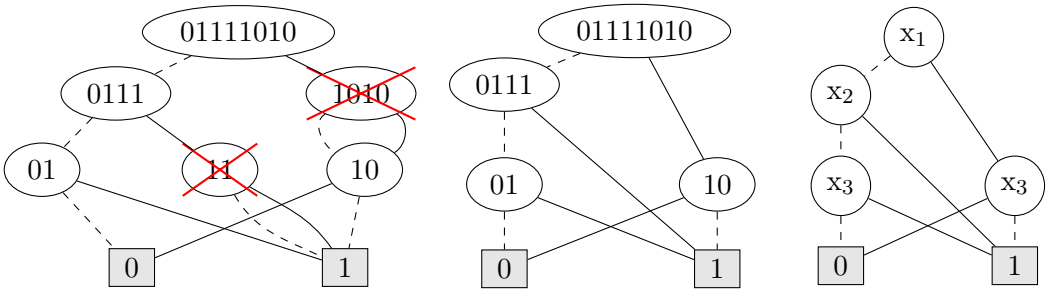


Figure 3.1: The binary decision diagram for Boolean function $g_1(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$.

Function(Input): Output	**Description**
$FirstHalf(string\ s)$: string	Returns the first half of s
$SecondHalf(string\ s)$: string	Returns the second half of s
$IsBead(string\ s)$: Boolean	Returns *True* iff s is a bead
$LeadToZero(string\ s)$: Boolean	Returns *True* iff s contains only 0
$LeadToOne(string\ s)$: Boolean	Returns *True* iff s contains only 1

Table 3.1: Some predefined functions in the algorithm.

Finally, we describe the algorithm to construct a binary decision diagram of maximum depth H using the beads of the truth table β associated to a sequence of variables $[\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{H}]$, based on the proposition 1. The detailed algorithm is described in Algorithm 7, and some predefined functions are listed in Table 3.1.

The algorithm creates nodes level by level in a *breadth-first* way. In detail, firstly, the binary decision diagram built is defined by the combination of *a list of nodes* and *a list of edges*. Each node is a pair (*node_id, variable*). The value of *node_id* is a unique integer called the `id` of the node, which is non-negative for non-terminal nodes. There are two terminal nodes: the node $(-1, 1)$ associated to the value 1 (i.e., positive class in the context of binary classification), and the node

Algorithm 7: GenBDD($\beta, \mathcal{X}$), an algorithm to construct a BDD from a given string β and variable sequence $\mathcal{X}$.

Input: String β, variable sequence $\mathcal{X} = [\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{H}]$.

```
1  nodes ← {}; edges ← {}; T ← {}
2  nodes.append((−1, 1)); nodes.append((−2, 0))
3  q ← Queue()
4  q.put((β, 0, 1, ∅)))
5  while not q.empty() do
6      (s, parent_id, level, direction) ← q.pop())
7      if Length(s) > 1 and IsBead(s) then
           // When the current string s is a Bead.
8          if s ∉ T then
               // s is a new string, i.e., not seen before
9              T.append(s)
10             index ← T.index(s) + 1
11             nodes.append((index, x_level))
12         index ← T.index(s) + 1
13         if parent_id ≥ 1 then
14             edges.append((parent_id, index, direction))
           // Put the left and right child into the queue.
15         q.put((FirstHalf(s), index, level + 1, left))
16         q.put((SecondHalf(s), index, level + 1, right))
17     else if Length(s) > 1 and not IsBead(s) then
           // When the current string s is not a Bead.
18         if LeadToOne(s) or LeadToZero(s) then
               // s leads to sink nodes
19             if LeadToOne(s) then
20                 sink ← −1
21             else
22                 sink ← −2
23             if p = 0 then
24                 edges.append((1, sink, left))
25                 edges.append((1, sink, right))
26             else
27                 edges.append((parent_id, sink, direction))
28         else
               // Otherwise put the left child into the queue.
29             q.put((FirstHalf(s), parent_id, level + 1 , direction))
30     else
           // The current string is a sink node.
31         if s = 1 then
32             sink ← −1
33         else
34             sink ← −2
35         edges.append((parent_id, sink, direction))
```

Output: BDD(*nodes*, *edges*)

$(-2, 0)$ associated to the value 0 (i.e., negative class in binary classification). Each edge is a tuple $(p, c, direction)$, where p is the `id` of the parent node, c is the `id` of the child node, and $direction \in \{left, right\}$) indicates if c is the left or right child of p.

Algorithm 7 uses a FIFO queue q, in which each item follows the format $(str, parent\_id, current\_level, direction)$. The first item pushed in the queue is a special case indicating the root, denoted by $(\beta, 0, 1, \emptyset)$, since the root has no parent.

At each iteration of the main loop, the algorithm pops an element of format $(s, parent\_id, level, direction)$ from the queue at Line 6. If s is a bead, the algorithm creates a new node at Line 11 associated with the level $level$ if s is seen for the first time. The set of edges is updated in Line 14 accordingly. The set of left and right children of s are added in the queue in Lines 15 and 16.

When the current string s is not a bead of size > 1, there might be two cases where s leads directly to a terminal node. Either s contains only 0s, or s contains only 1s. Depending on the size of s, the two cases are handled in two parts of the algorithm: from Line 17 to Line 27, and from Line 30 to Line 34, The case where s is not a bead that do not lead to a terminal node, only one child of s is added to the queue without creating nodes (since s is not a bead). The algorithm ends when all the elements of the queue are treated.

3.3 Proposed SAT and MaxSAT Models

In this section, we present our approach of learning optimal binary decision diagrams for binary classification. At first, we describe a SAT-based model to solve the decision problem $P_{bdd}(\mathcal{E}, H)$. Then, two improved versions are proposed to reduce the encoding size. Next, we show how to lift the SAT-based model into MaxSAT model to solve the optimisation problem $P^*_{bdd}(\mathcal{E}, H)$. In addition, we propose a post-processing procedure to merge compatible subtrees. Finally, we provide an experimental study to evaluate empirically our models and the compare them between state-of-the-art heuristic and exact methods.

3.3.1 An Initial SAT Model: `BDD1`

As Proposition 1 shows, a binary decision diagram of depth H could be constructed by the combination of a sequence of Boolean variables of size H: $[\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{H}]$, and a truth table β of order H associated to a Boolean function. For binary classification, in order to build a binary decision diagram of depth H, we aim to find a sequence of binary features of size H that maps one-to-one the sequence of Boolean variables. To build the binary decision diagram for binary classification, instead of using the sequence of Boolean variables, we consider to find a sequence of binary features of same size that maps one-to-one the sequence of Boolean variables.

To solve the classification problem $P_{bdd}(\mathcal{E}, H)$, we need to find the feature ordering of size H, and a truth table β associated to a Boolean function that correctly

classifies all examples of the dataset $\mathcal{E}$. Therefore, the SAT encoding contains two parts:

- ***Part 1**: Constraints to select features of the dataset into the feature ordering of size H.*
- ***Part 2**: Constraints to generate a truth table that classifies correctly all examples of $\mathcal{E}$ with the feature ordering found in the previous part.*

We introduce two sets of Boolean variables. These sets are described in Table 3.2. Variables $\boldsymbol{a_r^i}$ indicate whether the feature f_r is selected as i-th feature in the feature ordering. Variables $\boldsymbol{c_j}$ stores the information of the j-th value of the truth table/ The definitions of these two Boolean variables are shown in Table 3.2.

	Description of The Variables
a_r^i	**1** iff feature f_r is selected as i-th feature in the feature ordering, **0** otherwise $\forall i \in \{1, H\}, \forall r \in \{1, K\}$
c_j	**1** iff j-th value of the truth table is 1, **0** otherwise $\forall j \in \{1, 2^H\}$

Table 3.2: Description of the Boolean variables used in the SAT encoding of a binary decision diagram.

Using these sets of variables, we present the constraints of *Part 1* that capture the ordered restriction. At first, any feature f_r can be selected at most once to avoid the duplication, which is for $r = 1, \ldots, K$:

$$\sum_{i=1}^{H} a_r^i \leq 1 \tag{3.1}$$

Then, there is exactly one feature selected for each index of the feature ordering:

$$\sum_{r=1}^{K} a_r^i = 1 \tag{3.2}$$

Next, to avoid the first feature selected to make useless splits, we need to ensure that the truth table found is a bead.

$$\bigvee_{j=1}^{2^{H-1}} (c_j \oplus c_{j+2^{H-1}}) \tag{3.3}$$

Now, we explain the constraints of *Part 2* that generate a truth table classifying all examples correctly. First, we consider the relationship observed between the values of a truth table and the assignments of the given sequence of Boolean variables. As each value in the truth table corresponds to a unique assignment, we can define a function $rel(i, j)$ to obtain the value of the i-th feature in the feature

ordering of the size H, when given the j-th value in the truth table.

$$rel(i,j) = \lfloor \frac{j-1}{2^{H-i}} \rfloor \mod 2, \quad i \in [1,H], j \in [1,2^H] \tag{3.4}$$

A typical example is the first value of the truth table, which is reachable for the assignment that all features are assigned as zero. The usage of this relationship function is to decide if an example could arrive at a given value in the truth table with the given feature ordering. In detail, for an example $e_q \in \mathcal{E}$, we denote its value of feature f_r as $\sigma(r,q)$. If $\sigma(r,q) = rel(i,j)$, then, for example e_q, the feature f_r can be at the i-th position in the feature ordering to let the example reach the j-th value in the truth table. Oppositely, $\sigma(r,q) \neq rel(i,j)$ indicates that if feature f_r is selected as the i-th one in the feature ordering, the j-th value in the truth table is not reachable for the example e_q.

To classify all examples correctly, the idea is to ensure that no example following an assignment leads to a value in the truth table with its opposite class. Thus, we propose the following constraints for classification. Let e_q be a positive example (noted as $e_q \in \mathcal{E}^+$), for all values in the truth table, with $j = 1, \ldots, 2^H$:

$$\neg c_j \rightarrow \bigvee_{i=1}^{H} \bigvee_{r=1}^{K} (a_r^i \wedge rel(i,j) \oplus \sigma(r,q)) \tag{3.5}$$

That is, for each positive example e_q, any negative j-th value in the truth table must contain at least one feature f_r in its corresponding position i that raise the inequality between $\sigma(r,q)$ and $rel(i,j)$, so that e_q must not lead to any negative value. Analogously, we apply this idea to negative examples, which considers any positive values in the truth table. Let e_q be a negative example e_q (noted as $e_q \in \mathcal{E}^-$), for all values in the truth table, with $j = 1, \ldots, 2^H$:

$$c_j \rightarrow \bigvee_{i=1}^{H} \bigvee_{r=1}^{K} (a_r^i \wedge rel(i,j) \oplus \sigma(r,q)) \tag{3.6}$$

Example 10 *At first, we recall the binary dataset in Table 3.3 used in Example 3 and the decision tree of size* 5 *that classifies all of the examples shown in Figure 3.2. The dataset contains* 4 *binary features numered from left to right.*

We consider to encode a binary decision diagram with depth $H = 2$*. Therefore, two sets of variables are introduced, including* $\{a_1^1, a_1^2, a_2^1, a_2^2, a_3^1, a_3^2, a_4^1, a_4^2\}$ *and* $\{c_1, c_2, c_3, c_4\}$.

Using the propositional variables described, the Constraints 3.1, 3.2 and 3.3 are:

$$a_1^1 + a_1^2 \leq 1, \quad a_2^1 + a_2^2 \leq 1, \quad a_3^1 + a_3^2 \leq 1, \quad a_4^1 + a_4^2 \leq 1,$$
$$a_1^1 + a_2^1 + a_3^1 + a_4^1 = 1, \quad a_1^2 + a_2^2 + a_3^2 + a_4^2 = 1,$$
$$(c_1 \oplus c_3) \vee (c_2 \oplus c_4)$$

We now detail the constraints for classification (i.e., Constraint 3.5 and 3.6).

Ex.	L	C	E	S	H
e_1	1	0	1	0	0
e_2	1	0	0	1	0
e_3	0	0	1	0	1
e_4	1	1	0	0	0
e_5	0	0	0	1	1
e_6	1	1	1	1	0
e_7	0	1	1	0	0
e_8	0	0	1	1	1

Table 3.3: The toy dataset from [Narodytska *et al.* 2018].

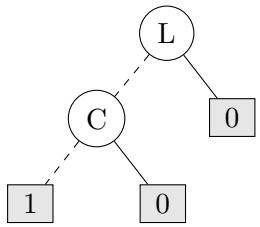


Figure 3.2: The decision tree classifying all examples of Table 3.3.

The example e_1 (Table 3.3) is a negative one: $e_1 \in \mathcal{E}^-$, we then apply Constraint 3.6. We remind the feature vector of e_1 is (1010)*. For the value of variable c_1, we have:*

$$\begin{aligned} c_1 \rightarrow &(a_1^1 \wedge 0 \oplus 1) \vee (a_2^1 \wedge 0 \oplus 0) \vee (a_3^1 \wedge 0 \oplus 1) \vee (a_4^1 \wedge 0 \oplus 0) \\ \vee &(a_1^2 \wedge 0 \oplus 1) \vee (a_2^2 \wedge 0 \oplus 0) \vee (a_3^2 \wedge 0 \oplus 1) \vee (a_4^2 \wedge 0 \oplus 0) \end{aligned}$$

This could be simplified as the following clause:

$$\neg c_1 \vee a_1^1 \vee a_3^1 \vee a_1^2 \vee a_3^2$$

Similarly, we could generate other classification constraints for each example.

For the negative example e_1, whose feature vector is (1010)*:*

$$\begin{aligned} &\neg c_1 \vee a_1^1 \vee a_3^1 \vee a_1^2 \vee a_3^2, \quad \neg c_2 \vee a_1^1 \vee a_3^1 \vee a_2^2 \vee a_4^2, \\ &\neg c_3 \vee a_2^1 \vee a_4^1 \vee a_1^2 \vee a_3^2, \quad \neg c_4 \vee a_2^1 \vee a_4^1 \vee a_2^2 \vee a_4^2, \end{aligned}$$

For the negative example e_2, whose feature vector is (1001)*:*

$$\begin{aligned} &\neg c_1 \vee a_1^1 \vee a_4^1 \vee a_1^2 \vee a_4^2, \quad \neg c_2 \vee a_1^1 \vee a_4^1 \vee a_2^2 \vee a_3^2, \\ &\neg c_3 \vee a_2^1 \vee a_3^1 \vee a_1^2 \vee a_4^2, \quad \neg c_4 \vee a_2^1 \vee a_3^1 \vee a_2^2 \vee a_3^2, \end{aligned}$$

For the positive example e_3, whose feature vector is (0010)*:*

$$\begin{aligned} &c_1 \vee a_3^1 \vee a_3^2, \quad c_2 \vee a_3^1 \vee a_1^2 \vee a_2^2 \vee a_4^2, \\ &c_3 \vee a_1^1 \vee a_2^1 \vee a_4^1 \vee a_3^2, \quad c_4 \vee a_1^1 \vee a_2^1 \vee a_3^1 \vee a_1^2 \vee a_2^2 \vee a_4^2, \end{aligned}$$

For the negative example e_4, whose feature vector is (1100)*:*

$$\begin{aligned} &\neg c_1 \vee a_1^1 \vee a_2^1 \vee a_1^2 \vee a_2^2, \quad \neg c_2 \vee a_1^1 \vee a_2^1 \vee a_3^2 \vee a_4^2, \\ &\neg c_3 \vee a_3^1 \vee a_4^1 \vee a_1^2 \vee a_2^2, \quad \neg c_4 \vee a_3^1 \vee a_4^1 \vee a_3^2 \vee a_4^2, \end{aligned}$$

For the positive example e_5, whose feature vector is (0001)*:*

$$c_1 \vee a_4^1 \vee a_4^2, \quad c_2 \vee a_4^1 \vee a_1^2 \vee a_2^2 \vee a_3^2,$$
$$c_3 \vee a_1^1 \vee a_2^1 \vee a_3^1 \vee a_4^2, \quad c_4 \vee a_1^1 \vee a_2^1 \vee a_3^1 \vee a_1^2 \vee a_2^2 \vee a_3^2,$$

For the negative example e_6, whose feature vector is (1111)*:*

$$\neg c_1 \vee a_1^1 \vee a_2^1 \vee a_4^1 \vee a_1^2 \vee a_2^2 \vee a_3^2 \vee a_4^2, \quad \neg c_4,$$
$$\neg c_2 \vee a_1^1 \vee a_2^1 \vee a_3^1 \vee a_4^1, \quad \neg c_3 \vee a_1^2 \vee a_2^2 \vee a_3^2 \vee a_4^2,$$

For the negative example e_7, whose feature vector is (0110)*:*

$$\neg c_1 \vee a_2^1 \vee a_3^1 \vee a_2^2 \vee a_3^2, \quad \neg c_2 \vee a_2^1 \vee a_3^1 \vee a_1^2 \vee a_4^2,$$
$$\neg c_3 \vee a_1^1 \vee a_4^1 \vee a_2^2 \vee a_3^2, \quad \neg c_4 \vee a_1^1 \vee a_4^1 \vee a_1^2 \vee a_4^2,$$

For the positive example e_8, whose feature vector is (0011)*:*

$$c_1 \vee a_3^1 \vee a_4^1 \vee a_3^2 \vee a_4^2, \quad c_2 \vee a_3^1 \vee a_4^1 \vee a_1^2 \vee a_2^2,$$
$$c_3 \vee a_1^1 \vee a_2^1 \vee a_3^2 \vee a_4^2, \quad c_4 \vee a_1^1 \vee a_2^1 \vee a_1^2 \vee a_2^2,$$

An assignment satisfying all constraints is the following:

$$a_1^1 = 1, a_1^2 = 0, a_2^1 = 0, a_2^2 = 1, a_3^1 = 0, a_3^2 = 0, a_4^1 = 0, a_4^2 = 0$$
$$c_1 = 1, c_2 = 0, c_3 = 0, c_4 = 0$$

This assignment indicates that the feature ordering is $[L, C]$*, and the truth table found is* 1000*. Table 3.4 illustrates the relationship between the values of truth table and the assignments of the given feature ordering. Figure 3.3 shows the corresponding binary decision diagram, which provides more compact representation than the decision tree shown in Figure 3.2.*

x_1 =L	x_2 =C	
0	0	$c_1 = 1$
0	1	$c_2 = 0$
1	0	$c_3 = 0$
1	1	$c_4 = 0$

Table 3.4: Truth table solution for the binary decision diagram of depth 2 classifying all examples of Table 3.3.

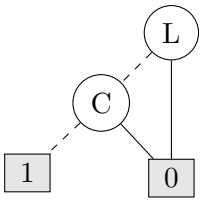


Figure 3.3: The built binary decision diagram.

To simplify the notation, in the following, we refer to this proposed SAT encoding for the problem $P_{bdd}(\mathcal{E}, H)$ as `BDD1`. We now present a theoretical analysis of the model size for `BDD1`. As we consider the dataset $\mathcal{E}$ contains M examples with K binary features, the encoding size of `BDD1` (in the number of literals) for the binary decision diagram of depth H is in $\boldsymbol{O(M \times H \times K \times 2^H)}$. The term

$M \times 2^H$ results from Constraints 3.5 and 3.6, each contains $O(H \times K)$ literals. For the remaining constraints, it is in $O(H \times K)$ for Constraints 3.1 and 3.2, $O(2^H)$ for Constraint 3.3.

The encoding size of BDD1 is quite huge due to the size of clauses generated by the constraints for classification. The massive amount of encoding size makes BDD1 impractical in practice.

3.3.2 A Second SAT Model: BDD2

In order to reduce the encoding size of BDD1, we propose to replace the original constraints for classification (i.e., Constraint 3.5 and 3.6), as they are the key factor of the high complexity. Another idea to realize the perfect classification is to let every positive (respectively, negative) example following an assignment that arrives at a positive (respectively, negative) value of the truth table.

As defined in the previous subsection, Equation 3.4 describes the relationship between the value of the i-th feature in the feature ordering and the j-th value in the truth table. This equation is effective to verify whether an example is in the correct way to a given value of the truth table. However, it fails to provide any information in which value of the truth table that the example leads to. Therefore, we introduce a new set of Boolean variables d_i^q to catch the value of example e_q for the i-th feature selected in the feature ordering. The definition of d_i^q is shown in Table 3.5.

	Description of The Variables
d_i^q	**1** iff for example e_q, the value of the i-th feature selected in the feature ordering is 1, **0** otherwise $\forall i \in \{1, H\}$, $\forall q \in \{1, M\}$

Table 3.5: Description of new propositional Boolean variables for the improved SAT encoding for binary decision diagrams.

We then propose constraints to relate the variables d_i^q and a_r^i based on the values of features of each example. For each example $e_q \in \mathcal{E}$, with $i = 1, \ldots, H$, $r = 1, \ldots, K$:

$$\begin{aligned} a_r^i &\rightarrow d_i^q \qquad \text{if } \sigma(q, r) = 1 \\ a_r^i &\rightarrow \neg d_i^q \qquad \text{if } \sigma(q, r) = 0 \end{aligned} \tag{3.7}$$

With the conjunction of variables d_i^q, one capture which value in the truth table that the example leading to. Therefore, fir each $e_q \in \mathcal{E}^+$, we have 2^H constraints to ensure it is correctly classified:

$$\begin{aligned} &\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge \neg d_H^q \rightarrow c_1 \\ &\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge d_H^q \rightarrow c_2 \\ &\qquad \cdots \\ &d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge \neg d_H^q \rightarrow c_{2^H-1} \\ &d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge d_H^q \rightarrow c_{2^H} \end{aligned} \tag{3.8}$$

That is, for any positive example following an assignment of the feature ordering, they lead to a positive value in the truth table. For the 2^H implication constraints, only one condition of them is satisfied, which indicates the real path of example e_q. Similarly, for any $e_q \in \mathcal{E}^-$, we also have 2^H constraints to ensure that it leads to a negative value in the truth table:

$$\begin{aligned} \neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge \neg d_H^q &\rightarrow \neg c_1 \\ \neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge d_H^q &\rightarrow \neg c_2 \\ \cdots & \\ d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge \neg d_H^q &\rightarrow \neg c_{2^H-1} \\ d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge d_H^q &\rightarrow \neg c_{2^H} \end{aligned} \tag{3.9}$$

Example 11 *Similarly to the beginning of Example 10, we firstly generate the same constraints as Constraints 3.1, 3.2 and 3.3. Next, the constraints generated based on Constraint 3.7 are:*

For the negative example e_1, whose feature vector is (1010)*:*

$$\begin{array}{cccc} \neg a_1^1 \vee d_1^1, & \neg a_2^1 \vee \neg d_1^1, & \neg a_3^1 \vee d_1^1, & \neg a_4^1 \vee \neg d_1^1, \\ \neg a_1^2 \vee d_2^1, & \neg a_2^2 \vee \neg d_2^1, & \neg a_3^2 \vee d_2^1, & \neg a_4^2 \vee \neg d_2^1, \end{array}$$

For the negative example e_2, whose feature vector is (1001)*:*

$$\begin{array}{cccc} \neg a_1^1 \vee d_1^2, & \neg a_2^1 \vee \neg d_1^2, & \neg a_3^1 \vee \neg d_1^2, & \neg a_4^1 \vee d_1^2, \\ \neg a_1^2 \vee d_2^2, & \neg a_2^2 \vee \neg d_2^2, & \neg a_3^2 \vee \neg d_2^2, & \neg a_4^2 \vee d_2^2, \end{array}$$

For the positive example e_3, whose feature vector is (0010)*:*

$$\begin{array}{cccc} \neg a_1^1 \vee \neg d_1^3, & \neg a_2^1 \vee \neg d_1^3, & \neg a_3^1 \vee d_1^3, & \neg a_4^1 \vee \neg d_1^3, \\ \neg a_1^2 \vee \neg d_2^3, & \neg a_2^2 \vee \neg d_2^3, & \neg a_3^2 \vee d_2^3, & \neg a_4^2 \vee \neg d_2^3, \end{array}$$

For the negative example e_4, whose feature vector is (1100)*:*

$$\begin{array}{cccc} \neg a_1^1 \vee d_1^4, & \neg a_2^1 \vee d_1^4, & \neg a_3^1 \vee \neg d_1^4, & \neg a_4^1 \vee \neg d_1^4, \\ \neg a_1^2 \vee d_2^4, & \neg a_2^2 \vee d_2^4, & \neg a_3^2 \vee \neg d_2^4, & \neg a_4^2 \vee \neg d_2^4, \end{array}$$

For the positive example e_5, whose feature vector is (0001)*:*

$$\begin{array}{cccc} \neg a_1^1 \vee \neg d_1^5, & \neg a_2^1 \vee \neg d_1^5, & \neg a_3^1 \vee \neg d_1^5, & \neg a_4^1 \vee d_1^5, \\ \neg a_1^2 \vee \neg d_2^5, & \neg a_2^2 \vee \neg d_2^5, & \neg a_3^2 \vee \neg d_2^5, & \neg a_4^2 \vee d_2^5, \end{array}$$

For the negative example e_6, whose feature vector is (1111)*:*

$$\begin{array}{cccc} \neg a_1^1 \vee d_1^6, & \neg a_2^1 \vee d_1^6, & \neg a_3^1 \vee d_1^6, & \neg a_4^1 \vee d_1^6, \\ \neg a_1^2 \vee d_2^6, & \neg a_2^2 \vee d_2^6, & \neg a_3^2 \vee d_2^6, & \neg a_4^2 \vee d_2^6, \end{array}$$

For the negative example e_7, whose feature vector is (0110)*:*

$$\begin{array}{llll} \neg a_1^1 \vee \neg d_1^7, & \neg a_2^1 \vee d_1^7, & \neg a_3^1 \vee d_1^7, & \neg a_4^1 \vee \neg d_1^7, \\ \neg a_1^2 \vee \neg d_2^7, & \neg a_2^2 \vee d_2^7, & \neg a_3^2 \vee d_2^7, & \neg a_4^2 \vee \neg d_2^7, \end{array}$$

For the positive example e_8, whose feature vector is (0011)*:*

$$\begin{array}{llll} \neg a_1^1 \vee \neg d_1^8, & \neg a_2^1 \vee \neg d_1^8, & \neg a_3^1 \vee d_1^8, & \neg a_4^1 \vee d_1^8, \\ \neg a_1^2 \vee \neg d_2^8, & \neg a_2^2 \vee \neg d_2^8, & \neg a_3^2 \vee d_2^8, & \neg a_4^2 \vee d_2^8, \end{array}$$

To make the classification, the constraints generated based on Constraint 3.8 and 3.9 are shown as follow:

$$\begin{array}{lllll} e_1 \in \mathcal{E}^- : & d_1^1 \vee d_2^1 \vee \neg c_1, & d_1^1 \vee \neg d_2^1 \vee \neg c_2, & \neg d_1^1 \vee d_2^1 \vee \neg c_3, & \neg d_1^1 \vee \neg d_2^1 \vee \neg c_4, \\ e_2 \in \mathcal{E}^- : & d_1^2 \vee d_2^2 \vee \neg c_1, & d_1^2 \vee \neg d_2^2 \vee \neg c_2, & \neg d_1^2 \vee d_2^2 \vee \neg c_3, & \neg d_1^2 \vee \neg d_2^2 \vee \neg c_4, \\ e_3 \in \mathcal{E}^+ : & d_1^3 \vee d_2^3 \vee c_1, & d_1^3 \vee \neg d_2^3 \vee c_2, & \neg d_1^3 \vee d_2^3 \vee c_3, & \neg d_1^3 \vee \neg d_2^3 \vee c_4, \\ e_4 \in \mathcal{E}^- : & d_1^4 \vee d_2^4 \vee \neg c_1, & d_1^4 \vee \neg d_2^4 \vee \neg c_2, & \neg d_1^4 \vee d_2^4 \vee \neg c_3, & \neg d_1^4 \vee \neg d_2^4 \vee \neg c_4, \\ e_5 \in \mathcal{E}^+ : & d_1^5 \vee d_2^5 \vee c_1, & d_1^5 \vee \neg d_2^5 \vee c_2, & \neg d_1^5 \vee d_2^5 \vee c_3, & \neg d_1^5 \vee \neg d_2^5 \vee c_4, \\ e_6 \in \mathcal{E}^- : & d_1^6 \vee d_2^6 \vee \neg c_1, & d_1^6 \vee \neg d_2^6 \vee \neg c_2, & \neg d_1^6 \vee d_2^6 \vee \neg c_3, & \neg d_1^6 \vee \neg d_2^6 \vee \neg c_4, \\ e_7 \in \mathcal{E}^- : & d_1^7 \vee d_2^7 \vee \neg c_1, & d_1^7 \vee \neg d_2^7 \vee \neg c_2, & \neg d_1^7 \vee d_2^7 \vee \neg c_3, & \neg d_1^7 \vee \neg d_2^7 \vee \neg c_4, \\ e_8 \in \mathcal{E}^+ : & d_1^8 \vee d_2^8 \vee c_1, & d_1^8 \vee \neg d_2^8 \vee c_2, & \neg d_1^8 \vee d_2^8 \vee c_3, & \neg d_1^8 \vee \neg d_2^8 \vee c_4, \end{array}$$

An assignment satisfying all constraints corresponds to:

$$\begin{array}{c} d_1^1 = 1, d_2^1 = 0, d_1^2 = 1, d_2^2 = 0, d_1^3 = 0, d_2^3 = 0, d_1^4 = 1, d_2^4 = 1, \\ d_1^5 = 0, d_2^5 = 0, d_1^6 = 1, d_2^6 = 1, d_1^7 = 0, d_2^7 = 1, d_1^8 = 0, d_2^8 = 0, \\ a_1^1 = 1, a_1^2 = 0, a_2^1 = 0, a_2^2 = 1, a_3^1 = 0, a_3^2 = 0, a_4^1 = 0, a_4^2 = 0 \\ c_1 = 1, c_2 = 0, c_3 = 0, c_4 = 0 \end{array}$$

It is easy to observe that this assignment constructs the same binary decision diagram as the one obtained in Example 10 and shown in Figure 3.3.

We refer to this improved SAT encoding for the problem $P_{bdd}(\mathcal{E}, H)$ as `BDD2`. The complexity of encoding size (the number of literals) of the `BDD2` model is in $\boldsymbol{O(M \times H \times (2^H + K))}$, for a binary decision diagram of depth H. The term $O(M \times H \times K)$ results from Constraint 3.7. To make the classification for each example, there are 2^H clauses containing $H + 1$ literals, which account for the term $O(M \times H \times 2^H)$.

Recall that the complexity of the encoding size of `BDD1` is $O(M \times H \times K \times 2^H)$. Compared to `BDD1`, we observe a clear theoretical advantage of `BDD2` in terms of the encoding size, thus the scalability. In addition, the additional Constraint 3.7 is in format of *2-SAT*, which is easily propagated.

3.3.3 A Third SAT Model: BDD3

As analysed in the end of the previous subsection, compared to BDD1, BDD2 benefits a lot from its classification constraints (Constraints 3.8 and 3.9). However, as each clause in the constraints for classification is *(H + 1)-SAT*, we are afraid of the problem of scalability when the depth H grows. Observe, however, that there are massive duplication of information in the conditions of the classification constraints. To view directly the duplication, consider a simple case that classifying a positive example e_q by the binary decision diagram of depth 4. The constraint for classification is generated based on Constraint 3.8. In Figure 3.4, we list all duplicated part of the constraints, by using boxes of different colors.

$$\neg d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{\neg d_3^q \wedge \neg d_4^q}} \rightarrow c_1 \qquad \neg d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{\neg d_3^q \wedge d_4^q}} \rightarrow c_2$$

$$\neg d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{d_3^q \wedge \neg d_4^q}} \rightarrow c_3 \qquad \neg d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{d_3^q \wedge d_4^q}} \rightarrow c_4$$

$$\neg d_1^q \wedge \boxed{d_2^q \wedge \boxed{\neg d_3^q \wedge \neg d_4^q}} \rightarrow c_5 \qquad \neg d_1^q \wedge \boxed{d_2^q \wedge \boxed{\neg d_3^q \wedge d_4^q}} \rightarrow c_6$$

$$\neg d_1^q \wedge \boxed{d_2^q \wedge \boxed{d_3^q \wedge \neg d_4^q}} \rightarrow c_7 \qquad \neg d_1^q \wedge \boxed{d_2^q \wedge \boxed{d_3^q \wedge d_4^q}} \rightarrow c_8$$

$$d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{\neg d_3^q \wedge \neg d_4^q}} \rightarrow c_9 \qquad d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{\neg d_3^q \wedge d_4^q}} \rightarrow c_{10}$$

$$d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{d_3^q \wedge \neg d_4^q}} \rightarrow c_{11} \qquad d_1^q \wedge \boxed{\neg d_2^q \wedge \boxed{d_3^q \wedge d_4^q}} \rightarrow c_{12}$$

$$d_1^q \wedge \boxed{d_2^q \wedge \boxed{\neg d_3^q \wedge \neg d_4^q}} \rightarrow c_{13} \qquad d_1^q \wedge \boxed{d_2^q \wedge \boxed{\neg d_3^q \wedge d_4^q}} \rightarrow c_{14}$$

$$d_1^q \wedge \boxed{d_2^q \wedge \boxed{d_3^q \wedge \neg d_4^q}} \rightarrow c_{15} \qquad d_1^q \wedge \boxed{d_2^q \wedge \boxed{d_3^q \wedge d_4^q}} \rightarrow c_{16}$$

Figure 3.4: All duplication in the constraints for classification of depth $H = 4$.

Simply from Figure 3.4, we observe the duplication of conjunctions. For example, the only difference between the implications for c_1 and c_9 ($c_{2^{H-1}+1}$) is the negation for the variable d_1^q. To avoid the duplication in the conjunction, for the common case of depth H, we consider the most basic condition of all the conjunctions of the last 2 literals, which are listed as follow:

$$\neg d_{H-1}^q \wedge \neg d_H^q, \quad \neg d_{H-1}^q \wedge d_H^q, \quad d_{H-1}^q \wedge \neg d_H^q, \quad d_{H-1}^q \wedge d_H^q$$

To catch this information and make the propagation easier, we introduce a new propositional Boolean variable b_{ij}^q to replace the value of the j-th conjunction in order of binary digits of the last i literals for example e_q ($[d_H^q, \ldots, d_{H-i+1}^q]$). The definition of variable b_{ij}^q is shown in Table 3.6.

We propose constraints to relate the variables b_{ij}^q and the conjunctions of d_i^q. At first, we link the 4 most basic conjunctions in size of 2 as we mentioned before.

	Description of The Variables
b_{ij}^q	**1** iff for example e_q, the value of the j-th conjunction of the last i literals is 1, the conjunction order is same as the order of binary digits with $i = 2, \ldots, H$, $j = 1, \ldots, 2^i$, $q = 1, \ldots, M$. $\forall i \in \{2, H\}$, $\forall j \in \{1, 2^i\}$, $\forall q \in \{1, M\}$

Table 3.6: Description of new Boolean variables for the advanced SAT encoding for binary decision diagrams.

For each example $e_q \in \mathcal{E}$:

$$\begin{aligned} \neg d_{H-1}^q \wedge \neg d_H^q \leftrightarrow b_{21}^q, \quad & \neg d_{H-1}^q \wedge d_H^q \leftrightarrow b_{22}^q, \\ d_{H-1}^q \wedge \neg d_H^q \leftrightarrow b_{23}^q, \quad & d_{H-1}^q \wedge d_H^q \leftrightarrow b_{24}^q \end{aligned} \tag{3.10}$$

Next, we consider the general case for the conjunctions of several literals. Assuming $H \geq 3$, for each example $e_q \in \mathcal{E}$, $i = 2, \ldots, H-1$, $j = 1 \ldots, 2^i$, we have the following constraints:

$$\begin{aligned} \neg d_{H-i}^q \wedge b_{ij}^q \leftrightarrow b_{(i+1)j}^q \\ d_{H-i}^q \wedge b_{ij}^q \leftrightarrow b_{(i+1)(j+2^i)}^q \end{aligned} \tag{3.11}$$

With the usage of variables b_{ij}^q, we can simply rewrite the original constraints for classification of Figure 3.4 for each positive example e_q as follow:

$$\begin{aligned} b_{H1}^q \rightarrow c_1, \quad b_{H2}^q \rightarrow c_2, \\ \ldots \\ b_{H(2^H-1)}^q \rightarrow c_{2^H-1}, \quad b_{H2^H}^q \rightarrow c_{2^H} \end{aligned} \tag{3.12}$$

For negative examples, the idea of transformation is the same.

Example 12 *Consider encoding a binary decision diagram with depth $H = 3$ by the improved encoding. We can list the constraints for classification of a general example e_q (assuming e_q is positive), based on Constraint 3.8:*

$$\begin{aligned} \neg d_1^q \wedge \neg d_2^q \wedge \neg d_3^q \rightarrow c_1, \quad & \neg d_1^q \wedge \neg d_2^q \wedge d_3^q \rightarrow c_2, \\ \neg d_1^q \wedge d_2^q \wedge \neg d_3^q \rightarrow c_3, \quad & \neg d_1^q \wedge d_2^q \wedge d_3^q \rightarrow c_4, \\ d_1^q \wedge \neg d_2^q \wedge \neg d_3^q \rightarrow c_5, \quad & d_1^q \wedge \neg d_2^q \wedge d_3^q \rightarrow c_6, \\ d_1^q \wedge d_2^q \wedge \neg d_3^q \rightarrow c_7, \quad & d_1^q \wedge d_2^q \wedge d_3^q \rightarrow c_8. \end{aligned}$$

Therefore, we have eight 4-SAT clauses. For this new encoding, the Constraints 3.10 and 3.11 are:

$$\begin{aligned} \neg d_2^q \wedge \neg d_3^q \leftrightarrow b_{21}^q, \quad & \neg d_2^q \wedge d_3^q \leftrightarrow b_{22}^q, \quad & d_2^q \wedge \neg d_3^q \leftrightarrow b_{23}^q, \quad & d_2^q \wedge d_3^q \leftrightarrow b_{24}^q, \\ \neg d_1^q \wedge b_{21}^q \leftrightarrow b_{31}^q, \quad & \neg d_1^q \wedge b_{22}^q \leftrightarrow b_{32}^q, \quad & \neg d_1^q \wedge b_{23}^q \leftrightarrow b_{33}^q, \quad & \neg d_1^q \wedge b_{24}^q \leftrightarrow b_{34}^q, \\ d_1^q \wedge b_{21}^q \leftrightarrow b_{35}^q, \quad & d_1^q \wedge b_{22}^q \leftrightarrow b_{36}^q, \quad & d_1^q \wedge b_{23}^q \leftrightarrow b_{37}^q, \quad & d_1^q \wedge b_{24}^q \leftrightarrow b_{38}^q, \end{aligned}$$

Moreover, the original eight 4-SAT clauses are rewritten as the following:

$$b^q_{31} \rightarrow c_1, \quad b^q_{32} \rightarrow c_2, \quad b^q_{33} \rightarrow c_3, \quad b^q_{34} \rightarrow c_4,$$
$$b^q_{35} \rightarrow c_5, \quad b^q_{36} \rightarrow c_6, \quad b^q_{37} \rightarrow c_7, \quad b^q_{38} \rightarrow c_8$$

We refer to this new SAT encoding as BDD3. The complexity of the encoding size (the number of literals) of BDD3 is in $\boldsymbol{O(M \times H \times (2^H + K))}$, that is the same complexity as the SAT model BDD2. However, for BDD3, we successfully transform all constraints into clauses of *3-SAT* or *2-SAT*, which could possibly do better in the propagation when H grows bigger. However, we could not guarantee the improvement in execution time. Therefore, we propose to compare BDD1, BDD2 and BDD3 with an experimental study in Section **??**.

3.3.4 MaxSAT Transformation

In previous subsections, we presented three SAT encodings to solve the decision problem $P_{bdd}(\mathcal{E}, H)$. We now present the MaxSAT transformation for the target optimisation problem $P^*_{bdd}(\mathcal{E}, H)$ described in Section 3.1. That is, given a set of examples $\mathcal{E}$, find a binary decision diagram of depth H that maximises the number of examples correctly classified.

The technique to transform the SAT encoding of binary decision diagrams into a MaxSAT encoding is quite simple. The key is to keep structural constraints as ***hard clauses***, and classification constraints as ***soft clauses***. All the SAT encodings that we proposed can be transformed, but we consider BDD2 and BDD3 as they have a lighter encoding size. In detail, as constraints for selecting features, Constraints 3.1, 3.2 and 3.3 are kept as hard clauses. For BDD2, Constraint 3.7 is kept as hard clause. Then, to classify the examples, we declare all clauses of Constraints 3.8 and 3.9 as soft clauses. The reason is that for any example e_q, the number of satisfied soft clauses associated to e_q is either $\mathbf{2^H}$, which indicates e_q is correctly classified, or $\mathbf{2^H - 1}$, which indicates e_q is wrongly classified. Therefore, the objective of maximising the number of satisfied soft clauses is equivalent to maximising the number of examples that are correctly classified. For BDD3, the difference is to keep Constraints 3.10 and 3.11 as hard clauses, then declare all clauses of Constraints 3.12 as soft clauses.

To simplify the notation, we refer to the MaxSAT encoding based on BDD2 as MaxSAT-BDD2, and the MaxSAT encoding based on BDD3 as MaxSAT-BDD3.

3.3.5 Merging Compatible Subtrees

Considering a binary decision diagram found by a MaxSAT solver associated to the truth table β. Based on the feature ordering, there might exist some values in β that capture no example, which is equivalent to the "*unknown*" nodes for the OODG method presented in Section 1.3.2. In fact, such values are decided by the MaxSAT solver in an arbitrary way, giving a certain bias for unseen examples. Inspired by the process of merging *compatible* subtrees applied to the OODG method, we

propose a post-processing procedure to merge the compatible subtrees for the binary decision diagrams found via the MaxSAT approach. This will result in changing the arbitrary values decided by MaxSAT solver in the truth table β.

The post-processing contains the three following phases, where phase 2 and phase 3 are adaptations of Algorithm 7:

1. Update the truth table β by replacing the values of β that capture no example with a special value "u".
2. From top to the bottom, for each level, check the existence of the beads, where "u" can be used to match 1 or 0, and create a node for each bead.
3. Then, for each level, after creating the nodes, we check the matches between all subtables of the next level. For matched ones, update the corresponding beads of current level to eliminate the "u" values.

Example 13 *Assume that the truth table β found via a MaxSAT model is* 00010111*, and the feature ordering found is $[f_1, f_2, f_3]$. We apply the first phase, and assume that the updated truth table $\beta^{'}$ is $u0u1011u$. Then, as $\beta^{'}$ is a bead, we create a root node at the level 1. Next, we check the subtables of $\beta^{'}$ ($u0u1$ and $011u$), and we move to the next level as they do not match. For level 2, we create a node for $u0u1$ and a node for $011u$ as they are all beads. Then, phase 3 is repeated. That is, we check all subtables of the next level, which are $\{u0, u1, 01, 1u\}$. We observe several matches: $u0$ and $1u$, $u1$ and 01. Then, the original beads $u0u1$ and $011u$ are updated as 1001 and 0110 Therefore, the updated beads of $\beta^{'}$ are $\{u0u1011u, 1001(u0u1), 0110(011u), 10, 01, 0, 1\}$.*

Figure 3.5 shows the binary decision diagram built by the truth table β that is found via a given MaxSAT model. To make the comparison, the left part of Figure 3.6 shows how the truth table $\beta^{'}$ is updated (with unknown values), and the right one shows the binary decision diagram after the merging post-processing.

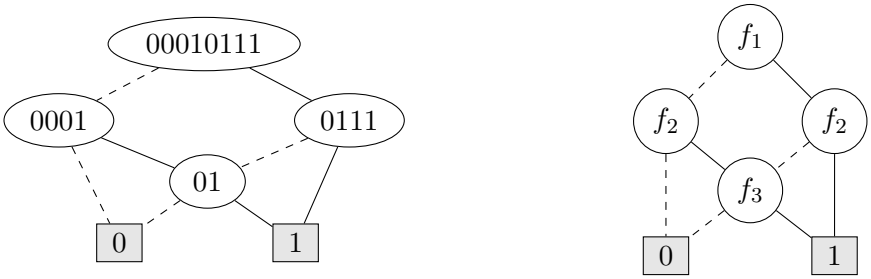


Figure 3.5: A binary decision diagram found by a MaxSAT solver.

3.4 Experimental Results

In this section, we present our large experimental studies to evaluate our propositions on different levels. The source code (developed in Python) and datasets are

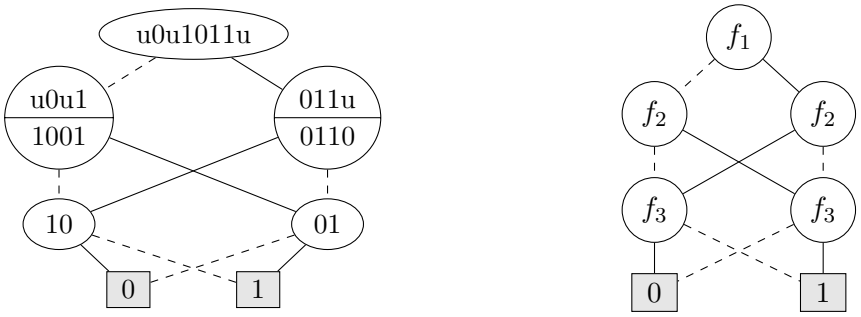


Figure 3.6: The binary decision diagram after merging compatible subtrees. The left one shows how the truth table β' is updated.

available online at https://gitlab.laas.fr/hhu/bddencoding. The outline of the experiments contains three parts. At first, we make preliminary experiments on the three proposed SAT models (BDD1, BDD2, and BDD3), to confirm the theoretical study of the reduction of the encoding size of BDD2 and BDD3 compared to BDD1. Moreover, we underline the difference between BDD2 and BDD3. Secondly, we evaluate the prediction performance between the proposed MaxSAT-BDD models (MaxSAT-BDD2 and MaxSAT-BDD3) and the heuristic methods (ODT and OODG from [Kohavi & Li 1995]). In the third experiment, we compare our best MaxSAT-BDD model with the exact method of learning optimal decision trees via MaxSAT (that we presented in Chapter 2), in terms of prediction quality, model and encoding size.

Same as the experiments made in Section 2.4, we also consider datasets from CP4IM[1]. These datasets are binarized by the classical one-hot encoding. In Table 3.7, we give the characteristics of these datasets. In detail, the column $\boldsymbol{M}$ indicates the number of examples in the dataset, the column $\boldsymbol{K_{orig}}$ indicates the original number of features, the column $\boldsymbol{K}$ indicates the number of binary features after binarization, and the column ***pos*** indicates the percentage of positive examples in the dataset.

All experiments are run on a cluster using Xeon E5-2695 v3@2.30GHz CPU with xUbuntu 16.04.6LTS.

3.4.1 Comparison of Different SAT Encodings

The first experiment aims to compare the encoding size between different SAT encodings (BDD1, BDD2, and BDD3) for the binary decision diagrams. We consider to find an optimal binary decision diagram having the perfect accuracy in the training set with the minimum depth. We use a simple linear search by solving multiple times the decision problem $P_{bdd}(\mathcal{E}, H)$, which decreases the depth when a binary decision diagram of depth H exists, and increases the depth in reverse. We set the initial depth $H_0 = 7$.

[1] https://dtai.cs.kuleuven.be/CP4IM/datasets/

Dataset	M	K_{orig}	K	pos
anneal	812	42	89	0.77
audiology	216	67	146	0.26
australian	653	51	124	0.55
cancer	683	9	89	0.35
car	1728	6	21	0.30
cleveland	296	45	95	0.54
hypothyroid	3247	43	86	0.91
kr-vs-kp	3196	36	73	0.52
lymph	148	27	68	0.55
mushroom	8124	21	112	0.52
tumor	336	15	31	0.24
soybean	630	16	50	0.15
splice-1	3190	60	287	0.52
tic-tac-toe	958	9	27	0.65
vote	435	16	48	0.61

Table 3.7: Detailed information of datasets from CP4IM used in experiments.

Considering the scalability problem, for each dataset, we use the hold-out method to split the training and testing set. We choose 5 different small splitting ratios $r = \{0.05, 0.1, 0.15, 0.2, 0.25\}$ to generate the training set. The remaining examples are used as the testing set. This process is repeated 10 times with different random seeds to avoid the influence of random seeds. The SAT solver we used is Kissat [Biere *et al.* 2020], which is the winner of SAT competition 2020. For each training process, we set 20 hours as the global timeout for the SAT solver.

Table 3.8 reports the average results of all instances that are solved to optimality by at least one encoding within the given global limited time. In detail, for the training process (with perfect accuracy), the column "**dopt**" indicate the average optimal depth. The encoding size (the number of literals) is given in column "**E_Size**", where the values are in thousands (10^3). The column "**Time**" indicates the run-time in seconds of successful runs. In addition, the column "**Acc**" indicates the average testing accuracy of the 5 random seeds in percent. The value "**N/A**" indicates the lack of results cause the timeout. Table 3.8 only reports results of instances solved by at least one encoding. However, some instances can not be solved to optimality even by the most efficient models (for example, dataset "*anneal*" for other ratios except 0.05). The last row summarizes briefly the average results of each encoding for all instances solved. To improve the readability of the table, the best values between the three SAT encodings are marked in blue.

From Table 3.8, we observe the great improvements in terms of encoding size and run-time of `BDD2` and `BDD3` compared to `BDD1`. From the average value, we observe that the encoding size of `BDD1` is more than 12 times smaller than the encoding size of `BDD2`, and more than 9 times smaller than the one of `BDD3`. The run-time of `BDD1` is almost 5 times larger than the run-time of `BDD2`, and 4 times for `BDD3`. Theses results confirm the massive reduction as the theoretical analysis

Datasets	r	BDD1				BDD2				BDD3			
		Acc	dopt	E_Size	Time	Acc	dopt	E_Size	Time	Acc	dopt	E_Size	Time
anneal	0.05	N/A	N/A	N/A	N/A	68.65	6.3	94.34	192.74	67.44	6.3	128.99	191.85
audilogy	0.05	75.99	2.0	8.25	0.46	76.86	2.0	8.32	0.07	76.86	2.0	8.32	0.07
	0.1	91.22	2.5	28.58	0.84	90.97	2.5	20.14	0.07	90.41	2.5	20.82	0.08
	0.15	92.54	2.8	53.68	1.39	93.3	2.8	32.6	0.09	93.08	2.8	34.28	0.12
	0.2	90.46	3.1	115.39	4.65	90.0	3.1	46.96	0.17	89.89	3.1	50.3	0.23
	0.25	92.94	3.6	235.28	31.79	92.45	3.6	68.14	0.39	92.52	3.6	74.75	0.54
australian	0.05	80.21	3.3	85.36	21.65	79.02	3.3	33.58	0.54	79.94	3.3	36.5	0.58
	0.1	N/A	N/A	N/A	N/A	77.46	6.2	152.12	7473.32	75.35	6.2	192.69	6358.56
cancer	0.05	86.97	2.7	35.52	0.49	86.69	2.7	20.37	0.08	87.0	2.7	21.98	0.1
	0.1	89.97	4.0	231.55	4.95	89.24	4.0	60.08	0.55	90.54	4.0	70.85	1.09
	0.15	90.7	5.2	1048.1	156.32	90.29	5.2	129.32	3.72	90.03	5.2	166.25	6.62
	0.2	91.53	6.4	3907.14	10224.45	91.57	6.4	264.95	55.2	91.88	6.4	368.31	53.17
	0.25	N/A	N/A	N/A	N/A	92.16	6.4	351.21	200.12	92.03	6.4	491.52	203.52
car	0.05	N/A	N/A	N/A	N/A	80.18	7.6	194.25	1924.9	78.67	7.6	320.01	1840.89
cleveland	0.05	68.19	2.5	14.06	0.86	64.72	2.5	10.3	0.07	65.25	2.5	10.77	0.07
	0.1	68.58	3.8	91.48	121.11	69.29	3.8	29.36	0.92	69.03	3.8	33.37	1.01
	0.15	72.53	4.8	276.5	800.35	70.83	4.8	56.3	15.07	71.71	4.8	68.38	14.21
	0.2	N/A	N/A	N/A	N/A	68.78	6.1	108.99	2616.23	68.86	6.1	143.96	2547.35
	0.25	N/A	N/A	N/A	N/A	68.74	6.9	181.28	16405.46	67.61	6.9	251.67	22367.9
hypothyrold	0.05	96.26	5.0	1318.04	319.09	96.3	5.0	182.24	2.98	96.56	5.0	233.26	3.04
lymph	0.05	67.23	2.0	3.33	0.13	68.79	2.0	3.39	0.07	68.79	2.0	3.39	0.07
	0.1	67.69	2.6	11.08	0.46	70.37	2.6	7.9	0.07	70.75	2.6	8.47	0.07
	0.15	70.16	3.4	34.11	2.83	71.98	3.4	15.24	0.16	71.83	3.4	17.44	0.29
	0.2	70.34	3.9	69.76	21.77	72.94	3.9	22.36	0.62	71.51	3.9	26.6	0.59
	0.25	72.23	5.1	230.83	410.1	68.48	5.1	39.11	4.31	69.82	5.1	50.54	5.7
mushroom	0.05	99.55	5.3	5180.35	3774.19	99.48	5.3	600.28	28.32	99.53	5.3	749.65	42.64
	0.1	99.81	5.8	15236.92	12552.47	99.87	5.8	1387.26	116.07	99.82	5.8	1790.16	168.06
	0.15	99.87	5.9	24750.11	20616.09	99.86	5.9	2136.23	210.98	99.85	5.9	2772.52	334.37
	0.2	99.97	6.0	35038.71	29342.71	99.92	6.0	2922.49	278.23	99.94	6.0	3812.44	573.5
	0.25	99.96	6.0	43816.89	33787.52	99.95	6.0	3651.83	437.4	99.95	6.0	4764.82	669.58
soybean	0.05	80.52	3.9	57.35	2.94	78.58	3.9	17.79	0.21	79.7	3.9	22.37	0.26
	0.1	84.47	5.9	708.43	1391.79	82.22	5.9	74.81	7.46	81.64	5.9	111.37	8.47
tic-tac-toe	0.05	64.46	5.9	235.73	41.18	66.37	5.9	38.27	5.62	66.51	5.9	62.82	6.25
	0.1	72.68	7.6	2061.03	18589.3	74.61	7.6	215.21	2846.24	72.49	7.6	353.61	2690.43
vote	0.05	90.89	2.1	6.07	0.27	91.09	2.1	5.67	0.08	91.09	2.1	5.81	0.08
	0.1	91.93	2.6	19.87	0.57	91.63	2.6	13.35	0.08	91.76	2.6	15.07	0.1
	0.15	92.27	3.4	69.67	1.2	92.27	3.4	27.5	0.16	92.57	3.4	34.07	0.27
	0.2	92.35	4.1	206.77	26.43	92.44	4.1	47.57	0.71	92.29	4.1	63.6	1.1
	0.25	93.0	4.9	539.23	348.56	92.42	4.9	81.31	3.4	91.96	4.9	115.79	4.53
Average	–	84.77	4.18	4112.88	4018.15	83.35	4.55	342.37	841.87	83.24	4.55	448.91	976.86

Table 3.8: Evaluation of different SAT encodings (BDD1, BDD2, and BDD3) for finding binary decision diagram of perfect accuracy with the smallest depth.

made before. Meanwhile, compared to BDD2, we observe that BDD3 does not make a breakthrough in reducing the run-time as the trade-off between the searching and propagation. In addition, the optimal depths found between these SAT encodings (except BDD1 as there are some N/A values) are identical, but there are some slight differences in the testing accuracy. This fact is explained by the different structures of the binary decision diagrams with the optimal depths.

3.4.2 Comparison with Existing Heuristic Approaches

In the second experiment, we consider solving the optimisation problem $P^*_{bdd}(\mathcal{E}, H)$ by the MaxSAT-BDD approaches, where the prediction performance is the major metric to be evaluated.

The structure of this experiment contains two parts. The first part is to compare the differences between MaxSAT-BDD2 and MaxSAT-BDD3 in terms of prediction performance and the effectiveness of reporting optimality, which is an extension of the previous experiment. The second part is the core of this experiment, which is to evaluate the prediction performance between the best MaxSAT-BDD approach and existing heuristic methods (ODT and OODG from [Kohavi & Li 1995]). As described in Section 1.3.2, after merging the isomorphic and compatible subtrees of ODT, the OODG method changes the bias for those "*unknown*" nodes, which makes no difference in the training examples but affects the prediction for unseen examples. Similarly, in Section 3.3.5, we described the post-processing of merging compatible subtrees for the binary decision diagrams found via the MaxSAT approach. Therefore, in this experiment, we consider the following three different biases for the MaxSAT-BDD deciding those "unknown" nodes:

- By assigning for each unknown node the majority class of examples in its parent (denoted as **MaxSAT-BDD-P**).
- By merging compatible subtrees (**MaxSAT-BDD-C**).
- By using the class decided by the MaxSAT solver (**MaxSAT-BDD-S**).

In these experiments, we face lighter scalablity problem than the previous one, and, for each dataset, we use the classical random 5-fold cross-validation with 5 different seeds. The MaxSAT solver we used is Loandra [Berg *et al.* 2019], which is an efficient incomplete MaxSAT solver. For each experiment of MaxSAT-BDD, the time limit for generating formulas is set to 15 minutes and the time limit for the solver is also set to 15 minutes.

For the fist part of these experiments (comparison between MaxSAT-BDD2 and MaxSAT-BDD3), we consider $H \in \{2, 3, 4, 5, 6, 7, 8\}$, as a wide range of depths could be better to perceive the differences. For the second part of these experiments (comparison between the best MaxSAT-BDD and heuristic methods), the candidate depths are restricted to $H \in \{2, 3, 4, 5, 6\}$ concerning the scalability.

Table 3.9 and 3.10 report the average results of all instances for the MaxSAT-BDD2 and MaxSAT-BDD3 approaches. Each line presents average values over 25 runs (5-folds with 5 random seeds). In detail, the column "**Opt**" indicates the percentage of instances that report optimality, where 0% indicates that all runs reach the time-out condition. The column "**U**" indicates the percentage of "unknowned" values in the truth table found, the column "**Train**" indicates the average training accuracy, the column "**Test-S**" indicates the average testing accuracy under the bias of MaxSAT-BDD-S, the column "**Test-P**" indicates the average testing accuracy under the bias of MaxSAT-BDD-P, the column "**Test-C**" indicates the average

Datasets	d	MaxSAT-BDD2									MaxSAT-BDD3								
		Opt	U	Train	Test-S	Test-P	Test-C	E_Size	Time_F	Time_S	Opt	U	Train	Test-S	Test-P	Test-C	E_Size	Time_F	Time_S
anneal	2	100	25	82.92	82.19	82.19	82.19	24.09	1.38	92.93	100	25	82.92	82.24	82.24	82.24	24.09	1.06	94.67
	3	0	49	84	83.55	83.55	83.55	37.21	2.29	TO	0	49	83.97	83.55	83.55	83.55	41.63	1.84	TO
	4	0	58.75	84.58	83.82	83.82	83.84	52.06	3.31	TO	0	58.75	84.64	84.04	84.01	84.01	61.68	2.66	TO
	5	0	64.38	85.33	83.87	83.87	83.92	71.08	4.25	TO	0	64.62	85.20	83.84	83.89	83.89	90.05	3.76	TO
	6	0	73.44	86.26	83.67	83.60	83.70	99.47	5.63	TO	0	74.44	86	83.74	83.77	83.77	135.07	5.53	TO
	7	0	82.69	86.55	83.77	83.74	83.72	148.69	6.09	TO	0	85.12	86.21	84.66	84.56	84.63	213.39	8.38	TO
	8	0	88.12	86.58	83.55	83.64	83.60	243.71	9.06	TO	0	87.83	86.16	83.60	83.60	83.69	358.30	13.48	TO
audiology	2	100	0	94.91	94.92	94.92	94.92	10.59	0.68	0.46	100	0	94.91	94.92	94.92	94.92	10.59	0.47	0.35
	3	100	0.50	96.78	95.84	95.84	95.84	16.41	1.06	6.63	100	1	96.78	95.74	95.93	95.93	17.59	0.76	6.99
	4	100	25.25	97.73	95.28	95.65	95.56	22.56	1.45	56.31	100	25	97.74	95.84	95.84	95.84	25.12	1.11	89.93
	5	72	48.50	98.4	95	95.27	94.44	29.82	1.90	578.99	54	50.78	98.38	95.14	95.25	95.14	34.87	1.49	636.08
	6	48	59.88	99.17	95.65	95.74	95.84	39.59	2.42	613.06	44	60.94	99.17	95.47	95.38	95.19	49.06	2.09	751.94
	7	48	76.84	99.7	94.36	94.45	94.73	54.93	2.34	467.22	48	75.53	99.7	94.63	94.90	95.09	72.14	2.96	468.41
	8	100	85.09	100	93.43	94.64	94.91	82.51	3.29	1.37	100	84.97	100	94.07	94.91	95.38	112.99	4.51	5.05
australian	2	100	18	86.70	85.94	85.94	85.94	26.79	1.67	167.99	100	22.50	86.71	86.41	86.41	86.41	26.79	1.24	147.66
	3	0	0	87.45	84.81	84.81	84.81	41.15	2.53	TO	0	1	87.45	84.96	84.96	84.96	44.71	1.91	TO
	4	0	7.50	88.45	86.03	86.03	86.03	56.85	3.45	TO	0	10.50	88.51	85.33	85.33	85.33	64.58	2.82	TO
	5	0	22.88	89.36	85.85	85.91	85.91	75.9	4.61	TO	0	18.62	89.29	85.09	85.12	85.12	91.15	3.93	TO
	6	0	33.12	90.05	85.67	85.67	85.7	102.49	5.98	TO	0	33.19	89.80	85.21	85.21	85.24	131.12	5.36	TO
	7	0	43.03	90.74	83.86	84.11	84.11	145.83	6.04	TO	0	47.94	90.26	84.47	84.26	84.62	197.86	8.01	TO
	8	0	53.07	91.49	82.66	82.43	82.77	226.03	8.64	TO	0	60.86	90.80	83.98	83.12	84.32	318.18	12.29	TO
cancer	2	100	0	93.88	93.59	93.59	93.59	20.29	1.25	5.89	100	0	93.88	93.63	93.63	93.63	20.29	0.95	6.23
	3	100	19	95.02	93.91	93.91	93.91	31.37	1.89	525.59	76	20	95.02	93.65	93.65	93.65	35.09	1.55	657.85
	4	0	25.50	96.06	95.49	95.49	95.49	43.89	2.76	TO	0	26.88	96.05	95.21	95.21	95.21	51.98	2.25	TO
	5	0	42	95.94	93.65	93.79	93.74	59.91	3.57	TO	0	43.25	95.94	94.47	94.5	94.5	75.87	3.21	TO
	6	0	55.56	96.84	94.35	94.29	94.35	83.83	4.87	TO	0	56.19	96.36	93.71	93.65	93.53	113.77	4.74	TO
	7	0	71.41	97.15	93.94	94.23	94.12	125.26	5.02	TO	0	72.22	96.66	93.56	93.85	93.85	179.68	6.98	TO
	8	0	79.55	97.36	92.95	93.12	93.15	205.24	7.69	TO	0	82.66	97.46	94.23	94.32	94.20	301.62	11.35	TO
car	2	100	0	85.53	85.53	85.53	85.53	13.32	0.79	24.82	100	0	85.53	85.53	85.53	85.53	13.32	0.61	20.52
	3	8	1	88.4	87.41	87.41	87.41	21.95	1.36	TO	4	1	88.35	87.59	87.59	87.59	31.35	1.3	TO
	4	0	3	89.84	88.54	88.54	88.54	34.44	1.94	TO	0	10	89.61	88.45	88.45	88.45	54.90	2.15	TO
	5	0	25	91.13	89.91	89.91	89.91	55.79	3.04	TO	0	25	91.22	90.01	90.01	90.01	96.15	3.73	TO
	6	0	39.25	93.51	92.99	92.99	92.99	97.06	5.06	TO	0	35.50	93.06	92.19	92.19	92.19	172.81	6.31	TO
	7	0	46.09	94.24	93.98	93.98	93.98	182.61	6.66	TO	0	46.88	93.96	93.52	93.52	93.52	320.29	11.45	TO
	8	0	56.45	91.41	89.62	89.64	89.64	365.55	12.48	TO	0	57.39	89.92	87.85	87.85	87.88	609.40	21.26	TO
cleveland	2	100	0	79.04	72.57	72.57	72.57	9.48	0.55	83.84	100	0	79.04	72.7	72.7	72.7	9.48	0.43	87.71
	3	0	0	85.07	83.37	83.37	83.37	14.73	0.87	TO	0	0	84.76	83.95	83.95	83.95	16.34	0.73	TO
	4	0	1.75	86.32	79.46	79.46	79.46	20.55	1.19	TO	0	2.50	86.33	79.73	79.66	79.73	24.05	1	TO
	5	0	3.25	88.65	78.72	78.72	78.72	27.89	1.57	TO	0	4.75	88.12	80.34	80.13	80.20	34.80	1.44	TO
	6	0	14.63	90.74	77.42	77.69	77.29	38.66	2.32	TO	0	18.56	90.66	76.62	76.41	76.48	51.63	2.22	TO
	7	0	34.12	93.4	75.41	75.95	76.02	57.04	2.3	TO	0	35.19	92.58	75.40	75.80	75.20	80.63	3.15	TO
	8	0	53.86	96.01	75.27	75.40	73.58	92.18	3.49	TO	0	54.75	95.73	75.34	75.61	74.73	133.95	5.26	TO
hypothyroid	2	100	0	97.84	97.84	97.84	97.84	92.65	5.65	77.76	100	0	97.84	97.84	97.84	97.84	92.65	4.2	91.94
	3	0	2	98.09	98.04	98.04	98.04	142.78	8.83	TO	0	1.88	98.09	98	98.01	98.01	160.44	7.19	TO
	4	0	13.75	98.27	98.14	98.14	98.13	200.09	12.55	TO	0	15.25	98.27	98.16	98.16	98.16	238.53	10	TO
	5	0	33.75	98.3	98.05	98.05	98.05	274.03	16.66	TO	0	34.63	98.28	98	98.01	98	349.88	14.54	TO
	6	0	46	98.37	97.92	97.95	97.95	385.4	23.03	TO	0	49.25	98.16	97.87	97.87	97.88	527.75	21.08	TO
	7	0	61.44	97.47	96.95	97.04	97.06	579.94	23.43	TO	0	66.12	96.50	96.13	96.21	96.24	838.66	31.91	TO
	8	0	72.95	95.36	94.81	94.99	94.94	957.42	35.5	TO	0	77.36	95.5	94.86	95	95.03	1415.63	52.72	TO

Table 3.9: Evaluation of MaxSAT-BDD2 and MaxSAT-BDD3 in terms of prediction performance and effectiveness of reporting optimality (Part 1).

Datasets	d	MaxSAT-BDD2									MaxSAT-BDD3								
		Opt	U	Train	Test-S	Test-P	Test-C	E_Size	Time_F	Time_S	Opt	U	Train	Test-S	Test-P	Test-C	E_Size	Time_F	Time_S
kr-vs-kp	2	0	0	77.83	77.01	77.01	77.01	77.88	4.72	TO	4	0	77.83	76.95	76.95	76.95	77.88	3.64	TO
	3	0	0	90.43	90.43	90.43	90.43	120.54	7.54	TO	0	0	90.43	90.43	90.43	90.43	137.92	6.11	TO
	4	0	18.75	94.09	94.09	94.09	94.09	170.28	10.20	TO	0	18.75	94.09	94.09	94.09	94.09	208.12	9	TO
	5	0	45.25	94.34	94.18	94.18	94.18	236.39	14.20	TO	0	45	94.29	94.24	94.24	94.24	311.05	12.59	TO
	6	0	53	92.80	92.55	92.54	92.55	339.35	19.59	TO	0	48	93.26	92.87	92.87	92.87	479.46	18.89	TO
	7	0	51.64	85.84	84.89	84.89	84.92	524.16	20.24	TO	0	53.62	88.8	88.38	88.39	88.4	778.81	29.14	TO
	8	0	68.80	86.78	86.33	86.36	86.38	889.04	32.56	TO	0	66.66	79.38	78.74	78.78	78.78	1340.06	49.04	TO
lymph	2	100	0	84.46	83.23	83.23	83.23	3.5	0.20	2.84	100	0	84.46	84.03	84.03	84.03	3.5	0.16	2.88
	3	32	4	86.76	78.35	78.35	78.35	5.55	0.40	829.42	28	5	86.76	78.88	78.88	78.88	6.36	0.26	858.12
	4	0	3	90.54	82.40	82.67	82.40	7.86	0.44	TO	0	4	90.54	82.81	82.81	82.67	9.61	0.41	TO
	5	0	11.37	93.51	83.21	83.88	83.60	10.94	0.63	TO	0	13	93.48	82.53	82.93	83.21	14.40	0.57	TO
	6	0	37.56	95.88	83.20	84.70	84.82	15.74	0.9	TO	0	35.81	95.74	83.35	83.10	84.70	22.23	0.9	TO
	7	0	56.78	97.80	79.58	80.79	82.56	24.37	0.93	TO	0	57.78	97.84	78.63	79.57	80.94	36.16	1.40	TO
	8	24	73.62	99.29	77.56	77.69	78.9	41.41	1.5	683.94	24	73.73	99.29	76.07	78.25	77.79	62.30	2.31	694.89
mushroom	2	100	0	95.13	95.13	95.13	95.13	299.19	18.83	425.74	100	0	95.13	95.13	95.13	95.13	299.19	13.84	434.89
	3	0	38	97.74	97.77	97.77	97.77	458.1	28.60	TO	0	41	97.94	97.93	97.93	97.93	502.30	21.93	TO
	4	0	49.75	98.78	98.74	98.74	98.74	635.09	40.63	TO	0	51	98.64	98.59	98.59	98.59	731.28	31.28	TO
	5	0	64.50	98.63	98.57	98.57	98.57	853.68	52.90	TO	0	63.50	96.62	96.48	96.48	96.48	1043.46	43.74	TO
	6	4	75.44	97.28	97.1	97.1	97.1	1165.88	75.77	855.34	8	74.75	92.72	92.89	92.89	92.89	1522.04	61.65	852.08
	7	20	84.84	93.25	93.26	93.26	93.26	1686.1	67.41	736.99	0	81.88	88.54	88.48	88.48	88.48	2333.42	91.08	TO
	8	44	89.16	89.62	89.7	89.7	89.7	2663.93	101.02	530.65	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
tumor	2	100	0	82.8	81.60	81.60	81.60	3.72	0.21	5.46	100	0	82.8	81.78	81.78	81.78	3.72	0.18	5.28
	3	0	1.25	83.84	80.43	80.43	80.43	6.02	0.35	TO	0	1	83.81	80.47	80.53	80.47	7.85	0.33	TO
	4	0	4.25	85.52	82.55	82.43	82.49	9.04	0.5	TO	0	3.75	85.52	82.91	82.97	82.97	13.02	0.52	TO
	5	0	23.75	87.51	86.06	85.77	85.83	13.79	0.8	TO	0	23.12	87.51	86.36	85.89	85.41	21.64	0.83	TO
	6	0	29.44	88.57	81.72	81.66	81.12	22.44	1.2	TO	0	31.31	88.53	81.60	82.02	81.66	37.17	1.43	TO
	7	0	50.22	90.16	81.96	81.60	82.02	39.72	1.46	TO	0	50.75	90.18	83.26	82.49	82.13	66.49	2.58	TO
	8	0	65.88	91.52	79.81	79.16	78.92	76	2.6	TO	0	66.33	91.46	80.35	80.65	80.58	123.42	4.55	TO
soybean	2	100	0	90.48	90.48	90.48	90.48	10.79	0.64	9.51	100	0	90.48	90.48	90.48	90.48	10.79	0.48	8.8
	3	68	17.50	91.39	90.41	90.41	90.41	16.99	1.02	706.39	54	16.88	91.38	90.40	90.40	90.40	20.41	0.88	755.15
	4	0	23.50	93.24	93.21	93.21	93.21	24.54	1.48	TO	0	24	93.24	93.02	93.02	93.02	32	1.33	TO
	5	0	44.75	94.31	92.95	93.02	92.95	35.34	2.04	TO	0	47	94.17	93.37	93.40	93.43	50.05	1.98	TO
	6	0	61.38	96.07	95.56	95.52	95.52	53.41	2.92	TO	0	61.37	96.06	95.52	95.56	95.49	81.03	3.15	TO
	7	0	73.50	96.69	95.49	95.46	95.56	87.65	3.41	TO	0	73.41	96.47	95.30	95.30	95.33	137.84	5.19	TO
	8	0	82	97.52	96.67	96.86	96.79	157.44	5.68	TO	0	81.45	97.14	96.10	96.06	96.25	246.35	9.05	TO
splice-1	2	0	0	84.04	84.04	84.04	84.04	296.61	18.69	TO	0	0	83.97	83.96	83.96	83.96	296.61	13.45	TO
	3	0	3	87.25	86.94	86.94	86.94	449.04	28.74	TO	0	5	87.3	86.95	86.95	86.95	466.39	21.23	TO
	4	0	14.25	88.3	88.04	88.04	88.04	608.3	39.67	TO	0	12	87.92	87.73	87.73	87.73	646.07	29.13	TO
	5	0	10.25	71.99	70.53	70.53	70.53	783.9	49.73	TO	0	3.62	66.94	66.29	66.29	66.29	858.42	37.78	TO
	6	0	26	62.92	61.88	61.9	61.89	996.27	64.65	TO	0	11.44	55.90	55.01	55.03	55.01	1136.12	48.06	TO
	7	0	44.97	60.82	59.96	60.01	60.03	1290.34	55.92	TO	0	46.66	54.45	52.92	52.93	52.92	1544.52	64.58	TO
	8	0	73.70	56.04	55.18	55.23	55.22	1764.15	72.1	TO	0	59.58	55.81	54.35	54.48	54.49	2214.33	87.79	TO
tic-tac-toe	2	100	0	71.05	68.35	68.35	68.35	9.25	0.56	48.01	100	0	71.05	68.35	68.35	68.35	9.25	0.41	45.67
	3	0	0	74.91	72.36	72.36	72.36	15.01	0.90	TO	0	0	74.86	72.19	72.19	72.19	20.22	0.81	TO
	4	0	5	76.87	74.22	74.22	74.22	22.88	1.34	TO	0	9	77.25	74.24	74.24	74.24	34.23	1.33	TO
	5	0	0.75	81.86	80.29	80.29	80.31	35.67	1.93	TO	0	1.62	82.17	79.29	79.33	79.27	58.05	2.21	TO
	6	0	5.06	84.82	80.19	80.17	80.08	59.52	2.99	TO	0	5.69	83.94	81.4	81.27	81.33	101.52	3.68	TO
	7	0	18.78	86.55	82.04	82.29	82.04	107.93	3.98	TO	0	26.59	82.28	78.14	78.41	78.35	184.26	6.67	TO
	8	0	40.92	86.83	81.86	82.23	82.36	210.37	7.19	TO	0	53.84	72.90	65.20	65.80	65.53	345.56	12.42	TO
vote	2	100	10	95.68	95.22	95.22	95.22	7.2	0.42	0.70	100	8	95.68	95.08	95.08	95.08	7.2	0.32	0.65
	3	100	2	96.69	94.62	94.57	94.57	11.38	0.65	94.11	100	0.50	96.69	94.62	94.62	94.62	13.75	0.57	138.77
	4	8	5.25	97.4	94.39	94.39	94.39	16.49	0.93	TO	4	4.50	97.38	94.16	94.16	94.21	21.64	0.87	TO
	5	0	7	98.21	94.48	94.62	94.57	23.83	1.37	TO	0	6.75	98.21	94.67	94.39	94.57	33.99	1.33	TO
	6	4	20.62	98.93	93.66	94.34	93.98	36.2	1.95	TO	0	22	98.93	94.44	94.44	94.11	55.27	2.11	TO
	7	20	42.03	99.46	92.64	93.47	93.15	59.75	2.22	776.54	8	40.81	99.44	92.05	93.38	93.38	94.41	3.56	TO
	8	28	59.28	99.75	91.22	92.87	93.24	107.88	3.79	647.85	28	58.41	99.75	92.23	92.55	93.15	169.26	6.36	653.15

Table 3.10: Evaluation of MaxSAT-BDD2 and MaxSAT-BDD3 in terms of prediction performance and effectiveness of reporting optimality (Part 2).

testing accuracy under the bias of MaxSAT-BDD-C. Then, the column "**E_Size**" indicates the encoding size (the number of literals) in thousands (10^3). Moreover, the time for generating formulas in seconds is given in column "**Time_F**", and the run-time used by the MaxSAT solver in seconds is given in column "**Time_S**". The value "**TO**" indicates the timeout. To improve the readability of the table, the best values of comparable metrics between MaxSAT-BDD2 and MaxSAT-BDD3 are marked in blue.

From Table 3.9 and 3.10, we can underline some observations. Firstly, although both MaxSAT-BDD approaches share the same complexity theoretically, MaxSAT-BDD2 produces formulas with lighter encoding sizes than MaxSAT-BDD3, and the differences grow bigger when the depth grows. The difference in the time of generating formulas also reflects this fact. Nevertheless, from the results in the percentage of reporting optimality and the run-time used by the MaxSAT solver, the increase of encoding size to reduce the propagation time used in MaxSAT-BDD3 approach does not show clear advantage in improving the percentage of finding optimal solutions and the prediction performance. Therefore, in the following experiments, we use only MaxSAT-BDD2 as it is the most effective MaxSAT-based approach, and we denote it directly as MaxSAT-BDD.

Another observation is there are slight differences (less than 5%) between the testing accuracy under different biases, although the percentage of "*unknown*" values is quite high for some cases. This observation suggests that the optimal solutions are somewhat robust to the different biases. In addition, we also notice that for all datasets (except one instance), MaxSAT-BDD report optimality when the depth is equal to 2.

We now present the results of the second part of this experiment, which is the comparison between MaxSAT-BDD2 (the best MaxSAT-BDD approach) and heuristic approaches in terms of prediction performance. Figure 3.7 shows the comparison of the average training accuracy between OODG and MaxSAT-BDD model. In this figure, different datasets are marked with different colors, and different depths are labelled with points of different sizes. From this scatter plot, we observe that at first the average training accuracy of both approaches improve with the increase of depth. Moreover, and more importantly, the MaxSAT-BDD model shows clear remarkable advantage in training accuracy than the heuristic OODG.

The Figure 3.8 shows the average testing accuracy of MaxSAT-BDD with different biases, ODT, and OODG using different depths averaged over all datasets. The white line and green triangle of each box indicate the median and the average values, respectively. Clearly, the MaxSAT-BDD models have better generalization performance than the heuristic methods ODT and OODG. This is particularly apparent with small depths. Meanwhile, increasing the depth improves the predictions for all methods as expected. However, this improvement in prediction performance of MaxSAT-BDD approach is not as significant as the improvement of ODT and OODG methods.

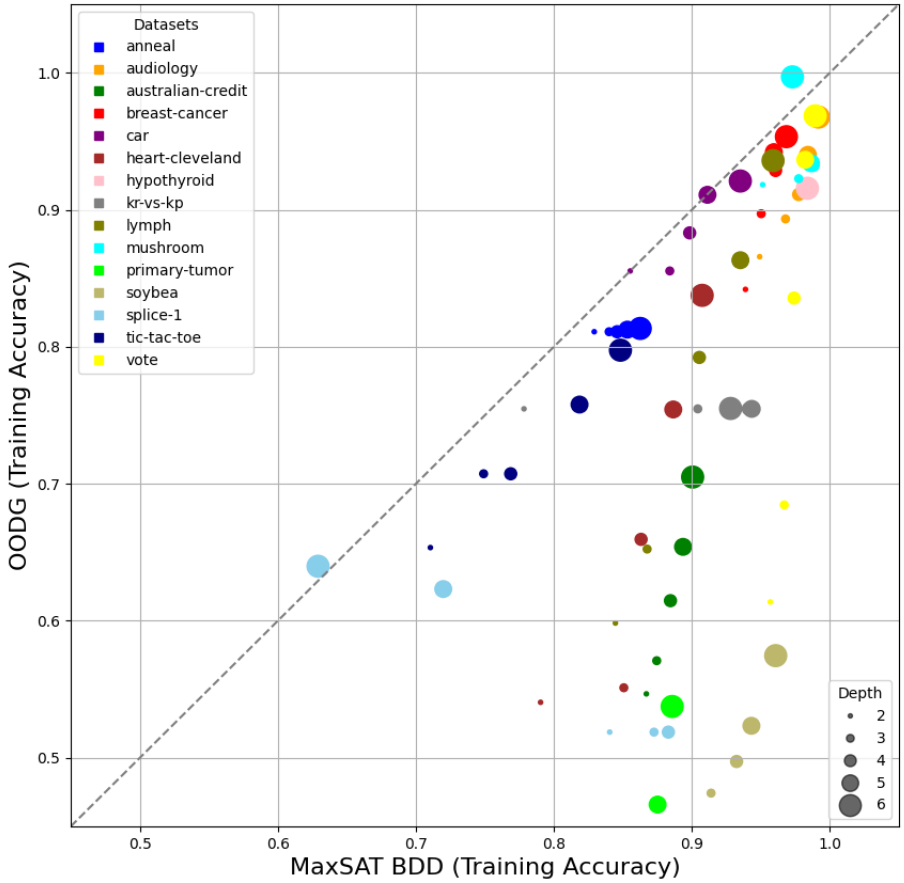


Figure 3.7: Comparison between the average training accuracy of OODG and MaxSAT-BDD.

3.4.3 Comparison with the Exact Decision Tree Approach

As the results of previous experiment comparing the MaxSAT-BDD with the heuristic methods shown, MaxSAT-BDD displayed its attraction in prediction performance. In this experiment, the purpose is to make the comparison between MaxSAT-BDD and the exact method learning optimal decision trees via MaxSAT (denoted as MaxSAT-DT, presented in Chapter 2), which we consider as the state-of-the-art method. Unlike the previous experiment applied several different biases for MaxSAT-BDD, we consider only the bias of merging compatible subtrees (MaxSAT-BDD-C) since no substantial difference was observed between different biases.

In detail, we follow the same settings of the previous experiment. That is, for each dataset, we use random 5-fold cross-validation with 5 different seeds. The candidate depths selected are $H \in \{2, 3, 4, 5, 6\}$, where each depth corresponds respectively to the number of different selected features for MaxSAT-BDD, and to the *maximum depth* of the decision tree for MaxSAT-DT. The incomplete MaxSAT solver used is Loandra, and for each experiment, the time limit for generating

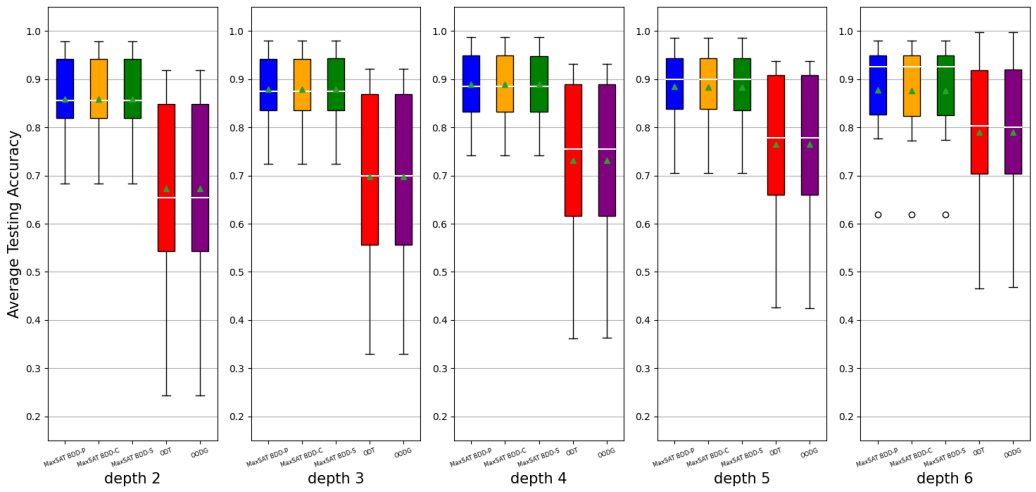


Figure 3.8: The average testing accuracy of methods with different biases in different depths: **MaxSAT-BDD-P**, **MaxSAT-BDD-C**, **MaxSAT-BDD-S**, **ODT**, **OODG** (respectively from left to right).

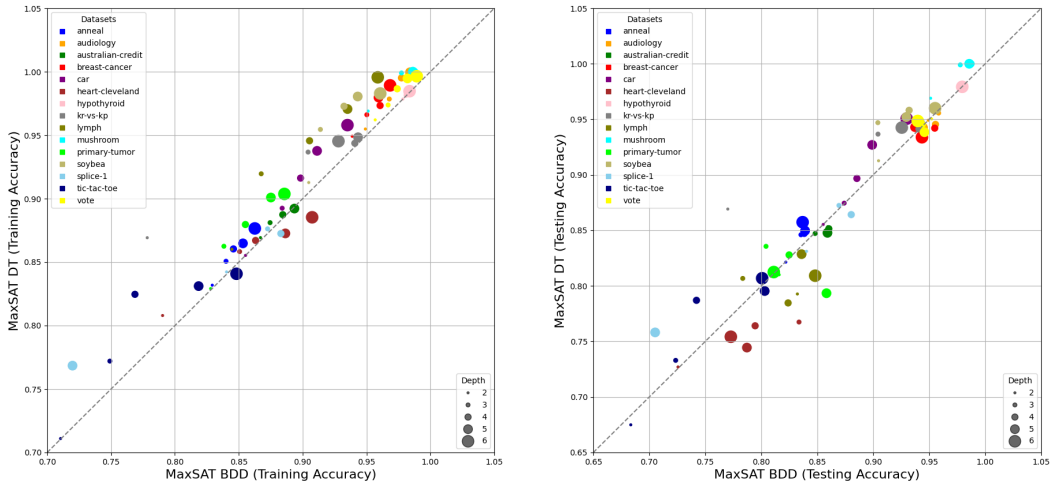


Figure 3.9: Comparison between the average prediction performance of MaxSAT-BDD and MaxSAT-DT, the left (right) scatter indicates the training accuracy (testing accuracy).

formulas and the time limit for the solver are both set to 15 minutes.

Based on results given in Tables 3.11, the Figure 3.9 illustrates the prediction performance between MaxSAT-BDD and MaxSAT-DT, where the left scatter shows the training accuracy, and the right scatter shows the testing accuracy. From this figure, we observe that MaxSAT-BDD is competitive to MaxSAT-DT in terms of prediction quality. In most cases, the training and testing accuracy of these two

Datasets	H	MaxSAT-BDD				MaxSAT-DT				
		Train	Test	Size	E_Size	Train	Test	Size	E_Size	F_d
anneal	2	82.92	82.19	5	24.09	83.18	82.14	6.84	52.72	2.88
	3	84	83.55	7	37.21	85.07	84.66	12.68	126.18	5.76
	4	84.58	83.84	9.4	52.06	86.05	84.78	18.68	315.45	8.64
	5	85.33	83.92	11.72	71.08	86.44	84.88	23.88	865.26	11.08
	6	86.26	83.70	14.68	99.47	87.6	85.76	39.16	2666.67	17.32
audiology	2	94.91	94.92	4	10.59	95.49	94.92	7	31.35	3
	3	96.78	95.84	5.04	16.41	97.82	95.56	11.56	88.75	5.28
	4	97.73	95.56	6.96	22.56	99.51	94.54	19.08	272.15	8.68
	5	98.40	94.44	9.88	29.82	99.95	93.98	27	915.29	11.72
	6	99.17	95.84	14.28	39.59	99.86	94.08	24.12	3323.61	10.88
australian	2	86.70	85.94	4.72	26.79	86.93	85.33	6.68	59.65	2.84
	3	87.45	84.81	5.32	41.15	88.09	84.87	13.08	146.15	5.68
	4	88.45	86.03	7.4	56.85	88.74	85.18	17.48	377.62	7.92
	5	89.36	85.91	10.44	75.9	89.28	84.75	22.52	1076.35	10.08
	6	90.05	85.7	17.32	102.49	89.49	84.84	27.08	3433.64	12.20
cancer	2	93.88	93.59	4	20.29	94.91	94.2	7	45.56	3
	3	95.02	93.91	5.84	31.37	96.6	94.73	15	110.85	6.96
	4	96.06	95.49	7.96	43.89	97.34	94.17	21	283.77	9.44
	5	95.94	93.74	10.68	59.91	97.99	94.35	29.32	800.89	13.20
	6	96.84	94.35	14.8	83.83	98.87	93.41	45.72	2536.91	19.88
car	2	85.53	85.53	4	13.32	85.53	85.53	6.84	32.01	2.92
	3	88.40	87.41	5.08	21.95	89.25	87.45	12.68	71.83	5.64
	4	89.84	88.54	6.84	34.44	91.62	89.68	20.36	162.46	7.68
	5	91.13	89.91	9.6	55.79	93.78	92.77	29.56	389.68	10.24
	6	93.51	92.99	13.36	97.06	95.8	95.06	31.96	1044.54	10.88
cleveland	2	79.04	72.57	4	9.48	80.76	72.84	7	25.57	3
	3	85.07	83.37	6	14.73	85.68	76.55	12.84	68.93	5.72
	4	86.32	79.46	7.84	20.55	86.77	76.75	17.80	200.76	8.04
	5	88.65	78.72	13.08	27.89	87.26	74.45	23.96	646.75	10.84
	6	90.74	77.29	21.04	38.66	88.58	75.81	28.84	2284.76	13.08
hypothyroid	2	97.84	97.84	4	92.65	97.84	97.84	5.96	182.20	2.48
	3	98.09	98.04	5.12	142.78	98.14	97.82	9.72	402.98	4.32
	4	98.27	98.13	6.72	200.09	98.38	98.01	15.40	885.51	7.12
	5	98.30	98.05	9.28	274.03	98.45	98	20.04	2016.31	8.92
	6	98.37	97.95	13.68	385.4	98.46	97.91	33.16	4957.57	14.04
kr-vs-kp	2	77.83	77.01	4	77.88	86.92	86.92	7	155.09	3
	3	90.43	90.43	5.28	120.54	93.81	93.79	12.44	342.99	5.08
	4	94.09	94.09	7.56	170.28	94.32	94.14	17.24	753.78	7.12
	5	94.34	94.18	9.52	236.39	94.85	94.69	25.40	1717.14	10.20
	6	92.80	92.55	11.52	339.35	93.91	93.69	29.32	4227.67	12.20
lymph	2	84.46	83.23	4	3.5	86.01	79.27	7	12.33	3
	3	86.76	78.35	5.92	5.55	91.93	80.54	14.68	36.65	6.64
	4	90.54	82.4	8.72	7.86	94.56	78.46	20.20	117.94	8.88
	5	93.51	83.6	13.52	10.94	97.09	82.46	27.08	413.09	11.88
	6	95.88	84.82	17.64	15.74	99.59	80.92	46.60	1550.34	18.96
mushroom	2	95.13	95.13	4	299.19	96.9	96.9	7	565.27	3
	3	97.74	97.77	6.8	458.1	99.9	99.9	13.72	1227.18	6.24
	4	98.78	98.74	9	635.09	100	100	19.80	2603.94	9.08
	5	98.63	98.57	11.32	853.68	100	100	23.40	5571.14	10.64
	6	97.28	97.10	14.6	1165.88	100	100	27.56	12376.90	12
tumor	2	82.80	81.6	4	3.72	82.92	81.01	6.76	10.46	2.88
	3	83.84	80.43	5.3	6.02	86.16	82.97	13.88	27.24	6.08
	4	85.52	82.49	8.64	9.04	87.89	82.85	20.92	76.40	9.16
	5	87.51	85.83	13.32	13.79	90.1	79.34	47.80	239.09	16.84
	6	88.57	81.12	19.84	22.44	90.34	81.31	37.32	838.63	15.04
soybean	2	90.48	90.48	4	10.79	91.27	91.27	7	25.55	3
	3	91.39	90.41	6.52	16.99	95.45	94.7	15	62.30	7
	4	93.24	93.21	9.04	24.54	97.25	95.9	22.20	160.18	9.88
	5	94.31	92.95	11.92	35.34	97.96	95.3	40.60	455.33	15.72
	6	96.07	95.52	14.88	53.41	98.27	96.03	33.40	1459.87	14.40
splice-1	2	84.04	84.04	4	296.61	84.22	83.17	6.92	555.22	2.96
	3	87.25	86.94	5.44	449.04	87.79	87.37	11.32	1231.59	4.64
	4	88.3	88.04	7.24	608.3	86.52	85.64	16.60	2717.90	7.12
	5	71.99	70.53	10.28	783.9	77.37	76.32	21.88	6226.75	9.48
	6	62.92	61.89	16.28	996.27	60.36	58.95	29.40	15406.05	12.28
tic-tac-toe	2	71.05	68.35	4	9.25	71.1	67.49	5.96	22.31	2.48
	3	74.91	72.36	6.16	15.01	77.15	73.55	11.48	51.98	5.20
	4	76.87	74.22	8.84	22.88	82.47	78.68	20.60	125.10	8.44
	5	81.86	80.31	13.88	35.67	83.08	79.50	28.44	328.33	11.16
	6	84.82	80.08	24.16	59.52	84.25	80.86	38.12	979.46	13.24
vote	2	95.68	95.22	3.76	7.2	96.21	95.03	7	18.33	3
	3	96.69	94.57	5.56	11.38	97.39	93.79	13.96	46.55	6.04
	4	97.40	94.39	8.16	16.49	98.62	94.57	21.16	126.45	9.32
	5	98.21	94.57	12.4	23.83	99.47	93.84	30.52	381.95	12.96
	6	98.93	93.98	18.44	36.2	99.62	94.76	35.40	1292.44	14.88

Table 3.11: Comparison of model size and encoding size between MaxSAT-BDD and MaxSAT-DT.

approaches are close.

In addition, Table 3.11 presents the complementary results of the evaluation. In detail, the column "**Size**" indicates the model size (number of nodes of the model), which reflects the interpretability of the model found. The column "**E_Size**" indicates the encoding size (the number of literals in thousands (10^3)). The column "**F_d**" indicates the average number of different features used in the decision tree. The best values are marked in blue.

From the results shown in Table 3.11, we observe that the binary decision diagrams found via MaxSAT-BDD always have smaller model size than the decision trees found via MaxSAT-DT. The same phenomenon is observed for the differences in encoding size. Moreover, such differences in model size and encoding size grow bigger when the depth increases. The reduction in model size provides better interpretability, while the lighter encoding size implies more chance to report optimality within same limited time.

We highlight a particular observation that for the dataset "*car*" and "*hypothyroid*" in depth 2, the binary decision diagrams found share the same ***optimal*** training accuracy with the decision trees found. However, meanwhile, for the cases mentioned, compared to the optimal binary decision diagrams found via MaxSAT-BDD, the optimal decision trees found via MaxSAT-DT make some useless splits, which is reflected by the average number of different features used in the decision trees. To illustrate this phenomenon, we present, in Figure 3.10, the binary decision diagram and the decision tree found for the dataset "*car*" of depth 2 . Both model share the same training accuracy, but the decision tree makes one more split (the split of "$f_{3,2}$") without increasing the prediction performance.

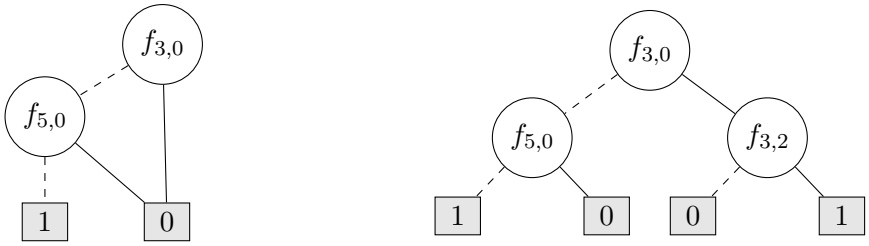


Figure 3.10: The illustrating example showing the useless split made by decision tree found via MaxSAT-DT.

3.5 Heuristic MaxSAT Model

From the experiments made in previous section, although compared to MaxSAT-DT, MaxSAT-BDD approach profits a lot in encoding size. However, it still suffers from the scalability problem by reporting optimality when depth increases within limited time. The encoding size of MaxSAT-BDD is in $\boldsymbol{O(M \times H \times (2^H + K))}$, where the number of examples M and the depth H are fixed when dataset and the depth are preset. Therefore, to increase the scalability of MaxSAT-BDD, we

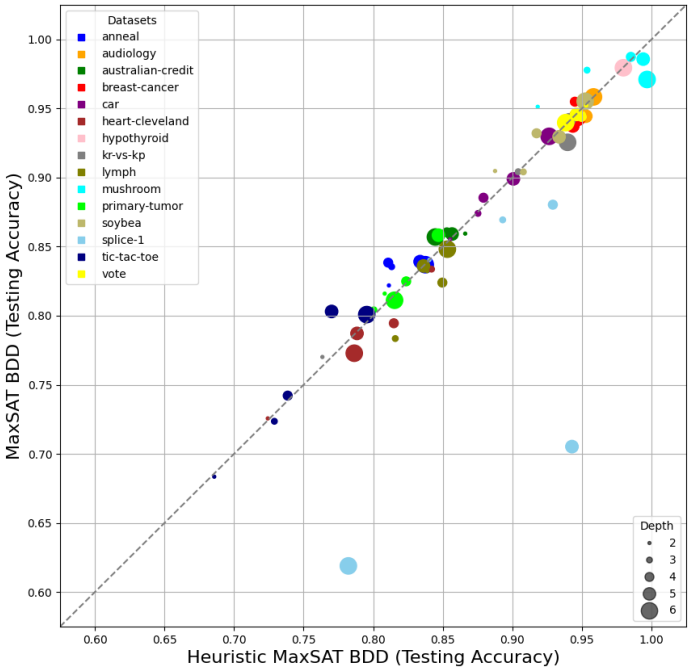


Figure 3.11: Comparison of heuristic MaxSAT-BDD and original MaxSAT-BDD in testing accuracy.

aim to reduce the number of features K. For this purpose, we propose to apply a pre-processing step to select a subset of (important) features, leading to a heuristic version of the MaxSAT-BDD approach. By doing this, the search space is greatly reduced by focusing only on the selected features.

For the pre-processing step of feature selection, we choose to run CART [Breiman *et al.* 1984], as its efficiency to build a decision tree of given depth H. Then, the features selected in the decision tree are used as the subset of important features for the MaxSAT-BDD model. Meanwhile, other methods of feature selection are also applicable for the pre-processing step, but are not considered in this study.

The experimental evaluation follows the same settings as the previous one. That is, for each dataset, we use random 5-fold cross-validation with 5 different seeds. The candidate depths selected are $H \in \{2, 3, 4, 5, 6\}$. The incomplete MaxSAT solver used is Loandra, and for each experiment, the time limit for generating formulas and the time limit for the solver are both set to 15 minutes.

Figure 3.11 shows the average testing accuracy of the heuristic MaxSAT-BDD and the MaxSAT-BDD without the pre-processing. We observe that the heuristic version is very competitive to the original exact MaxSAT-BDD in terms of learning generalization. This is particularly clear for datasets with a large number of features.

Moreover, Figure 3.12 presents the difference between the heuristic MaxSAT-BDD and the CART, where the left scatter shows the training accuracy, and the right

scatter shows the testing accuracy. From this figure, we observe that CART almost always gets better training accuracy, which is reasonable as it decides the subset of important features. However, the heuristic MaxSAT-BDD is still competitive in terms of generalisation.

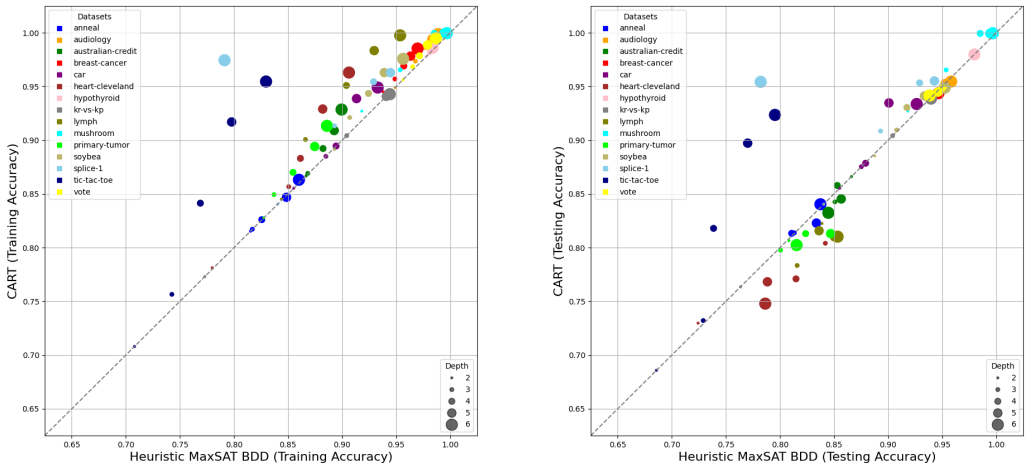


Figure 3.12: Comparison between the average prediction performance of heuristic MaxSAT-BDD and CART, the left (right) scatter indicates the training accuracy (testing accuracy).

In addition, Table 3.12 displays all details of the results of evaluation between heuristic MaxSAT-BDD, the original one (exact MaxSAT-BDD), and the CART method. As excepted, compared to the original MaxSAT-BDD, the heuristic version has favorable advantage in the encoding size, which helps handle larger problems. The column "**Opt**" indicates the percentage of instances solved to optimality (for the given subset of selected features). From the results, we observe that it is far easier for heuristic MaxSAT-BDD to report optimality than the original version, which is natural as it treat with fewer candidate features.

3.6 Summary of Chapter

In this chapter, we firstly introduced an essential proposition to build a binary decision diagram via the corresponding truth table. Then, with the help of the proposition, we proposed three progressive SAT encodings to find binary decision diagrams in smallest depths with perfect accuracy. Next, we presented how the SAT encodings are adapted to MaxSAT encodings to optimize the accuracy of binary decision diagrams in given depth. Meanwhile, we explained how to apply the post-processing of merging compatible subtrees to the MaxSAT models. Our computational experiments demonstrate the efficiency of our MaxSAT approach to learn optimal binary decision diagrams in terms of prediction performance, model

Datasets	d	CART				Heuristic MaxSAT-BDD						MaxSAT-BDD					
		Train	Test	Size	F_d	Opt	Train	Test	Size	E_Size	Time	Opt	Train	Test	Size	E_Size	Time
anneal	2	81.53	81.21	6.12	2.56	100	81.53	81.13	3.56	1.45	0.13	100	82.92	82.19	5	24.09	92.93
	3	81.72	81.38	11.08	4.92	100	81.71	81.33	5.24	4.03	1.64	0	84	83.55	7	37.21	TO
	4	82.60	81.33	18.04	8.40	100	82.57	81.08	7.08	9.64	109.65	0	84.58	83.84	9.40	52.06	TO
	5	84.69	82.29	27.88	12.32	12	84.86	83.37	11.12	20.62	780.08	0	85.33	83.92	11.72	71.08	TO
	6	86.32	84.04	39.80	17	0	86.01	83.74	13.56	42.6	845.72	0	86.26	83.70	14.68	99.47	TO
audiology	2	94.91	94.92	5	2	100	94.91	94.92	4	0.35	0.01	100	94.91	94.92	4	10.59	0.46
	3	97.36	94.82	9	4	100	96.78	95.38	5.04	1	0.02	100	96.78	95.84	5.04	16.41	6.63
	4	98.73	95.37	13.08	6	100	97.73	95.56	7.04	2.27	0.08	100	97.73	95.56	6.96	22.56	56.31
	5	99.42	95.28	17.08	8	100	98.31	95.28	9.76	4.8	0.49	72	98.40	94.44	9.88	29.82	578.99
	6	99.88	95.47	19.08	9	100	98.87	95.84	13	9.78	2.13	48	99.17	95.84	14.28	39.59	613.06
australian	2	86.68	86.62	7	3	100	86.68	86.62	4.92	1.26	0.09	100	86.7	85.94	4.72	26.79	167.99
	3	86.91	84.26	13.08	6	100	86.83	85.09	5.48	3.59	2.41	0	87.45	84.81	5.32	41.15	TO
	4	89.23	85.79	24.92	11.84	84	88.24	85.30	6.8	9.22	536.16	0	88.45	86.03	7.40	56.85	TO
	5	90.9	84.53	41.64	19.24	0	89.27	85.67	10.64	20.28	845.28	0	89.36	85.91	10.44	75.90	TO
	6	92.86	83.24	64.28	28.84	0	89.95	84.47	16.12	41.85	TO	0	90.05	85.7	17.32	102.49	TO
cancer	2	94.5	93.91	7	3	100	93.81	93.59	4	1.32	0.06	100	93.88	93.59	4	20.29	5.89
	3	95.7	94.41	13.24	6.08	100	94.89	94.14	5.64	3.78	0.52	100	95.02	93.91	5.84	31.37	525.59
	4	96.91	94.26	21.08	9.88	100	95.71	94.50	7.8	8.77	20.44	0	96.06	95.49	7.96	43.89	TO
	5	97.83	94.20	30.36	14.04	60	96.35	94.35	10.92	18.32	637.98	0	95.94	93.74	10.68	59.91	TO
	6	98.54	94.38	38.84	17.68	0	96.98	94.67	15.20	36.33	864.36	0	96.84	94.35	14.8	83.83	TO
car	2	85.53	85.53	5	2	100	85.53	85.53	4	2.77	0.14	100	85.53	85.53	4	13.32	24.82
	3	88.5	87.53	7	3	100	88.5	87.53	5	6.94	0.74	8	88.40	87.41	5.08	21.95	TO
	4	89.46	87.86	11	5	100	89.45	87.93	6.4	16.64	14.83	0	89.84	88.54	6.84	34.44	TO
	5	93.88	93.47	18.20	7.80	24	91.34	90.08	9.24	37.43	843.14	0	91.13	89.91	9.60	55.79	TO
	6	94.9	93.37	28.68	10.32	0	93.30	92.65	11.76	79.23	TO	0	93.51	92.99	13.36	97.06	TO
cleveland	2	78.13	72.97	7	2.72	100	77.99	72.43	3.76	0.55	0.04	100	79.04	72.57	4	9.48	83.84
	3	85.68	80.41	15	6.24	100	85.07	84.2	6	1.68	2.28	0	85.07	83.37	6	14.73	TO
	4	88.31	77.09	29.96	13	24	86.15	81.49	7.6	4.46	811.89	0	86.32	79.46	7.84	20.55	TO
	5	92.9	76.82	49.88	21.36	0	88.21	78.84	13.24	9.82	862.94	0	88.65	78.72	13.08	27.89	TO
	6	96.3	74.79	67.80	28.92	0	90.64	78.64	20.2	19.2	TO	0	90.74	77.29	21.04	38.66	TO
hypothyroid	2	97.84	97.84	6.92	2.96	100	97.84	97.84	4	6.2	0.43	100	97.84	97.84	4	92.65	77.76
	3	98.13	97.86	12.84	5.52	100	98.09	97.99	5.16	16.95	4.62	0	98.09	98.04	5.12	142.78	TO
	4	98.39	98.15	22.04	9.80	100	98.28	98.2	6.56	41.23	262.45	0	98.27	98.13	6.72	200.09	TO
	5	98.48	98.04	31.72	14.24	0	98.32	98.07	8.84	87.02	TO	0	98.30	98.05	9.28	274.03	TO
	6	98.6	97.99	43.56	18.92	0	98.37	97.99	13.32	175.62	TO	0	98.37	97.95	13.68	385.40	TO
kr-vs-kp	2	77.30	76.35	5	2	100	77.30	76.35	4	5.12	0.55	0	77.83	77.01	4	77.88	TO
	3	90.43	90.43	8.44	3.72	100	90.43	90.43	5.40	13.92	5.77	0	90.43	90.43	5.28	120.54	TO
	4	94.09	94.09	13.88	6.44	100	94.09	94.09	7.68	33.69	56.23	0	94.09	94.09	7.56	170.28	TO
	5	94.09	94.09	21.88	9.96	20	94.09	94.09	8.48	74.68	795.33	0	94.34	94.18	9.52	236.39	TO
	6	94.29	93.87	31.32	14.04	0	94.42	93.97	11.80	157.84	846.3	0	92.80	92.55	11.52	339.35	TO
lymph	2	84.53	82.25	7	3	100	84.39	83.89	4	0.29	0.01	100	84.46	83.23	4	3.50	2.84
	3	90.07	78.34	14.92	6.52	100	86.62	81.59	6	0.88	0.29	32	86.76	78.35	5.92	5.55	829.42
	4	95.1	80.91	25.80	11.76	100	90.40	84.97	8.36	2.16	17.38	0	90.54	82.40	8.72	7.86	TO
	5	98.34	81.58	33.40	14.96	88	92.97	83.62	12.88	4.21	277.57	0	93.51	83.60	13.52	10.94	TO
	6	99.76	81.02	36.76	15.76	40	95.37	85.33	17.64	7.78	626.2	0	95.88	84.82	17.64	15.74	TO
mushroom	2	92.71	92.71	7	3	100	91.83	91.83	4	15.61	6.82	100	95.13	95.13	4	299.19	425.74
	3	96.56	96.54	11	5	100	95.37	95.37	5.8	40.33	23.11	0	97.74	97.77	6.80	458.10	TO
	4	99.95	99.94	16.92	7.96	100	98.52	98.52	8.24	93.45	89.39	0	98.78	98.74	9	635.09	TO
	5	99.96	99.94	18.92	8.96	96	99.41	99.41	11.12	183.13	458.08	0	98.63	98.57	11.32	853.68	TO
	6	99.97	99.96	20.92	9.76	4	99.70	99.69	13.32	367.46	821.83	4	97.28	97.10	14.60	1165.88	855.34
tumor	2	82.77	80.65	7	3	100	82.75	80.83	4	0.66	0.05	100	82.8	81.6	4	3.72	5.46
	3	84.94	79.76	15	6.72	100	83.71	80.06	5.56	1.99	3.11	0	83.84	80.43	5.3	6.02	TO
	4	87.01	81.30	28.68	11.96	76	85.46	82.38	8.48	4.82	510.07	0	85.52	82.49	8.64	9.04	TO
	5	89.42	81.31	48.20	17.52	0	87.47	84.69	13.12	10.06	TO	0	87.51	85.83	13.32	13.79	TO
	6	91.34	80.24	68.60	21.04	0	88.60	81.54	20.52	19.12	TO	0	88.57	81.12	19.84	22.44	TO
soybean	2	89.13	88.54	6.92	2.76	100	89.30	88.76	4	1.17	0.05	100	90.48	90.48	4	10.79	9.51
	3	92.11	90.95	13.08	5.60	100	90.71	90.79	5	3.34	1.22	68	91.39	90.41	6.52	16.99	706.39
	4	94.36	93.05	21	9.40	100	92.44	91.75	7.92	7.9	96.96	0	93.24	93.21	9.04	24.54	TO
	5	96.29	94.13	31.88	13.40	28	93.90	93.37	11.08	16.58	804.03	0	94.31	92.95	11.92	35.34	TO
	6	97.58	94.89	43.48	16.88	0	95.65	95.24	14.56	33.04	TO	0	96.07	95.52	14.88	53.41	TO
splice-1	2	84.04	84.04	7	3	100	84.04	84.04	4	6.13	4.06	0	84.04	84.04	4	296.61	TO
	3	91.34	90.85	15	6.20	100	89.31	89.31	5.32	17.7	67.13	0	87.25	86.94	5.44	449.04	TO
	4	95.44	95.34	28.52	10.60	4	92.92	92.92	7.36	42.15	841.79	0	88.30	88.04	7.24	608.30	TO
	5	96.29	95.52	46.92	17.20	0	94.48	94.29	9.68	93.08	TO	0	71.99	70.53	10.28	783.90	TO
	6	97.45	95.44	76.44	29.64	0	79.13	78.22	15.12	205.48	TO	0	62.92	61.89	16.28	996.27	TO
tic-tac-toe	2	70.80	68.58	7	2.96	100	70.80	68.58	3.76	1.84	0.65	100	71.05	68.35	4	9.25	48.01
	3	75.65	73.21	15	6.76	100	74.26	72.90	6.04	5.6	92.88	0	74.91	72.36	6.16	15.01	TO
	4	84.14	81.8	27	11.04	0	76.89	73.86	9.04	12.99	TO	0	76.87	74.22	8.84	22.88	TO
	5	91.7	89.73	43.72	16	0	79.79	77.02	14.24	27.15	TO	0	81.86	80.31	13.88	35.67	TO
	6	95.46	92.36	66.68	19.28	0	82.97	79.54	24.04	52.34	TO	0	84.82	80.08	24.16	59.52	TO
vote	2	95.63	94.90	7	3	100	95.66	95.31	3.12	0.85	0.02	100	95.68	95.22	3.76	7.20	0.70
	3	96.84	94.94	14.44	6.56	100	96.53	95.03	5.56	2.52	0.35	100	96.69	94.57	5.56	11.38	94.11
	4	97.86	94.80	24.60	10.72	100	97.16	94.99	8	5.86	13.07	8	97.40	94.39	8.16	16.49	TO
	5	98.87	94.48	34.20	14.16	92	97.91	94.62	12.24	11.77	280.12	0	98.21	94.57	12.40	23.83	TO
	6	99.49	94.16	40.04	16.60	60	98.69	93.84	18.08	22.77	507.15	4	98.93	93.98	18.44	36.20	TO

Table 3.12: Details of evaluation between heuristic MaxSAT-BDD, the original MaxSAT-BDD, and CART.

size, and encoding size. At the end, we showed an heuristic version based on feature selection, to increase the scalability of our MaxSAT approach without losing the prediction quality.

Conclusions and Future Works

Conclusions

We brought contributions to MaxSAT-based exact methods to learn optimal interpretable machine learning models, specifically in optimal decision trees and optimal binary decision diagrams.

- **Optimal Decision Trees**
  By adapting the previous SAT encoding of the literature, we propose the first MaxSAT-based exact method to learn optimal decision trees aiming to optimize the empirical accuracy. Then, we introduce some new constraints to restrict the tree depth. Finally, benefiting from the nature of MaxSAT framework, we integrate the proposed encoding in boosting techniques to improve the generalization performance.

  The initial motivation behind this work is to avoid the overfitting issue and to address the scalability issue. The experimental results at first confirm the exist of overfitting phenomenon in previous SAT approach. Secondly, they show that a larger tree topology could be explored with our MaxSAT approach. Meanwhile, the decision trees found obtain good prediction quality compared to heuristic method and a recent state-of-the-art exact method. At last, clear improvements in generalization performance are observed after the integration of boosting technique.

- **Optimal Binary Decision Diagrams**
  In particular, we propose the first MaxSAT-based exact method to learn optimal binary decision diagrams, which optimizes the empirical accuracy. At first, we introduce an initial SAT encoding to model binary decision diagrams in limited depth with perfect accuracy. Then, we present how to adapt the SAT-based model into MaxSAT approach. Finally, we present a preprocessing for selecting some important features to increase the scalability of our MaxSAT-based approach.

  The initial motivation behind this work is to obtain more compact interpretable machine learning models. Compared to state-of-the-art heuristic methods, the experimental results show clear advances of our MaxSAT-based method in prediction quality. Furthermore, a huge shrink in formulation size and model size is observed in the comparison between our MaxSAT-based method and state-of-the-art exact method with competitive prediction performance. Additionally, great reductions in encoding size are displayed after the application of pre-processing, which somehow reduces the scalability issue.

Future Works

There are a number of potential valuable future research directions in the filed of exact methods to learn optimal interpretable machine learning models. We present some interesting directions that strongly relate to our contributions.

- **Optimal Multi-Variate Decision Trees via MaxSAT**
  Currently, recent exact methods mostly aims to find optimal single-variate decision trees, where each branching node corresponds to a single feature. It would be interesting to propose a MaxSAT-based exact method to learn optimal multi-variate decision trees [Murthy *et al.* 1993, Brodley & Utgoff 1995], in which each branching node corresponds to a combination of multiple features. The usage of multi-variate could grow the search space, leading to optimal decision trees with better prediction quality, but without the growth in structure.

- **Integration of Incremental Learning in Learning Optimal Decision Trees**
  To the best of our knowledge, current exact methods of learning optimal decision trees are mostly declarative and independent. The property declarative is that each mathematical formulation corresponds to a search problem of finding a decision tree restricted for a given dataset. The property independent is that there is no connection between mathematical formulations for similar search problems, like finding decision trees with different depths but for the same dataset, or finding decision trees with same restrictions but for different datasets. In addition, in general exact methods suffers the scalability problem, it would be interesting to integrate incremental learning of decision trees [Utgoff *et al.* 1997] in exact methods. This integration could somehow improve the scalability of exact methods when treating a large dataset part by part incrementally.

- **Optimal Multi-Variate Decision Graphs via MaxSAT**
  In the literature review of exact methods for optimal decision graph, we mention a recent MILP-based exact method [Florio *et al.* 2022], where the decision graphs found are multi-variate. The advantage of multi-variate has been described in the first direction. Based on the MaxSAT-based exact method we proposed, it would be interesting to propose new variables and constraints to realise the multi-variate structure.

- **Faster Solvers for Learning Optimal Interpretable Machine Learning Models**
  As current SAT-based or MaxSAT-based exact methods to learn optimal interpretable machine learning models are generally suffers the scalability problem, it would be an interesting direction to explore faster specific solvers for such problems. One possible way is to integrate some machine learning methods

into the branching heuristic of solvers for choosing variables during the search process [Bengio *et al.* 2021]. This idea is not limited for SAT or MaxSAT solvers, but for all solvers based on search.

Appendices

Appendix A

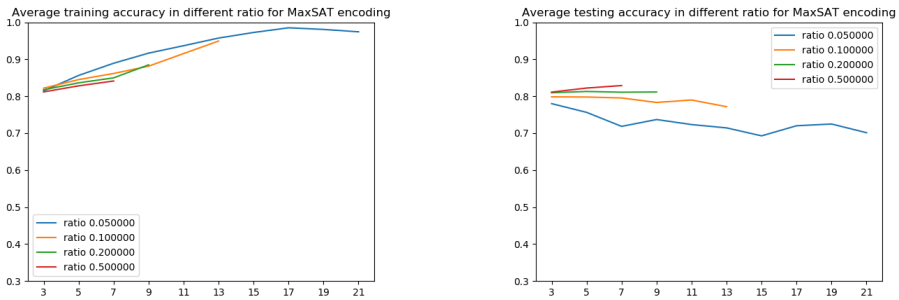
Detailed Results for the Overfitting Phenomenon

In this appendix, we present the detailed results for the overfitting phenomenon of previous SAT method to learn optimal decision trees with perfect empirical accuracy, which is described in Section 2.4.

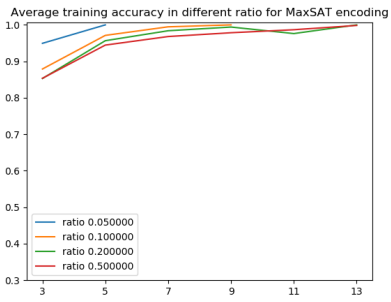
We recall the experimental setting. We use hold-out method to split training and testing set, we choose 4 different ratios $r = \{0.05, 0.1, 0.2, 0.5\}$ (in Section 2.4 only three small ratios are chosen, but here we show additional results) to generate training set, and the remaining examples are set as testing set. This process is repeated 10 times with different ratios, and the complete MaxSAT solver we used is RC2. Although it should not set time limit until the solver finds optimal solution, considering the scalability, for each experiment, the global time limit for the solver is 30 hours.

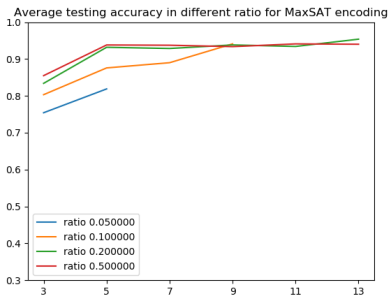
We present the tendency of average training and testing accuracy separately by the dataset, where the left figure indicates the training accuracy, and the right figure indicates the testing accuracy. In each figure, the x-axis indicates the tree size of the decision tree found, and the y-axis indicates the accuracy. We mention the basic information of each dataset at the same time.

- "*anneal*", 812 examples, 42 original features, 89 binarized features:

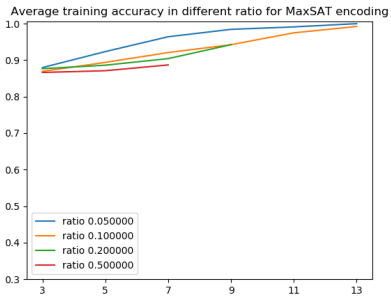


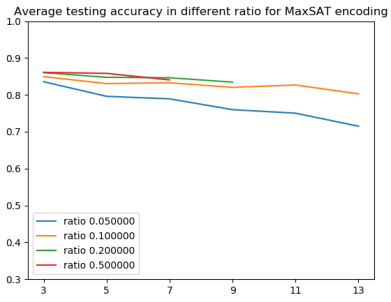
- "*audilogy*", 216 examples, 66 original features, 146 binarized features:



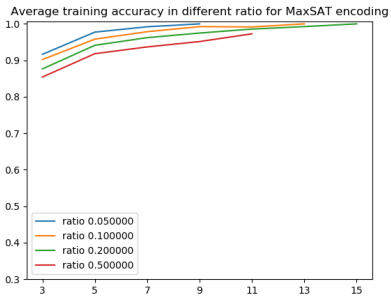


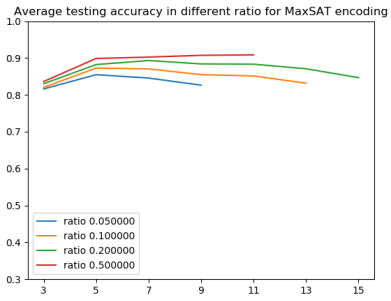
- "*austrlian-credit*", 653 examples, 51 original features, 124 binarized features:



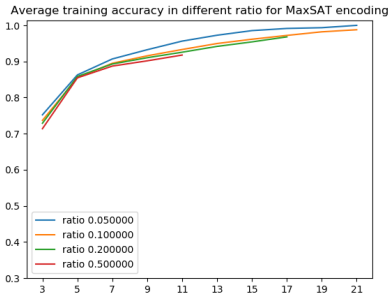


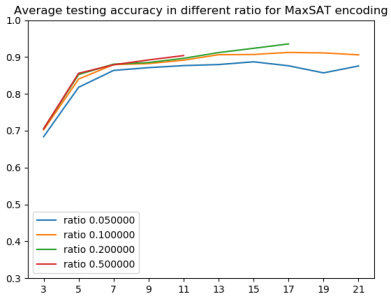
- "*breast-cancer*", 683 examples, 9 original features, 89 binarized features:



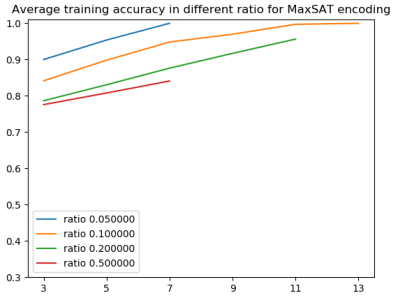


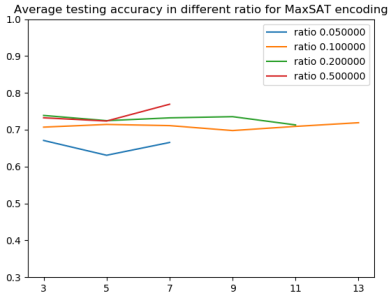
- "*car*", 1728 examples, 6 original features, 21 binarized features:



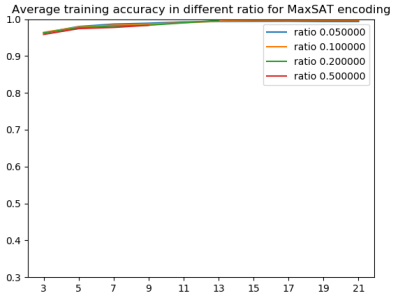


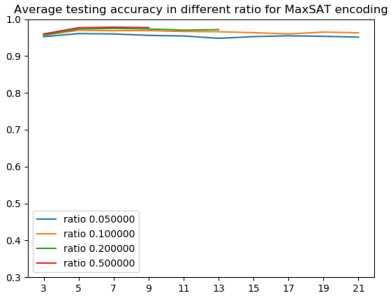
- "*heart-cleveland*", 296 examples, 45 original features, 95 binarized features:



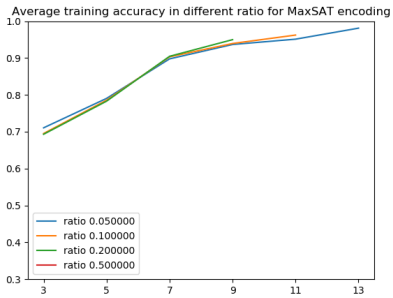


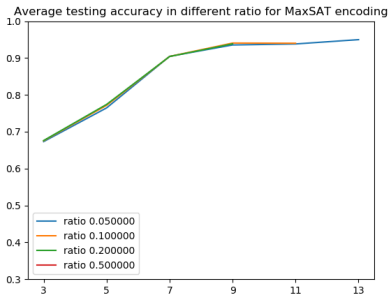
- "*hypothrid*", 3247 examples, 43 original features, 86 binarized features:



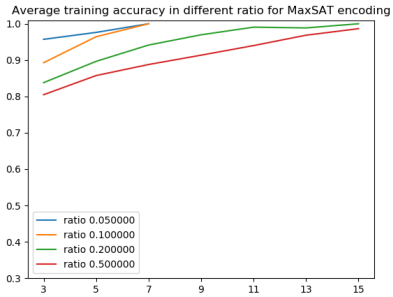


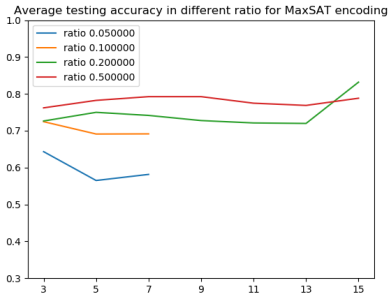
- "*kr-vs-kp*", 3196 examples, 36 original features, 73 binarized features:



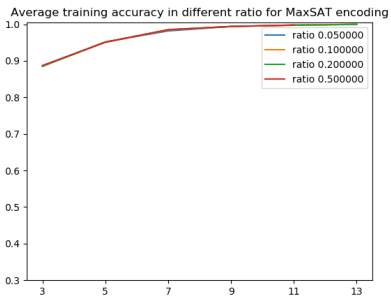


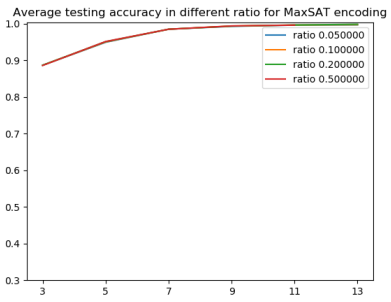
- "*lymph*", 147 examples, 27 original features, 68 binarized features:



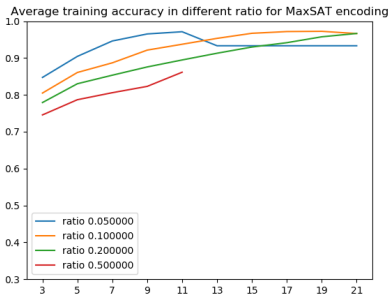


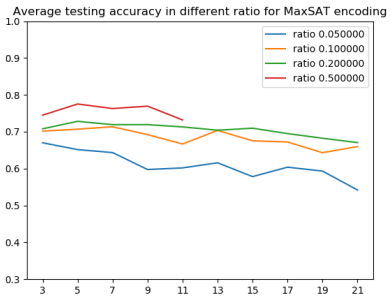
- "*mushroom*", 8124 examples, 21 original features, 112 binarized features:



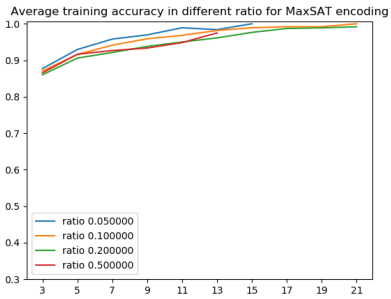


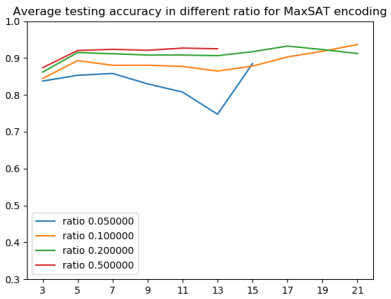
- "*primary-tumor*", 336 examples, 15 original features, 31 binarized features:



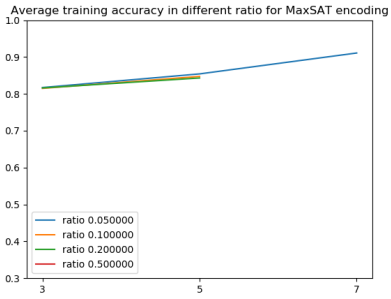


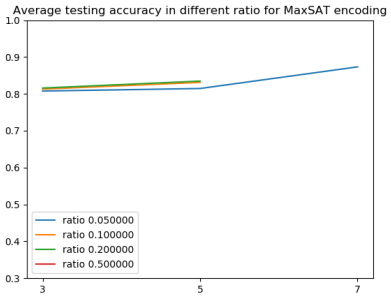
- "*soybean*", 630 examples, 16 original features, 50 binarized features:



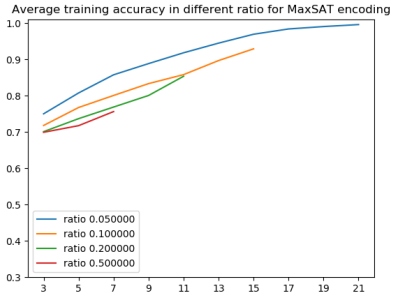


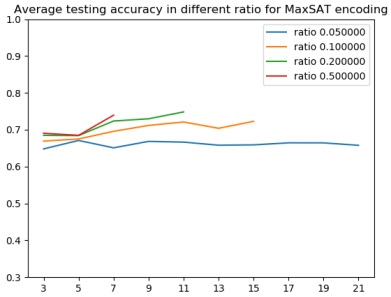
- "*splice-1*", 3190 examples, 60 original features, 287 binarized features:



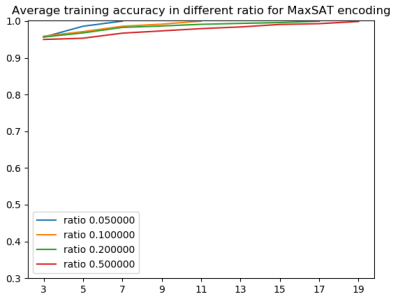


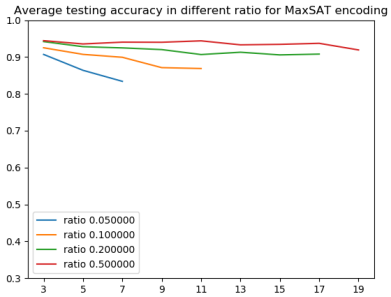
- "*tic-tac-toe*", 958 examples, 9 original features, 27 binarized features:





- "*vote*", 435 examples, 16 original features, 48 binarized features:





The overfitting phenomenon is not remarkable for every dataset, but we almost systematically observe a plateau whereby the testing accuracy stays constant at best while the training accuracy increases.

Appendix B

Description of Benchmarks for Learning Optimal Decision Trees and Boosted Trees

MaxSAT Evaluation 2021: Solver and Benchmark Descriptions, volume B-2021-2 of Department of Computer Science Series of Publications B, University of Helsinki 2021.

Description of Benchmarks on Learning Optimal Decision Trees and Boosted Trees

Hao Hu, Emmanuel Hebrard, Marie-José Huguet, Mohamed Siala
LAAS-CNRS, Université de Toulouse, INSA, Toulouse, France
{hhu, hebrard, huguet, siala}@laas.fr

I. Introduction

Decision Trees are one of the most essential models in machine learning, as they are both intrepretable and effective to compute. Unlike traditional top-down heuristic induction for computing decision trees, recently, several exact methods have been introduced to find optimal decision trees via different declarative methods, such as Constraint Programming [4], Boolean Satisfiablity(SAT) [3], and MaxSAT [2]. The objective of the MaxSAT approach is to find decision trees with limited depths that maximize the number of examples correctly classified. It performs better in prediction for unseen data than the SAT approach, as the SAT approach requires perfect accuracy leading to overfitting.

As other exact methods of learning optimal decision trees, the MaxSAT approach also has scalability issues. However, incomplete MaxSAT solvers can produce high quality solutions within a limited time. In addition, the MaxSAT approach can be easily adapted to classic Boosting methods such as AdaBoost [1], to improve the prediction performance. The adaptation is realized by updating the weights of soft clauses corresponding to examples to update the data distribution of each iteration in AdaBoost.

II. MaxSAT Approach of Learning Optimal Decision Trees

A. Problem Definition

The problem solved by the MaxSAT approach is the following optimization problem:

$P(\mathcal{E}, N)$: *Given a set of examples $\mathcal{E}$, find a full binary decision tree of size N that maximizes the number of examples in $\mathcal{E}$ that are correctly classified.*

Since non-binary features can always be transformed as binary features, binary decision trees can handle all data sets. Moreover, to limit the tree depth described in the Introduction, constraints for controlling the size and depth of the tree can be posted in the MaxSAT approach.

To solve $P(\mathcal{E}, N)$, for each example $e_q \in \mathcal{E}$, the MaxSAT approach introduces a Boolean variable b_q, where b_q is true if and only if e_q is correctly classified. Then, all b_q are set as soft clauses and other constraints are set as hard clauses. Therefore, assuming the set of examples $\mathcal{E}$ used is consistent, the unweighted MaxSAT formulation is used.

B. MaxSAT Encoding

The MaxSAT encoding in [2] is largely based on the SAT model from [3] that it extends. The SAT encoding consists of three parts:

- **Part 1**: Constraints on the structure of a valid binary tree in fixed size.
- **Part 2**: Constraints for mapping features (respectively, classes) to internal nodes (respectively, leaf nodes).
- **Part 3**: Constraints for correctly classifying all examples in the example set.

To lift the SAT model into a MaxSAT encoding, for each example e_q, every constraint of **Part 3** concerning e_q is linked to a variable b_q acting as the blocking literal. Then, to achieve the limit in maximum(or exact) depth for decision trees found, two more parts of constraints are added:

- **Part 4**: Constraints for controlling trees in fixed depth.
- **Part 5**: Constraints for controlling tree size under an upper bound not a fixed sized.

Finally, all constraints mentioned are set as hard clauses, and all blocking literal b_q as soft clauses. There is no weight function on the clauses as in this case we try to learn the tree with optimal accuracy and no example is more important than the others.

III. AdaBoost Adaptation

A. AdaBoost Algorithm

Boosting methods are a family of ensemble methods, which train multiple dependent classifiers with the same data set and then combine them to get better predictions than a single classifier. As a typical Boosting method, AdaBoost [1] builds T classifiers in a sequence of T iterations. At each iteration t, AdaBoost learns a classifier h_t and updates the data distribution of the $(t+1)$-th iteration $\mathcal{D}_{t+1}$ based on the t-th data distribution $\mathcal{D}_t$ using the equation 1:

$$\mathcal{D}_{t+1}(x_q) = \frac{\mathcal{D}_t(x_q)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_q) = c_q \\ \exp(\alpha_t) & \text{if } h_t(x_q) \neq c_q \end{cases} \quad (1)$$

In equation 1, each example $e_q = (x_q, c_q)$ is a 2-tuple, where x_q denotes the value vector for all features of this example, and $c_q \in \{0, 1\}$ denotes its class. The coefficient $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$ helps the previously misclassified examples gain more importance in the next iteration, where ϵ_t is the error rate of t-th iteration. Z_t is a normalization factor.

The final predictor is a weighted vote where every classifier h_t is associated with a weight α_t, which is calculated as follows:

$$H(x_q) = 1 \text{ if } \sum_{t=1}^{T} \alpha_t g(h_t(x_q)) > 0 \text{ and } 0 \text{ otherwise} \quad (2)$$

where $g(0) = -1$, $g(1) = 1$. The function H denotes the aggregated predictor.

B. Integration in the MaxSAT Approach

To integrate AdaBoost in the MaxSAT approach, the key idea is to update the data distribution by updating the weights of soft clauses corresponding to examples. The final weighted voting follows the original AdaBoost algorithm in Equation 2.

As the weighted MaxSAT formulation allows only positive integer weights, weights updated from Equation 1 are approximated. We set all weights at the first iteration with the value 1 as initial distribution. Then, two steps of approximation are made to calculate the positive integer weight w_q^{t+1} of soft clause b_q in $(t+1)$-th iteration based on w_q^t, the corresponding weight in previous iteration. Firstly, we update and normalize the weights:

$$\hat{w}_q^{t+1} = \frac{w_q^t * factor_q^t}{\sum_{q=1}^{M}(w_q^t * factor_q^t)} \quad (3)$$

where $factor_q^t$ is the factor based on the prediction:

$$factor_q^t = \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_q) = c_q \\ \exp(\alpha_t) & \text{if } h_t(x_q) \neq c_q \end{cases} \quad (4)$$

Secondly, we discretize the weight $\hat{w}_q^{t+1}$ as follows:

$$w_q^{t+1} = round(\frac{\hat{w}_q^{t+1}}{\min_{i\in\{1,\ldots,M\}}(\hat{w}_i^{t+1})}) \quad (5)$$

IV. Benchmark Instances

There are two first-level folders in the zip archive. The first one is named ***"decision-tree"*** and contains 60 WCNF files that correspond to learning optimal decision trees with 4 different maximum depths for 15 datasets. These benchmarks are suited to the ***unweighted incomplete*** track. The other is named ***"adaboost"*** and contains 120 WCNF files that correspond to learning boosted trees with 4 different maximum depths for 6 datasets in 5 different iterations. These benchmarks are suited to the ***weighted incomplete*** track.

Both first-level folders contains several second-level folders, which correspond to the names of the encoded ML datasets. Each second-level folder contains all WCNF files corresponding to this dataset. The datasets we used to generate WCNF are from CP4IM[1]. More precisely, they are binarized with the *one-hot-encoding*. Since AdaBoost greatly improves the training and test accuracy in some cases, we selected the datasets in which classic decision trees performed poorly to generate WCNF for AdaBoost. Further information on those datasets is given in Table I, where $\#s$ indicates the number of instances, $\#f_b$ indicates the number of binarized features.

[1] https://dtai.cs.kuleuven.be/CP4IM/datasets/

TABLE I
INFORMATION OF DATASETS FOR LEARNING OPTIMAL BOOSTED TREES.

Dataset	anneal	australian	car	heart	tumor	tic-tac-toe
$\#s$ / $\#f_b$	812/89	653/124	1728/21	296/95	336/31	958/27

The name of each WCNF file follows the format: formula_*ratio_seed*_atleast_*size_maxdepth*_reduced_incomplete_*type*.WCNF.

- *ratio*: The sample ratio used when generating a training set using the hold-out method.
- *seed*: The seed used to make the stratified sampling. By default, we use 2021.
- *size*: The upper bound on the size of the decision tree.
- *maxdepth*: The upper bound on the depth of the decision tree.
- *type*: The application of the decision tree generation problem encoded by this WCNF file. There are two possible types:
  - *tree*: This WCNF is for learning a classic decision tree.
  - *adaboost_iter*: This WCNF is for learning decision tree for the (*iter*)-th iteration of AdaBoost.

References

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997. [Online]. Available: https://doi.org/10.1006/jcss.1997.1504

[2] H. Hu, M. Siala, E. Hebrard, and M. Huguet, "Learning optimal decision trees with maxsat and its integration in adaboost," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, C. Bessiere, Ed. ijcai.org, 2020, pp. 1170–1176. [Online]. Available: https://doi.org/10.24963/ijcai.2020/163

[3] N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva, "Learning optimal decision trees with SAT," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence - IJCAI*, July 13-19, 2018, pp. 1362–1368.

[4] H. Verhaeghe, S. Nijssen, G. Pesant, C. Quimper, and P. Schaus, "Learning optimal decision trees using constraint programming," *Constraints An Int. J.*, vol. 25, no. 3-4, pp. 226–250, 2020. [Online]. Available: https://doi.org/10.1007/s10601-020-09312-3

Appendix C

Résumé Étendu

Dans cette annexe, nous décrivons les contributions principales de la thèse.

C.1 Introduction

Au cours de la dernière décennie, l'apprentissage automatique a remporté de grands succès pour resoudre de nombreux problèmes du monde réel. Ces succès ont accru le besoin d'explications en Intelligence Artificielle (IA), en particulier pour les systèmes d'aide à la décision ayant un impact sur les individus. Le domaine de (*Explanaible AI* ou XAI) inclut un large spectre de recherche dont un des buts est de produire des systèmes d'IA compréhensibles par les humains afin d'augmenter la confiance qu'ils peuvent avoir dans les systèmes d'IA. L'objectif est de pouvoir comprendre le processus conduisant à une décision ou à une prédiction donnée.

Dans cette thèse, nous nous intéressons à l'interprétabilité de l'apprntissage automatique. Dans la littérature, deux approches principales existent pour augmenter cette interprétabilité. La première approche, appelée *post hoc*, se concentre sur des modèles de type boîte noire, par exemple les réseaux de neurones ou le deep learning, et considère des explications a posteriori [Guidotti *et al.* 2018]. Elle peut produire des explications sur la sortie du modèle de boîte noire pour détailler la raison d'une prédiction donnée ou une inspection de boîte noire pour expliquer comment fonctionne le modèle boîte noire. La deuxième approche est appelée *transparence-by-design*. Son objectif est de produire des modèles d'apprentissage compréhensibles pour les humains, grâce aux leurs structures simples. Par exemple, les *arbres de décision*, ou les *règles de décision* sont considérés comme des modèles interprétables lorsqu'ils sont de petite taille.

Certains inconvénients de l'approche *post hoc* qui peut fournir des explications trompeuses ou fausses, ont été mis en évidence dans [Rudin 2019, Laugel *et al.* 2019]. Pour les applications cruciales, où les décisions peuvent avoir un impact sur les individus, de tels inconvénients soulèvent la necessité de modèles transparents en apprentissage automatique. Dans [Rudin *et al.* 2021], dix défis pour le développement de modèles d'apprentissage interprétables intrinsèquement sont détaillés. Le premier défi concerne la manière d'apprendre efficacement des modèles d'apprentissage optimaux et parcimonieux. Cette thèse s'inscrit dans cet défi.

Il existe de nombreuses méthodes heuristiques permettant d'obtenir des modèles d'apprentissage interprétables. Bien ces méthodes heuristiques classiques aient un temps de calcul réduit, les modèles d'apprentissage interprétables construits

sont souvent de grande taille, ce qui rend difficile la compréhension de son fonctionnement. Récemment, de nombreuses méthodes exactes ont été proposées pour obtenir des modèles interprétables. Par rapport aux méthodes heuristiques, les méthodes exactes offrent la promesse d'une optimalité, par exemple sur la *taille de modèle* ou sa *précision*. Dans ce contexte, les méthodes d'optimisation combinatoire, telles que la *programmation par contraintes*, la *programmation linéaire mixte en nombres entiers*, la *satisfaisabilité booléenne (SAT)* et la *programmation dynamique* ont été appliquées avec succès pour l'apprentissage de modèles interprétables optimaux. Ces approches déclaratives sont particulièrement intéressantes car elles offrent une certaine flexibilité pour gérer des exigences supplémentaires pendant l'apprentissage d'un modèle.

Dans cette thèse, nous nous concentrons sur une approche basée sur la *Satisfiabilité Booléenne Maximale* (MaxSAT), pour apprendre des modèles interprétables optimaux, où MaxSAT est une version d'optimisation de SAT, Un inconvénient des méthodes exactes basées sur SAT est la recherche d'une précision empirique parfaite pour une taille donnée de modèle ce qui présente un risque de sur-apprentissage ou *overfitting*. Ainsi, il s'agit d'étudier si les méthodes exactes basées sur MaxSAT peuvent éviter cet inconvénient en optimisant la précision empirique. De plus, le formalisme MaxSAT a une extension permettant l'utilisation de clauses pondérées, où les poids des clauses peuvent naturellement se rapprocher de la distribution des exemples composant les jeux de données utilisés dans l'apprentissage. Cette extension rend les méthodes exactes basées sur MaxSAT faciles à adapter avec les méthodes d'apprentissage d'ensemble de type Boosting.

Cette thèse présente deux contributions principales :

1. Une nouvelle formalisation avec MaxSAT pour apprendre des *arbres de décisions optimaux*, et son extension basée sur *AdaBoost*. .

2. Une nouvelle formalisation avec MaxSAT pour apprendre des *diagrammes de décisions binaires optimaux*.

Nos contributions sont évaluées par une approche expérimentale. Nous décrivons ces contributions et les résultats obtenus dans les sections suivantes.

C.2 Arbres de décision optimaux par MaxSAT et integration dans AdaBoost

En tant que modèle d'apprentissage très populaire, les arbres de décision bénéficient principalement de leur interprétabilité et du large éventail de méthodes efficaces pour les calculer. Plusieurs méthodes heuristiques classiques construisent généralement l'arbre de haut en bas, en divisant les ensembles de données avec des caractéristiques sélectionnées par différentes métriques heuristiques. Cependant, ces méthodes heuristiques souffrent de la difficulté d'interprétabilité, en raison de l'explosion de la taille et de la profondeur des arbres générés.

Récemment, plusieurs méthodes exactes d'apprentissage d'arbres de décision optimaux ont été proposées pour offrir des garanties d'optimalité. La Section 1.3.1.2 fournit la revue de la littérature de certaines approches d'optimisation combinatoire pour les arbres de décision optimaux. Dans ces approches, les critères largement utilisées pour déterminer des arbres de décision optimaux sont la *taille de l'arbre*, la *profondeur de l'arbre* et la *précision*. Contrairement à d'autres approches d'optimisation combinatoire, il n'existe pas de méthodes exactes basées sur la *Satisfaisabilité Booléenne* (ou ses variantes) pour optimiser la *précision* de l'arbre de décision. En fait, les précédentes méthodes exactes en satisfisfiabilité booléenn visent à garantir la précision parfait sur un ensemble d'entrainment et sont basées sur SAT [Bessiere *et al.* 2009, Narodytska *et al.* 2018]. Ces méthodes optimisent la taille des arbres, et [Avellaneda 2020, Janota & Morgado 2020] optimisent la profondeur des arbres permettant d'obtenir une precision parfaite sur l'ensemble d'entrainement, ce qui est souvent critiqué car cela peut entraîner un phénomène d'overfitting.

Par exemple, le *problème decision* résolu par le modèle SAT en [Narodytska *et al.* 2018] est:

- $P_{dt}(\mathcal{E}, N)$*: Pour un ensemble d'exemples* $\mathcal{E}$ *donné, existe-t-il un arbre de décision binaire valide (chaque nœud interne a exactement deux enfants) de taille* N*, qui classifie correctement tous les exemples de* $\mathcal{E}$ *?*

Ce modèle SAT pour le problème $P_{dt}(\mathcal{E}, N)$ [Narodytska *et al.* 2018] s'appuie sur *trois familles* de contraintes (les détails sont dans la Section 2.2) :

- ***Partie 1****: Contraintes pour encoder un arbre binaire valide de taille* N*.*
- ***Partie 2****: Contraintes pour relier les attributs (respectivement, les classes) aux nœuds internes (respectivement, nœuds feuilles).*
- ***Partie 3****: Contraintes pour classifier tous les exemples correctement.*

Pour répondre au risque d'overfitting, les approches MaxSAT peuvent être utilisées pour maximiser le nombre d'exemples correctement classifiés. De plus, la résolution de problème MaxSAT peut s'appuyer sur l'utilisation de solvers MaxSAT dit incomplets, ce qui augmente l'extensibilité, grâce aux meilleurs resultats obtenus dans des temps limités.

Dans cette thèse, le *problème d'optimisation* considéré est le suivant :

- $P^*_{dt}(\mathcal{E}, H)$*: Pour un ensemble d'exemples* $\mathcal{E}$ *donné, trouver un arbre de décision binaire valide de profondeur maximale/exacte fixée* H*, qui maximise le nombre d'exemples de* $\mathcal{E}$ *correctement classifiés.*

Trois adapatations sont proposées pour arriver à résoudre le problème d'optimisation $P^*_{dt}(\mathcal{E}, H)$:

- ***Adapatation 1****: Modèle MaxSAT pour résoudre le problème d'optimisation $P^*_{dt}(\mathcal{E}, N)$: Pour un ensemble d'exemples $\mathcal{E}$ donné, trouver un arbre de décision binaire valide de taille N, qui maximise le nombre d'exemples de $\mathcal{E}$ correctement classifiés.*

- ***Adapatation 2****: Ajouter des contraintes pour contrôler la profondeur maximale/exacte de l'arbre de taille donnée N. Cette adaptation consiste à proposer un modèle MaxSAT pour résoudre le problème $P^*_{dt}(\mathcal{E}, N, H)$: trouver un arbre de décision binaire de taille N avec une profondeur maximale/exacte H, qui maximise le nombre d'exemples de $\mathcal{E}$ correctement classifiés.*

- ***Adapatation 3****: Ajouter des contraintes pour encoder la relaxation de la taille de l'arbre avec N comme borne supérieure. Cette adaptation consiste à proposer un modèle MaxSAT pour résoudre le problème $P^*_{dt}(\mathcal{E}, [N_l, N], H)$: trouver un arbre de décision binaire dont la taille est dans $[N_l, N]$, avec une profondeur maximale/exacte H, qui maximise le nombre d'exemples de $\mathcal{E}$ correctement classifiés.*

L'intégration de ces trois adaptations permet de résoudre le problème $P^*_{dt}(\mathcal{E}, H)$ visé. Plus précisément, on peut résoudre le problème $P^*_{dt}(\mathcal{E}, [3, 2^{H+1}-1], H)$ pour la contrainte sur la profondeur maximale, et le problème $P^*_{dt}(\mathcal{E}, [2H+1, 2^{H+1}-1], H)$ pour la contrainte sur la profondeur exacte. Additionellement, le modèle MaxSAT peut être intégré dans AdaBoost pour améliorer les performances de prédiction. L'idée est de mettre à jour les distributions de données en modifiant les poids des clauses souples correspondantes. Les détails sont decrits dans la Section 2.5.

C.2.1 Modèle MaxSAT proposé

C.2.1.1 Modèle MaxSAT adapaté pour $P^*_{dt}(\mathcal{E}, N)$

Comme nous l'avons introduit précédemment, le modèle SAT de la littérature contient trois familles de contraintes, où les contraintes de classifier correctement tous les exemples sont importantes. Par conséquent, pour réaliser la première adaptation, les contraintes (Contraintes de *Partie 1 et Partie 2*) sont conservées comme des ***clauses dures***. Ensuite, pour classer chaque exemple (Contraintes de *Partie 3*), nous proposons une variable booléenne b_q pour chaque exemple $e_q \in \mathcal{E}$, où b_q est **vrai** *si et seulement si* l'exemple e_q est correctement classifié.

Ensuite, nous relions la variable b_q aux contraintes de classification des exemples (Contraintes de *Partie 3*) en tant que ***clauses dures***. Autrement dit, pour tout exemple positif $e_q \in \mathcal{E}^+$, et tout nœud feuille j, avec $j = 1, \ldots, N$:

$$b_q \rightarrow (v_j \wedge \neg c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r,q)}) \tag{C.1}$$

Pour tout exemple negatif $e_q \in \mathcal{E}^-$, et tout nœud feuille j, avec $j = 1, \ldots, N$:

$$b_q \rightarrow (v_j \wedge c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r,q)}) \tag{C.2}$$

Les eexpressions dans les parenthèses sont des contraintes de classification pour l'exemple e_q dans le nœud feuille j. Afin de modéliser l'objectif de maximisation du nombre d'exemples correctement classifiés, chaque littéral b_q est déclaré comme une ***clause souple***. D'après la définition de b_q, le nombre de clauses souples satisfaites est égal au nombre d'exemples correctement classifiés.

C.2.1.2 Contraintes pour contrôler la profondeur de l'arbre de taille fixée

La seconde adaptation vise à contrôler la profondeur H pour un arbre ayant une taille donnée N. En effet, il existe des topologies différentes pour un arbre de décision binaire avec la même taille. Par exemple, la Figure C.1 montre les deux situations extrêmes de la topologie d'un arbre binaire utilisant la même taille $N = 7$: un arbre binaire complet (équilibré) (celui de gauche de profondeur $H = 2$), et un arbre binaire totalement déséquilibré (celui de droite de profondeur $H = 3$). Notez que nous comptons la profondeur de l'arbre à partir de la racine comme profondeur 0.

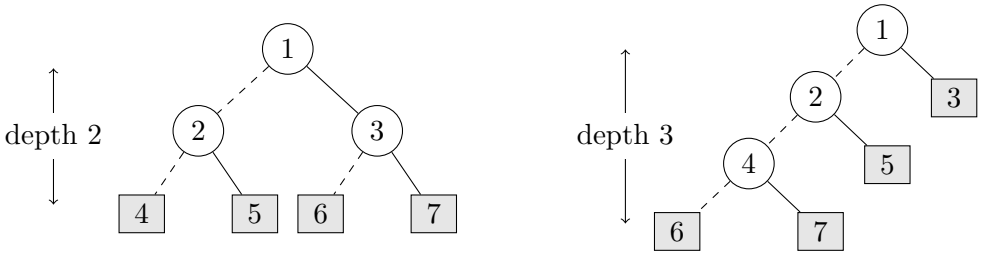


Figure C.1: Deux arbres binaires valides de taille 7 avec des profondeurs différentes.

Dans un arbre binaire, la profondeur correspondante d'un nœud j donné varie dans un intervalle reflétant ces deux situations extrêmes (les noeuds sont numérotés par niveau).

- la *borne supérieure* de la profondeur est associée à l'arbre entièrement déséquilibré, qui est $\lceil (j-1)/2 \rceil$.
- la *borne inférieure* de la profondeur est associée à l'arbre complet (équilibré), qui est $\lceil \log(j+1) \rceil - 1$.

Pour refléter cette propriété entre le nœud et sa profondeur, nous introduisons une variable booléenne $depth_{jt}$ pour indiquer que le nœud j est en profondeur t ou pas, $depth_{jt}$ est **vrai** *si et seulement si* le nœud j est en profondeur t. Les idées principales pour exprimer les contraintes controlant la profondeur des noeuds sont données ci-dessous (les détails sont dans la Section 2.3) :

- *La racine doit toujours être à la profondeur* 0.
- *Chaque nœud doit être à une seule profondeur.*
- *La relation de l'augmentation de la profondeur. En détail, si le nœud i est en profondeur t, et le nœud j est un enfant du nœud i, alors le nœud j doit être en profondeur $t+1$.*
- *Pour contrôler H comme la* ***profondeur maximale****, tous les nœuds possibles à la profondeur H doivent être des nœuds feuilles.*
- *Pour contrôler H comme la* ***profondeur exacte****, il faut qu'au moins d'un nœud se trouve à la profondeur H.*

Pour combiner les contraintes de contrôle de la profondeur dans le modèle MaxSAT, nous gardons simplement ces contraintes comme des ***clauses dures***, parce qu'elles sont considérées comme une extension des contraintes structurelles.

C.2.1.3 Contraintes pour relaxer la taille de l'arbre

Après la première et la seconde adaptation, le modèle MaxSAT peut non seulement résoudre le problème d'optimisation $P^*_{dt}(\mathcal{E}, N)$, mais aussi contrôler la profondeur maximale ou exacte. La dernière adaptation vise à relaxer la taille de l'arbre pour résoudre notre problème d'optimisation original $P^*_{dt}(\mathcal{E}, H)$.

Rappelons l'exemple de la Figure C.1, il existe une relation entre la taille et la profondeur dans un arbre binaire valide. Autrement dit, lorsque la profondeur d'un arbre binaire valide est donnée, la taille de l'arbre est dans un intervalle correspondant. En détail, pour la profondeur maximale H, la borne supérieure de la taille est $2^H - 1$. De plus, si H est la profondeur exacte, la borne inférieure de la taille est $2H + 1$. Rappelons que la taille d'un un arbre binaire valide ne peut être qu'un nombre impair à partir de 3.

Supposons que la *borne supérieure de la taille* est N, nous introduisons une variable booléenne m_j, qui est **vrai** *si et seulement si* au moins j noeuds sont utilisés pour construire l'arbre. Les idées principales pour les contraintes de relaxation de la taille de l'arbre sont proposées ci-dessous (les détails sont dans la Section 2.3) :

- *Au moins* 3 *noeuds sont utilisés pour construire l'arbre.*
- *Si au moins $j+2$ nœuds sont utilisés pour construire l'arbre, il doit utiliser au moins j nœuds.*
- *Pour chaque clause dure, supponsons que j est le noeud de plus grand index, il faut vérifier m_j est vrai (ou m_{j+1} si j est un nombre pair) avant de vérifier si la clause dure est satisfaite.*
- *Pour respecter la* ***borne inférieure de la taille*** *N_l, il faut mettre m_{N_l} à vrai, pour contraindre à utiliser au moins N_l noeuds.*

C.2.2 Expérimentations

Nous considerons trois expérimentations pour évaluer nos contributions :

- ***Experimentation 1**: Mettre en évidence le comportement d'overfitting des arbres de décision obtenus avec l'approche SAT de la littérature.*
- ***Experimentation 2**: Evaluer les performances de prédiction entre notre modèle MaxSAT avec des méthodes de l'état de l'art : une méthode heuristique (**CART** [Breiman et al. 1984]) et une méthode exacte (**DL8.5** [Aglin et al. 2020]).*
- ***Experimentation 3**: Evaluer les performances de prédiction entre notre modèle MaxSAT initial et son integration dans AdaBoost [Freund & Schapire 1997].*

Nos experimentations portent sur les jeux de données de **CP4IM**[1]. Ces jeux de données sont binarisés avec l'encodage "one-hot" classique. Nous avons exécuté toutes les experimentations sur un cluster utilisant un processeur Xeon E5-2695 v3@2.30GHz exécutant xUbuntu 16.04.6LTS. Le solver MaxSAT utilisé est `Loandra` [Berg *et al.* 2019].

Les détails sur les protocoles expérimentaux et les résultats sont dans la Section 2.4. Nous résumons ci-après les observations principales.

La première observation issue de l'*Experimentation 1* est la vérification de l'existence du phénomène d'overfitting pour des arbres de décision optimaux obtenus avec l'approche SAT de la littérature. Des résultats détaillés sont dans l'Appendix A.

La deuxième observation issue de l'*Experimentation 2* est que notre modèle MaxSAT obtient une performance de prédiction compétitive par rapport aux méthodes de l'état de l'art (heuristique et exacte). Bien que notre modèle MaxSAT ne peut pas toujours obtenir la solution optimale dans le temps limite fixé (15 minutes), la solution trouvée est proche de la solution optimale trouvée par la méthode exacte *DL8.5* de la littérature. Par ailleurs, *DL8.5* souffre de limitation de mémoire, alors que notre modèle MaxSAT bénéficie d'une meilleure évolutivité. Les résultats détaillés sont présentés dans les Table 2.8 et 2.9 en Section 2.4.

La dernière observation issue de l'*Experimentation 3* est que l'integration de notre modèle MaxSAT dans AdaBoost permet d'améliorer la performance de prédiction du modèle MaxSAT initial. Pour quelques jeux de données, l'amélioration de la précision augmente de plus de 10%. Les résultats détaillés sont présentés dans la Table 2.10 en Section 2.5.

[1] `https://dtai.cs.kuleuven.be/CP4IM/datasets/`

C.3 Diagrammes de décision binaires optimaux par MaxSAT

Les diagrammes de décision binaires (BDD) sont un autre modèle d'apprentissage interprétable en apprentissage supervisé. En particulier, ils peuvent être utilisés pour la classification binaire avec un des données caractérisées par des attributs binaires. En tant que représentation compacte des fonctions booléennes, les diagrammes de décision binaires sont largement étudiés dans la conception de circuits numériques, la vérification de modèles ou la représentation des connaissances [Akers 1978, Moret 1982, Bryant 1986, Knuth 2009]. La Figure C.2 fournit un exemple de diagramme de décision binaire de profondeur 3 pour répresenter la fonction booléenne $g(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)$. Dans cet exemple, la séquence de variables booléennes choisie est $[x_1, x_2, x_3]$ et la profondeur du BDD est égale à la longueur de la séquence de variables booléennes.

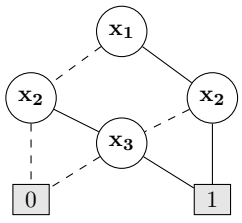


Figure C.2: Un exemple illustrant le BDD de profondeur 3.

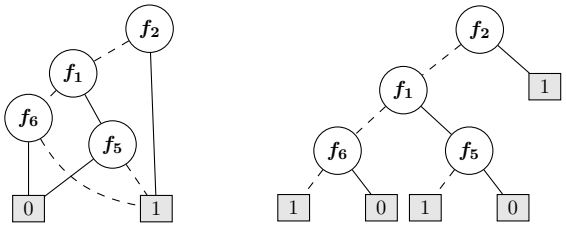


Figure C.3: Le BDD avec accuracy parfait pour les données dans Table 1.2 (à gauche), et l'arbre de décision correspondant (à droite).

Les diagrammes de décision binaire permettent d'éviter le problème de *réplication* et le problème de *fragmentation* dans la classification binaire [Oliver 1992, Kohavi 1994], qui sont deux défauts majeurs dont souffrent les arbres de décision [Matheus & Rendell 1989, Pagallo & Haussler 1990, Rokach & Maimon 2014]. Le problème de *réplication* apparaît lorsque deux sous-arbres identiques se trouvent dans l'arbre de décision. Un exemple illustratif est dans la Figure C.3, qui montre le BDD est plus compact que l'arbre de décision correspondant pour éviter le problème de réplication. Le problème de *fragmentation* apparaît lorsque seuls quelques exemples sont associés aux noeuds feuilles. Pour éviter ce problème, [Kohavi 1994] a proposé une post-traitement de fusion de sous-arbres compatibles. Des détails sont fournis dans la Section 1.3.2.

Les avantages des diagrammes de décision binaires justifient leur possible substitution aux arbres de décision dans l'apprentissage interprétable, malgré qu'ils ne suscitent pas autant d'intérêt que les arbres de décision. A notre connaissance, [Cabodi *et al.* 2021] est la seule méthode exacte récente pour apprendre des diagrammes de décision binaires optimaux avant notre recherche. L'objectif de cette approche est d'apprendre des diagrammes de décision binaires optimaux de plus petites *tailles* (en nombre de noeuds) avec la précision parfaite, ce qui conduit à deux inconvénients. Le premier est le possible overfitting en raison de l'objectif de précision

parfaite. L'autre est le manque de controle dans la profondeur du diagramme de décision binaire appris, pouvant produire des diagrammes de petite taille mais de grande profondeur.

Pour éviter ces inconvénients, nous considérons une nouvelle cible consistant à apprendre un diagramme de décision binaire de profondeur limitée qui optimise la précision. Cette cible s'exprime comme le *problème d'optimisation* suivant :

- $P^*_{bdd}(\mathcal{E}, H)$*: Pour un ensemble d'exemples* $\mathcal{E}$ *donné, trouver un diagramme de décision binaire avec la profondeur* H*, qui maximise le nombre d'exemples de* $\mathcal{E}$ *correctement classifiés.*

Inspirés par la méthodologie de résolution de nos recherches précédentes $P^*_{dt}(\mathcal{E}, H)$, nous introduisons d'abord un modèle SAT pour trouver le diagramme de décision binaire nécessitant le plus petit nombre d'attributs (la profondeur d'un BDD est égale au nombre d'attributs utilisés pour l'apprentissage) pour classifier correctement tous les exemples, qui est décrit comme le *problème décision* suivant:

- $P_{bdd}(\mathcal{E}, H)$*: Pour un ensemble d'exemples* $\mathcal{E}$ *donné, existe-t-il un diagramme de décision binaire avec la profondeur* H*, qui classife correctement tous les exemples de* $\mathcal{E}$ *?*

Ensuite, nous introduisons un modèle MaxSAT basé sur le modèle SAT pour résoudre le problème d'optimisation $P^*_{bdd}(\mathcal{E}, H)$, en utilisant la même technique que celle mise en oeuvre pour notre modèle MaxSAT apprenant des arbres de décision.

C.3.1 Modèle SAT et MaxSAT proposé

C.3.1.1 Une proposition essentielle

Avant d'introduire le modèle SAT ou MaxSAT proposé, nous en expliquons le principe qui est basé sur proposition essentielle vient de [Knuth 2009]. Ctte proposition présente comment construire un diagramme de décision binaire en utilisant sa table de vérité. Des détails sont dans la Section 3.2.

Nous présentons un exemple pour illustrer l'idée principale. Pour la fonction booléenne $g_1(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) = (\mathrm{x}_1 \vee \mathrm{x}_2 \vee \mathrm{x}_3) \wedge (\neg \mathrm{x}_1 \vee \mathrm{x}_2 \vee \neg \mathrm{x}_3) \wedge (\neg \mathrm{x}_1 \vee \neg \mathrm{x}_2 \vee \neg \mathrm{x}_3)$, la table de vérité correspondant est 01111010. Avec la séquence de variables booléennees $[\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3]$, la proposition nous permet de construire le diagramme de décision binaire dans la partie à droite en Figure C.4.

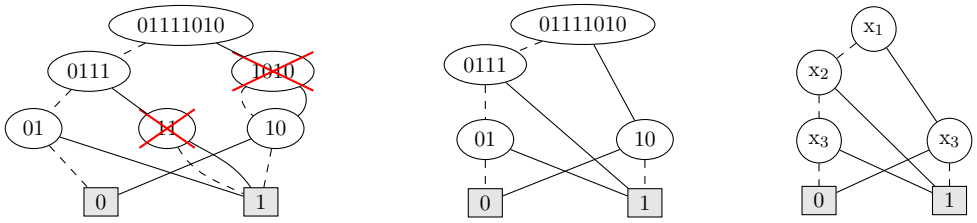


Figure C.4: Le BDD pour la fonction booléenne $g_1(\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3)$.

C.3.1.2 Modèle SAT proposé pour $P_{bdd}(\mathcal{E}, H)$

En s'appuyant sur la proposition précédente, un diagramme de décision binaire de profondeur H peut être construit par la combinaison d'une suite de variables booléennes de taille H : $[\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{H}]$, et d'une table de vérité associée à une fonction booléenne. Pour la classification binaire, nous cherchons ainsi à trouver une séquence d'attributs binaires de taille H qui mappe un-par-un la séquence de variables booléennes. En résumé, pour résoudre le problème de décision $P_{bdd}(\mathcal{E}, H)$, il faut trouver une séquence d'attributs binaires de taille H, et une table de vérité associée à une fonction booléenne qui classifie correctement tous exemples de $\mathcal{E}$. Le modèle SAT proposé s'appuie sur *deux familles* de contraintes:

- ***Partie 1**: Contraintes pour relier les attributs du jeu de données à la séquence d'attributs de taille H.*
- ***Partie 2**: Contraintes pour générer la table de vérité, qui classifie correctement tous les exemples de $\mathcal{E}$ avec la séquence d'attributs choisi.*

Pour encoder ces contraintes, nous proposons *trois ensembles* de variables booléennees. La variable a_r^i est **vrai** *si et seulement si* l'attribut f_r est relié à la i-ème position de la séquence d'attributs. La variable c_j est **vrai** *si et seulement si* la j-ème valeur dans la table de vérité est 1. La variable d_i^q est **vrai** *si et seulement si* pour l'exemple e_q, la valeur de i-ème attribut dans la séquence d'attributs est 1.

Pour encoder les contraintes de *Partie 1*, les idées principales sont données ci-dessous (les détails sont dans la Section 3.3.1):

- *Chaque attribut f_r peut être relié au plus une fois.*
- *Exactement un attribut est relié à chaque position de la séquence d'attributs.*
- *Pour éviter que la racine fasse une scission inutile, il faut garantir que la table de vérité générée est un bead (la partie gauche et partie droite sont différentes).*

Pour encoder les contraintes de *Partie 2*, d'abord, nous proposons des contraintes pour relier la variable d_i^q et la variable a_r^i par la valeur d'attribut f_r d'exemple e_q. Nous notons $\sigma(q, r)$ la valeur de l'attribut f_r pour un exemple e_q, les contraintes proposées sont les suivantes:

$$\begin{aligned} a_r^i &\rightarrow d_i^q \qquad \text{si } \sigma(q,r) = 1 \\ a_r^i &\rightarrow \neg d_i^q \qquad \text{si } \sigma(q,r) = 0 \end{aligned} \tag{C.3}$$

Ensuite, pour classifier correctement tous les exemples, l'idée principale est pour chaque exemple positif (negatif) suite à une affectation de la séquence d'attributs, ils conduisent à une valeur positive (negative) dans la table de vérité. Les 2^H contraintes permettant de garantir qu l'exemple positif e_q soit correctement classifié sont données ci-dessous:

$$\begin{aligned}
\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge \neg d_H^q &\rightarrow c_1 \\
\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge d_H^q &\rightarrow c_2 \\
&\cdots \\
d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge \neg d_H^q &\rightarrow c_{2^H-1} \\
d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge d_H^q &\rightarrow c_{2^H}
\end{aligned} \tag{C.4}$$

De la même manière, les 2^H contraintes pour garantir que l'exemple négatif e_q soit correctement classifié sont données ci-dessous:

$$\begin{aligned}
\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge \neg d_H^q &\rightarrow \neg c_1 \\
\neg d_1^q \wedge \neg d_2^q \wedge \cdots \wedge \neg d_{H-1}^q \wedge d_H^q &\rightarrow \neg c_2 \\
&\cdots \\
d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge \neg d_H^q &\rightarrow \neg c_{2^H-1} \\
d_1^q \wedge d_2^q \wedge \cdots \wedge d_{H-1}^q \wedge d_H^q &\rightarrow \neg c_{2^H}
\end{aligned} \tag{C.5}$$

C.3.1.3 Modèle MaxSAT pour $P^*_{bdd}(\mathcal{E}, H)$

La technique pour transformer le modèle SAT des diagrammes de décision binaires vers un modèle MaxSAT est simple. Le principe est de conserver les *contraintes structurelles (Partie 1)* comme des ***clauses dures*** et les *contraintes de classification (Partie 2 sauf Constrainte C.3)* comme des ***clauses souples***. La raison est que pour tout exemple e_q, le nombre de *clauses souples satisfaites* associées à e_q est soit $\mathbf{2^H}$, ce que indique que e_q est *classifié correctement*, soit $\mathbf{2^H - 1}$, ce que indique que e_q est *mal classifié*.

Par conséquent, l'objectif de maximiser le nombre de clauses souples satisfaites équivaut à maximiser le nombre d'exemples correctement classifiés, ce qui résout le problème d'optimisation $P^*_{bdd}(\mathcal{E}, H)$.

C.3.2 Expérimentations

Nous considerons deux expérimentations pour évaluer nos contributions:

- ***Experimentation 1****: Comparer les performances de prédiction entre notre modèle MaxSAT avec la méthode heuristique* ***OODG*** *[Kohavi & Li 1995].*
- ***Experimentation 2****: Comparer les performances de prédiction, les tailles de modèles, et les tailles d'encodage entre notre modèle MaxSAT pour les BDD avec le modèle MaxSAT que nous avons proposé pour les arbres de décision [Hu et al. 2020]*).

Comme précédemment, nos experimentations portent sur les jeux de données binarisés de **CP4IM**. Toutes les experimentations sont exécutées sur un cluster utilisant un processeur Xeon E5-2695 v3@2.30GHz fonctionnant sous xUbuntu 16.04.6LTS. Le solver MaxSAT utilisé est `Loandra` [Berg *et al.* 2019]. Les détails

sur les protocoles expérimentaux et les résultats sont fournis dans la Section 3.4. Nous résumons ci-après les observations principales.

La première observation issue de l'*Experimentation 1* est que notre modèle MaxSAT obtient toujours de meilleures performances de prédiction par rapport à la méthode heuristique *OODG*.

La deuxième observation issue de l'*Experimenatation 2* est que notre modèle MaxSAT pour les BDD obtient des performances de prédiction compétives par rapport au modèle MaxSAT pour les arbres de décision. En complément, notre modèle MaxSAT pour les BDD obtient un encodage plus léger que l'encodage pour les arbres de décision, et le BDD trouvé par notre modèle MaxSAT est plus compact en taille que l'arbre de décision trouvé par notre modèle MaxSAT générant des arbres de décision. Les résultats détaillés sont dans la Table 3.11 en Section 3.4.

Bibliography

[Aglin *et al.* 2020] Gaël Aglin, Siegfried Nijssen and Pierre Schaus. *Learning Optimal Decision Trees Using Caching Branch-and-Bound Search.* In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 3146–3153. AAAI Press, 2020. (Cited on pages 31, 53 and 121.)

[Akers 1978] Akers. *Binary Decision Diagrams.* IEEE Transactions on Computers, vol. C-27, no. 6, pages 509–516, 1978. (Cited on pages 17 and 122.)

[Alos *et al.* 2021] Josep Alos, Carlos Ansótegui and Eduard Torres. *Learning Optimal Decision Trees Using MaxSAT.* CoRR, vol. abs/2110.13854, 2021. (Cited on page 31.)

[Audemard & Simon 2009] Gilles Audemard and Laurent Simon. *Predicting Learnt Clauses Quality in Modern SAT Solvers.* In Craig Boutilier, editor, IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009, pages 399–404, 2009. (Cited on page 7.)

[Audemard & Simon 2018] Gilles Audemard and Laurent Simon. *On the Glucose SAT Solver.* Int. J. Artif. Intell. Tools, vol. 27, no. 1, pages 1840001:1–1840001:25, 2018. (Cited on pages 7 and 62.)

[Avellaneda 2020] Florent Avellaneda. *Efficient Inference of Optimal Decision Trees.* In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 3195–3202. AAAI Press, 2020. (Cited on pages 2, 29, 30, 40 and 117.)

[Bacchus *et al.* 2021a] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo and Ruben Martins, editors. Maxsat evaluation 2021: Solver and benchmark descriptions. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2021. (Cited on pages 4 and 60.)

[Bacchus *et al.* 2021b] Fahiem Bacchus, Matti Järvisalo and Ruben Martins. *Maximum Satisfiabiliy.* In Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh, editors, Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021. (Cited on page 8.)

[Bacchus *et al.* 2022] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, Ruben Martins and Andreas Niskanen, editors. Maxsat evaluation 2022: Solver and benchmark descriptions. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2022. (Cited on pages 4 and 60.)

[Bengio *et al.* 2021] Yoshua Bengio, Andrea Lodi and Antoine Prouvost. *Machine learning for combinatorial optimization: A methodological tour d'horizon.* Eur. J. Oper. Res., vol. 290, no. 2, pages 405–421, 2021. (Cited on page 101.)

[Berg *et al.* 2019] Jeremias Berg, Emir Demirovic and Peter J. Stuckey. *Core-Boosted Linear Search for Incomplete MaxSAT.* In Louis-Martin Rousseau and Kostas Stergiou, editors, Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings, volume 11494 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2019. (Cited on pages 10, 55, 61, 87, 121 and 125.)

[Bergman *et al.* 2016] David Bergman, André A. Ciré, Willem-Jan van Hoeve and John N. Hooker. Decision diagrams for optimization. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, 2016. (Cited on page 32.)

[Berre & Parrain 2010] Daniel Le Berre and Anne Parrain. *The Sat4j library, release 2.2.* J. Satisf. Boolean Model. Comput., vol. 7, no. 2-3, pages 59–6, 2010. (Cited on page 9.)

[Bertsimas & Dunn 2017] Dimitris Bertsimas and Jack Dunn. *Optimal classification trees.* Mach. Learn., vol. 106, no. 7, pages 1039–1082, 2017. (Cited on page 30.)

[Bessiere *et al.* 2009] Christian Bessiere, Emmanuel Hebrard and Barry O'Sullivan. *Minimising Decision Tree Size as Combinatorial Optimisation.* In Ian P. Gent, editor, Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings, volume 5732 of *Lecture Notes in Computer Science*, pages 173–187. Springer, 2009. (Cited on pages 2, 29, 30, 40, 46 and 117.)

[Biere *et al.* 2020] Armin Biere, Katalin Fazekas, Mathias Fleury and Maximillian Heisinger. *CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020.* In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo and Martin Suda, editors, Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020. (Cited on page 85.)

[Biere *et al.* 2021] Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh, editors. Handbook of satisfiability - second edition, volume 336 of *Fron-*

tiers in Artificial Intelligence and Applications. IOS Press, 2021. (Cited on page 6.)

[Breiman *et al.* 1984] Leo Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone. Classification and regression trees. Wadsworth, 1984. (Cited on pages 16, 27, 53, 95 and 121.)

[Breiman 1996] Leo Breiman. *Bagging Predictors*. Mach. Learn., vol. 24, no. 2, pages 123–140, 1996. (Cited on page 24.)

[Breiman 2001] Leo Breiman. *Random Forests*. Mach. Learn., vol. 45, no. 1, pages 5–32, 2001. (Cited on page 24.)

[Brodley & Utgoff 1995] Carla E. Brodley and Paul E. Utgoff. *Multivariate Decision Trees*. Mach. Learn., vol. 19, no. 1, pages 45–77, 1995. (Cited on pages 16 and 100.)

[Brown *et al.* 2005] Gavin Brown, Jeremy L. Wyatt, Rachel Harris and Xin Yao. *Diversity creation methods: a survey and categorisation*. Inf. Fusion, vol. 6, no. 1, pages 5–20, 2005. (Cited on page 22.)

[Bryant 1986] Randal E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Trans. Computers, vol. 35, no. 8, pages 677–691, 1986. (Cited on pages 17, 32 and 122.)

[Cabodi *et al.* 2021] Gianpiero Cabodi, Paolo E. Camurati, Alexey Ignatiev, João Marques-Silva, Marco Palena and Paolo Pasini. *Optimizing Binary Decision Diagrams for Interpretable Machine Learning Classification*. In Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021, pages 1122–1125. IEEE, 2021. (Cited on pages 3, 36, 37, 68 and 122.)

[Cai & Lei 2020] Shaowei Cai and Zhendong Lei. *Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability*. Artif. Intell., vol. 287, page 103354, 2020. (Cited on page 61.)

[Chen & Guestrin 2016] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen and Rajeev Rastogi, editors, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pages 785–794. ACM, 2016. (Cited on pages 14 and 23.)

[Chu *et al.* 2022] Yi Chu, Shaowei Cai, Zhendong Lei and Xiang He. *NuWLS-c: Solver Description*. MaxSAT Evaluation 2022, page 28, 2022. (Cited on page 61.)

[Cover & Thomas 2001] Thomas M. Cover and Joy A. Thomas. Elements of information theory. Wiley, 2001. (Cited on page 34.)

[Davies & Bacchus 2011] Jessica Davies and Fahiem Bacchus. *Solving MAXSAT by Solving a Sequence of Simpler SAT Instances*. In Jimmy Ho-Man Lee, editor, Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. (Cited on page 9.)

[Davies & Bacchus 2013] Jessica Davies and Fahiem Bacchus. *Exploiting the Power of mip Solvers in maxsat*. In Matti Järvisalo and Allen Van Gelder, editors, Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013. (Cited on page 9.)

[Davis *et al.* 1962] Martin Davis, George Logemann and Donald W. Loveland. *A machine program for theorem-proving*. Commun. ACM, vol. 5, no. 7, pages 394–397, 1962. (Cited on page 6.)

[Demirovic *et al.* 2022] Emir Demirovic, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao and Peter J. Stuckey. *MurTree: Optimal Decision Trees via Dynamic Programming and Search*. J. Mach. Learn. Res., vol. 23, pages 26:1–26:47, 2022. (Cited on page 31.)

[Devriendt 2021] Jo Devriendt. *Exact: evaluating a pseudo-Boolean solver on MaxSAT problems*. MaxSAT Evaluation 2021, page 12, 2021. (Cited on page 61.)

[Dietterich 2000a] Thomas G. Dietterich. *Ensemble Methods in Machine Learning*. In Josef Kittler and Fabio Roli, editors, Multiple Classifier Systems, First International Workshop, MCS 2000, Cagliari, Italy, June 21-23, 2000, Proceedings, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000. (Cited on page 21.)

[Dietterich 2000b] Thomas G. Dietterich. *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization*. Mach. Learn., vol. 40, no. 2, pages 139–157, 2000. (Cited on page 22.)

[Eén & Sörensson 2003] Niklas Eén and Niklas Sörensson. *An Extensible SAT-solver*. In Enrico Giunchiglia and Armando Tacchella, editors, Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. (Cited on pages 6 and 7.)

[Efron & Tibshirani 1993] Bradley Efron and Robert Tibshirani. An introduction to the bootstrap. Springer, 1993. (Cited on pages 13 and 24.)

[Florio *et al.* 2022] Alexandre M. Florio, Pedro Martins, Maximilian Schiffer, Thiago Serra and Thibaut Vidal. *Optimal Decision Diagrams for Classification.* CoRR, vol. abs/2205.14500, 2022. (Cited on pages 36, 37, 38, 68 and 100.)

[Freund & Schapire 1997] Yoav Freund and Robert E. Schapire. *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting.* J. Comput. Syst. Sci., vol. 55, no. 1, pages 119–139, 1997. (Cited on pages 23 and 121.)

[Friedman *et al.* 2000] Jerome Friedman, Trevor Hastie and Robert Tibshirani. *Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors).* The annals of statistics, vol. 28, no. 2, pages 337–407, 2000. (Cited on page 23.)

[Friedman 2001] Jerome H Friedman. *Greedy function approximation: a gradient boosting machine.* Annals of statistics, pages 1189–1232, 2001. (Cited on page 23.)

[Friedman 2002] Jerome H Friedman. *Stochastic gradient boosting.* Computational statistics & data analysis, vol. 38, no. 4, pages 367–378, 2002. (Cited on page 23.)

[Geurts *et al.* 2006] Pierre Geurts, Damien Ernst and Louis Wehenkel. *Extremely randomized trees.* Mach. Learn., vol. 63, no. 1, pages 3–42, 2006. (Cited on page 25.)

[Gomes *et al.* 1998] Carla P. Gomes, Bart Selman and Henry A. Kautz. *Boosting Combinatorial Search Through Randomization.* In Jack Mostow and Chuck Rich, editors, Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA, pages 431–437. AAAI Press / The MIT Press, 1998. (Cited on page 7.)

[Goodfellow *et al.* 2014] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville and Yoshua Bengio. *Generative Adversarial Nets.* In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence and Kilian Q. Weinberger, editors, Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, pages 2672–2680, 2014. (Cited on page 14.)

[Guidotti *et al.* 2018] Riccardo Guidotti, Anna Monreale, Franco Turini, Dino Pedreschi and Fosca Giannotti. *A Survey Of Methods For Explaining Black Box Models.* CoRR, vol. abs/1802.01933, 2018. (Cited on pages 1 and 115.)

[Hansen & Salamon 1990] Lars Kai Hansen and Peter Salamon. *Neural Network Ensembles*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 12, no. 10, pages 993–1001, 1990. (Cited on page 22.)

[Hastie *et al.* 2009] Trevor Hastie, Robert Tibshirani and Jerome H. Friedman. The elements of statistical learning: Data mining, inference, and prediction, 2nd edition. Springer Series in Statistics. Springer, 2009. (Cited on page 10.)

[He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society, 2016. (Cited on page 14.)

[Heras *et al.* 2008] Federico Heras, Javier Larrosa and Albert Oliveras. *MiniMaxSAT: An Efficient Weighted Max-SAT solver*. J. Artif. Intell. Res., vol. 31, pages 1–32, 2008. (Cited on page 9.)

[Ho 1995] Tin Kam Ho. *Random decision forests*. In Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I, pages 278–282. IEEE Computer Society, 1995. (Cited on page 25.)

[Hu *et al.* 2019] Xiyang Hu, Cynthia Rudin and Margo I. Seltzer. *Optimal Sparse Decision Trees*. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox and Roman Garnett, editors, Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 7265–7273, 2019. (Cited on page 31.)

[Hu *et al.* 2020] Hao Hu, Mohamed Siala, Emmanuel Hebrard and Marie-José Huguet. *Learning Optimal Decision Trees with MaxSAT and its Integration in AdaBoost*. In Christian Bessiere, editor, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pages 1170–1176. ijcai.org, 2020. (Cited on pages 4, 29, 31, 39 and 125.)

[Hu *et al.* 2021] Hao Hu, Emmanuel Hebrard, Marie-José Huguet and Mohamed Siala. *Description of Benchmarks on Learning Optimal Decision Trees and Boosted Trees*. MaxSAT Evaluation 2021, page 39, 2021. (Cited on pages 4 and 60.)

[Hu *et al.* 2022] Hao Hu, Marie-José Huguet and Mohamed Siala. *Optimizing Binary Decision Diagrams with MaxSAT for Classification*. In Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence,

EAAI 2022 Virtual Event, February 22 - March 1, 2022, pages 3767–3775. AAAI Press, 2022. (Cited on pages 4, 36 and 67.)

[Hyafil & Rivest 1976] Laurent Hyafil and Ronald L. Rivest. *Constructing Optimal Binary Decision Trees is NP-Complete.* Inf. Process. Lett., vol. 5, no. 1, pages 15–17, 1976. (Cited on page 29.)

[Ignatiev *et al.* 2018] Alexey Ignatiev, Filipe Pereira, Nina Narodytska and João Marques-Silva. *A SAT-Based Approach to Learn Explainable Decision Sets.* In Didier Galmiche, Stephan Schulz and Roberto Sebastiani, editors, Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, volume 10900 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018. (Cited on page 20.)

[Ignatiev *et al.* 2019] Alexey Ignatiev, António Morgado and João Marques-Silva. *RC2: an Efficient MaxSAT Solver.* J. Satisf. Boolean Model. Comput., vol. 11, no. 1, pages 53–64, 2019. (Cited on pages 9 and 54.)

[Janota & Morgado 2020] Mikolás Janota and António Morgado. *SAT-Based Encodings for Optimal Decision Trees with Explicit Paths.* In Luca Pulina and Martina Seidl, editors, Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings, volume 12178 of *Lecture Notes in Computer Science*, pages 501–518. Springer, 2020. (Cited on pages 2, 29, 30, 40 and 117.)

[Joshi *et al.* 2021] Saurabh Joshi, Prateek Kumar, Sukrut Rao and Ruben Martins. *Open-WBO-Inc in MaxSAT Evaluation 2020.* MaxSAT Evaluation 2020, page 26, 2021. (Cited on page 64.)

[Ke *et al.* 2017] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye and Tie-Yan Liu. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 3146–3154, 2017. (Cited on page 14.)

[Kim *et al.* 2016] Been Kim, Oluwasanmi Koyejo and Rajiv Khanna. *Examples are not enough, learn to criticize! Criticism for Interpretability.* In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon and Roman Garnett, editors, Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, pages 2280–2288, 2016. (Cited on page 14.)

[Knuth 2009] Donald E. Knuth. The art of computer programming, volume 4, fascicle 1: Bitwise tricks & techniques; binary decision diagrams. Addison-Wesley Professional, 12th édition, 2009. (Cited on pages 17, 67, 69, 122 and 123.)

[Kohavi & Li 1995] Ron Kohavi and Chia-Hsin Li. *Oblivious Decision Trees, Graphs, and Top-Down Pruning.* In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, pages 1071–1079. Morgan Kaufmann, 1995. (Cited on pages 19, 33, 84, 87 and 125.)

[Kohavi 1994] Ron Kohavi. *Bottom-Up Induction of Oblivious Read-Once Decision Graphs.* In Francesco Bergadano and Luc De Raedt, editors, Machine Learning: ECML-94, European Conference on Machine Learning, Catania, Italy, April 6-8, 1994, Proceedings, volume 784 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 1994. (Cited on pages 3, 19, 32, 33 and 122.)

[Kohavi 1995] Ron Kohavi. *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.* In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, pages 1137–1145. Morgan Kaufmann, 1995. (Cited on page 13.)

[Korhonen *et al.* 2017] Tuukka Korhonen, Jeremias Berg, Paul Saikko and Matti Järvisalo. *MaxPre: An Extended MaxSAT Preprocessor.* In Serge Gaspers and Toby Walsh, editors, Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017. (Cited on page 62.)

[Koshimura *et al.* 2012] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita and Ryuzo Hasegawa. *QMaxSAT: A Partial Max-SAT Solver.* J. Satisf. Boolean Model. Comput., vol. 8, no. 1/2, pages 95–100, 2012. (Cited on page 9.)

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks.* In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou and Kilian Q. Weinberger, editors, Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, pages 1106–1114, 2012. (Cited on page 14.)

[Kügel 2010] Adrian Kügel. *Improved Exact Solver for the Weighted MAX-SAT Problem.* In Daniel Le Berre, editor, POS-10. Pragmatics of SAT, Edinburgh,

UK, July 10, 2010, volume 8 of *EPiC Series in Computing*, pages 15–27. EasyChair, 2010. (Cited on page 9.)

[Lakkaraju *et al.* 2016] Himabindu Lakkaraju, Stephen H. Bach and Jure Leskovec. *Interpretable Decision Sets: A Joint Framework for Description and Prediction.* In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen and Rajeev Rastogi, editors, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pages 1675–1684. ACM, 2016. (Cited on page 20.)

[Lam & Suen 1997] Louisa Lam and Ching Y. Suen. *Application of majority voting to pattern recognition: an analysis of its behavior and performance.* IEEE Trans. Syst. Man Cybern. Part A, vol. 27, no. 5, pages 553–568, 1997. (Cited on page 22.)

[Laugel *et al.* 2019] Thibault Laugel, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard and Marcin Detyniecki. *The Dangers of Post-hoc Interpretability: Unjustified Counterfactual Explanations.* In Sarit Kraus, editor, Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, pages 2801–2807. ijcai.org, 2019. (Cited on pages 1 and 115.)

[Lei *et al.* 2021] Zhendong Lei, Shaowei Cai, Fei Geng, Dongxu Wang, Yongrong Peng, Dongdong Wan, Yiping Deng and Pinyan Lu. *SATLike-c: Solver Description.* MaxSAT Evaluation 2021, page 19, 2021. (Cited on page 61.)

[Li & Manyà 2021] Chu Min Li and Felip Manyà. *MaxSAT, Hard and Soft Constraints.* In Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh, editors, Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 903–927. IOS Press, 2021. (Cited on page 8.)

[Li *et al.* 2007] Chu Min Li, Felip Manyà and Jordi Planes. *New Inference Rules for Max-SAT.* J. Artif. Intell. Res., vol. 30, pages 321–359, 2007. (Cited on page 9.)

[Li *et al.* 2021] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet and Kun He. *Combining Clause Learning and Branch and Bound for MaxSAT.* In Laurent D. Michel, editor, 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021, volume 210 of *LIPIcs*, pages 38:1–38:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on page 9.)

[Lübke & Schupp 2022] Ole Lübke and Sibylle Schupp. *noSAT-MaxSAT.* MaxSAT Evaluation 2022, page 29, 2022. (Cited on page 61.)

[Marques-Silva & Manquinho 2008] João Marques-Silva and Vasco M. Manquinho. *Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms.* In Hans Kleine Büning and Xishun Zhao, editors, Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings, volume 4996 of *Lecture Notes in Computer Science*, pages 225–230. Springer, 2008. (Cited on page 9.)

[Marques-Silva *et al.* 2021] João Marques-Silva, Inês Lynce and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers.* In Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh, editors, Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021. (Cited on page 7.)

[Martins *et al.* 2014] Ruben Martins, Vasco M. Manquinho and Inês Lynce. *Open-WBO: A Modular MaxSAT Solver,.* In Carsten Sinz and Uwe Egly, editors, Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014. (Cited on pages 9 and 61.)

[Martins *et al.* 2021] Ruben Martins, Norbert Manthey, Miguel Terra-Neves, Vasco Manquinho and Inês Lynce. *Open-WBO@ MaxSAT Evaluation 2020.* MaxSAT Evaluation 2020, page 24, 2021. (Cited on page 61.)

[Matheus & Rendell 1989] Christopher J. Matheus and Larry A. Rendell. *Constructive Induction On Decision Trees.* In N. S. Sridharan, editor, Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989, pages 645–650. Morgan Kaufmann, 1989. (Cited on pages 19 and 122.)

[Miller 2019] Tim Miller. *Explanation in artificial intelligence: Insights from the social sciences.* Artif. Intell., vol. 267, pages 1–38, 2019. (Cited on page 14.)

[Mitchell 1997] Tom M. Mitchell. Machine learning, international edition. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. (Cited on pages 10 and 12.)

[Molnar 2022] Christoph Molnar. Interpretable machine learning. 2 édition, 2022. (Cited on page 14.)

[Moret 1982] Bernard M. E. Moret. *Decision Trees and Diagrams.* ACM Comput. Surv., vol. 14, no. 4, pages 593–623, 1982. (Cited on pages 17 and 122.)

[Moskewicz *et al.* 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang and Sharad Malik. *Chaff: Engineering an Efficient SAT*

Solver. In Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001, pages 530–535. ACM, 2001. (Cited on pages 6 and 7.)

[Murthy *et al.* 1993] Sreerama K. Murthy, Simon Kasif, Steven Salzberg and Richard Beigel. *OC1: A Randomized Induction of Oblique Decision Trees*. In Richard Fikes and Wendy G. Lehnert, editors, Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993, pages 322–327. AAAI Press / The MIT Press, 1993. (Cited on pages 16 and 100.)

[Nadel 2020] Alexander Nadel. *Polarity and Variable Selection Heuristics for SAT-Based Anytime MaxSAT*. J. Satisf. Boolean Model. Comput., vol. 12, no. 1, pages 17–22, 2020. (Cited on page 61.)

[Nadel 2022] Alexander Nadel. *Introducing Intel(R) SAT Solver*. In Kuldeep S. Meel and Ofer Strichman, editors, 25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel, volume 236 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on page 62.)

[Narodytska *et al.* 2018] Nina Narodytska, Alexey Ignatiev, Filipe Pereira and João Marques-Silva. *Learning Optimal Decision Trees with SAT*. In Jérôme Lang, editor, Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, pages 1362–1368. ijcai.org, 2018. (Cited on pages 2, 29, 30, 31, 36, 39, 40, 41, 42, 44, 45, 46, 54, 64, 68, 75 and 117.)

[Nijssen & Fromont 2007] Siegfried Nijssen and Élisa Fromont. *Mining optimal decision trees from itemset lattices*. In Pavel Berkhin, Rich Caruana and Xindong Wu, editors, Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007, pages 530–539. ACM, 2007. (Cited on page 31.)

[Oliver 1992] Jonathan Oliver. Decision graphs: an extension of decision trees. Citeseer, 1992. (Cited on pages 3, 19, 32 and 122.)

[Pagallo & Haussler 1990] Giulia Pagallo and David Haussler. *Boolean Feature Discovery in Empirical Learning*. Mach. Learn., vol. 5, pages 71–99, 1990. (Cited on pages 19 and 122.)

[Pedregosa *et al.* 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, vol. 12, pages 2825–2830, 2011. (Cited on page 55.)

[Quinlan 1986] J. Ross Quinlan. *Induction of Decision Trees.* Mach. Learn., vol. 1, no. 1, pages 81–106, 1986. (Cited on pages 25 and 26.)

[Quinlan 1993] J. Ross Quinlan. C4.5: programs for machine learning. Morgan Kaufmann, 1993. (Cited on pages 15, 16 and 25.)

[Reisch *et al.* 2020] Julian Reisch, Peter Großmann and Natalia Kliewer. *Stable Resolving - A Randomized Local Search Heuristic for MaxSAT.* In Ute Schmid, Franziska Klügl and Diedrich Wolter, editors, KI 2020: Advances in Artificial Intelligence - 43rd German Conference on AI, Bamberg, Germany, September 21-25, 2020, Proceedings, volume 12325 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2020. (Cited on page 61.)

[Rivest 1987] Ronald L. Rivest. *Learning Decision Lists.* Mach. Learn., vol. 2, no. 3, pages 229–246, 1987. (Cited on pages 19 and 20.)

[Rodríguez *et al.* 2006] Juan José Rodríguez, Ludmila I. Kuncheva and Carlos J. Alonso. *Rotation Forest: A New Classifier Ensemble Method.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 28, no. 10, pages 1619–1630, 2006. (Cited on page 25.)

[Rokach & Maimon 2014] Lior Rokach and Oded Maimon. Data mining with decision trees: Theory and applications. World Scientific Publishing Co., Inc., USA, 2nd édition, 2014. (Cited on pages 19 and 122.)

[Rokach 2016] Lior Rokach. *Decision forest: Twenty years of research.* Inf. Fusion, vol. 27, pages 111–125, 2016. (Cited on page 22.)

[Rudin *et al.* 2021] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova and Chudi Zhong. *Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges.* CoRR, vol. abs/2103.11251, 2021. (Cited on pages 1 and 115.)

[Rudin 2019] Cynthia Rudin. *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.* Nat. Mach. Intell., vol. 1, no. 5, pages 206–215, 2019. (Cited on pages 1, 14 and 115.)

[Russell & Norvig 2020] Stuart J. Russell and Peter Norvig. Artificial intelligence: A modern approach (4th edition). Pearson, 2020. (Cited on page 10.)

[Saikko *et al.* 2016] Paul Saikko, Jeremias Berg and Matti Järvisalo. *LMHS: A SAT-IP Hybrid MaxSAT Solver.* In Nadia Creignou and Daniel Le Berre, editors, Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016. (Cited on page 9.)

[Schidler & Szeider 2021] André Schidler and Stefan Szeider. *SAT-based Decision Tree Learning for Large Data Sets*. In Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 3904–3912. AAAI Press, 2021. (Cited on page 31.)

[Shannon 1938] C. E. Shannon. *A symbolic analysis of relay and switching circuits*. Electrical Engineering, vol. 57, no. 12, pages 713–723, 1938. (Cited on page 17.)

[Shannon 1948] Claude E. Shannon. *A mathematical theory of communication*. Bell Syst. Tech. J., vol. 27, no. 3, pages 379–423, 1948. (Cited on page 25.)

[Shati *et al.* 2021] Pouya Shati, Eldan Cohen and Sheila A. McIlraith. *SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features*. In Laurent D. Michel, editor, 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021, volume 210 of *LIPIcs*, pages 50:1–50:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on page 31.)

[Silva & Sakallah 1996] João P. Marques Silva and Karem A. Sakallah. *GRASP - a new search algorithm for satisfiability*. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996, pages 220–227. IEEE Computer Society / ACM, 1996. (Cited on page 6.)

[Silva & Sakallah 1999] João P. Marques Silva and Karem A. Sakallah. *GRASP: A Search Algorithm for Propositional Satisfiability*. IEEE Trans. Computers, vol. 48, no. 5, pages 506–521, 1999. (Cited on page 6.)

[Silva 1999] João P. Marques Silva. *The Impact of Branching Heuristics in Propositional Satisfiability Algorithms*. In Pedro Barahona and José Júlio Alferes, editors, Progress in Artificial Intelligence, 9th Portuguese Conference on Artificial Intelligence, EPIA '99, Évora, Portugal, September 21-24, 1999, Proceedings, volume 1695 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1999. (Cited on page 6.)

[Sinz 2005] Carsten Sinz. *Towards an Optimal CNF Encoding of Boolean Cardinality Constraints*. In Peter van Beek, editor, Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005. (Cited on pages 41 and 69.)

[Tang *et al.* 2006] E. Ke Tang, Ponnuthurai N. Suganthan and Xin Yao. *An analysis of diversity measures.* Mach. Learn., vol. 65, no. 1, pages 247–271, 2006. (Cited on page 22.)

[Utgoff *et al.* 1997] Paul E. Utgoff, Neil C. Berkman and Jeffery A. Clouse. *Decision Tree Induction Based on Efficient Tree Restructuring.* Mach. Learn., vol. 29, no. 1, pages 5–44, 1997. (Cited on pages 16, 40 and 100.)

[Utgoff 1988] Paul E. Utgoff. *ID5: An Incremental ID3.* In John E. Laird, editor, Machine Learning, Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, Michigan, USA, June 12-14, 1988, pages 107–120. Morgan Kaufmann, 1988. (Cited on page 16.)

[Vaswani *et al.* 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. *Attention is All you Need.* In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan and Roman Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017. (Cited on page 14.)

[Verhaeghe *et al.* 2020] Hélène Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper and Pierre Schaus. *Learning optimal decision trees using constraint programming.* Constraints An Int. J., vol. 25, no. 3-4, pages 226–250, 2020. (Cited on page 30.)

[Verwer & Zhang 2017] Sicco Verwer and Yingqian Zhang. *Learning Decision Trees with Flexible Constraints and Objectives Using Integer Optimization.* In Domenico Salvagnin and Michele Lombardi, editors, Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings, volume 10335 of *Lecture Notes in Computer Science*, pages 94–103. Springer, 2017. (Cited on page 30.)

[Verwer & Zhang 2019] Sicco Verwer and Yingqian Zhang. *Learning Optimal Classification Trees Using a Binary Linear Program Formulation.* In The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 1625–1632. AAAI Press, 2019. (Cited on pages 30 and 31.)

[Zabih & McAllester 1988] Ramin Zabih and David A. McAllester. *A Rearrangement Search Strategy for Determining Propositional Satisfiability.* In Howard E. Shrobe, Tom M. Mitchell and Reid G. Smith, editors, Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN,

USA, August 21-26, 1988, pages 155–160. AAAI Press / The MIT Press, 1988. (Cited on page 6.)

[Zheng *et al.* 2022a] Jiongzhi Zheng, Kun He, Zhuo Chen, Jianrong Zhou and Chu-Min Li. *Decision Tree based Hybrid Walking Strategies*. MaxSAT Evaluation 2022, page 24, 2022. (Cited on page 61.)

[Zheng *et al.* 2022b] Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chu-Min Li and Felip Manyà. *BandMaxSAT: A Local Search MaxSAT Solver with Multi-armed Bandit*. In Luc De Raedt, editor, Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022, pages 1901–1907. ijcai.org, 2022. (Cited on page 61.)

[Zhou & Feng 2017] Zhi-Hua Zhou and Ji Feng. *Deep Forest: Towards An Alternative to Deep Neural Networks*. In Carles Sierra, editor, Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, pages 3553–3559. ijcai.org, 2017. (Cited on page 14.)

[Zhou *et al.* 2021] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong and Wancai Zhang. *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. In Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 11106–11115. AAAI Press, 2021. (Cited on page 14.)

[Zhou 2012] Zhi-Hua Zhou. Ensemble methods: Foundations and algorithms. Chapman & Hall/CRC, 1st édition, 2012. (Cited on pages 10, 20 and 22.)

[Zhou 2021] Zhi-Hua Zhou. Machine learning. Springer, 2021. (Cited on pages 10 and 12.)

[Zhu & Shoaran 2021] Bingzhao Zhu and Mahsa Shoaran. *Tree in Tree: from Decision Trees to Decision Graphs*. Advances in Neural Information Processing Systems, vol. 34, pages 13707–13718, 2021. (Cited on page 32.)