Interpretable and Constrained Machine Learning via Combinatorial Optimization

by

Pouya Shati

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Department of Computer Science
University of Toronto

Interpretable and Constrained Machine Learning via Combinatorial Optimization

Pouya Shati
Doctor of Philosophy

Department of Computer Science
University of Toronto
2025

Abstract

Recent transformative advances in machine and deep learning have enabled the recognition of complex patterns from vast data, with or without human supervision. These advances have catalyzed the application of machine learning techniques to a diversity of problem settings, from healthcare to robotics. Unfortunately, sophisticated machine learning techniques can result in the construction of domain models that are challenging for humans to understand and in which it is difficult to enforce user-specified constraints. This lack of interpretability, as well as the inability to impose further constraints, has become a major obstacle to human trust in machine learning models and a challenge to the broad adoption of machine learning-based systems.

While one of the goals of artificial intelligence is, arguably, to extend its reasoning beyond the cognitive capabilities of humankind, such a pursuit should not disregard human-compatibility. Interpretable machine learning aims to develop human-understandable models and to explain so-called black-box models. Constrained machine learning addresses the problem by enforcing expert knowledge and formal requirements on solutions.

In this dissertation we introduce methods to produce machine learning models/artifacts that are interpretable and amenable to human-specified constraints. We do so through the exploitation of techniques for solving combinatorial optimization problems.

In particular, we propose novel formulations to learn inherently interpretable models such as decision trees. Our work includes encoding a variety of solution formats and objectives. We further propose frameworks and encodings for the integration of constraints during or after training. We show that our approaches successfully produce high-quality interpretable and constrained solutions in short runtimes, solving new problems and improving the state of the art in others. Our collective work shows that interpretability does not necessarily come at the expense of quality and, in fact, sometimes improves it. Utilizing constraints can

also improve accuracy despite reducing the space of feasible solutions. Lastly, we discuss how the work presented in this dissertation can be extended in several interesting directions and inspire new approaches for using combinatorial optimization problems in machine learning.

To My Family

Acknowledgements

I would like to start by expressing my deepest gratitude to my family, though I know words cannot do justice to all that they have done for me. We often pursue success by weighing all of the trade-offs along the way. However, we rarely consider the sacrifices that we impose on those who love us most. I want the unsung heroes of my life—my mother, father, and sister—to know that their support has shown me what it means to live truly. I feel unburdened by the arbitrary milestones we collectively and self-congratulatory came to perceive as goals. There is no higher goal to which I aspire than to embody their loving spirit in my life and be someone of whom they can be proud.

I would like to extend my heartfelt thanks to my supervisors—Professor Eldan Cohen and Professor Sheila McIlraith. My PhD had a troubled start which was further exacerbated by the restrictions imposed due to COVID-19. Despite the difficult circumstances, Sheila and Eldan not only provided me with an academic home but also made my PhD Journey better than I could have ever wished for. Sheila has taught me so much and has never let me feel deprived of her unwavering support and guidance, no matter how busy she was at the time. She truly deserves all the recognition she receives and more. Although I was Eldan's first graduate student, one would never know it, as I simply cannot think of any room for improvement in his excellent mentorship. I am confident that he will continue to excel and achieve great things in the illustrious career that lies ahead of him.

I would also like to thank my internal committee members—Professor Elias B. Khalil and Professor Xujie Si—as well as the members of the final exam committee—Professor Gilles Pesant and Professor Rick Valenzano. My PhD and this dissertation would not have been what they are without their insightful guidance and feedback. Last but not least, I would like to thank my friends in Sheila's and Eldan's research groups for their companionship and collaboration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Advances in modern Machine Learning (ML) including deep learning [75], reinforcement learning [255], and transformers [278], have been transformative and in that they have enabled solving complex problems in a diversity of domains. Given the prospect that ML will be integrated into much of our societal infrastructure, it is important to remain vigilant in ensuring that ML will act in support of human interests. Fortunately, there is considerable work on identifying the potential risks of employing ML-based solutions and the challenges of aligning them with human values [220, 149, 199, 239].

Modern ML algorithms can train complex models with a massive number of parameters where numerous small computations are involved in making each decision. The complexity of the models enables recognition of patterns within large corpora of data and generalization to new inputs with minimal intentional design [309, 127]. However, they commonly learn models that lack human *interpretability*, such as deep convolutional neural networks [242] and boosted trees [240]. It is challenging to understand the behavior of non-interpretable models, explain the decisions they make, conduct formal reasoning and augmentation on them, and detect their influential features and parameters [268, 308, 298].

The term *black-box* often describes a system where the inner workings are inaccessible, and interaction is limited to providing inputs and receiving outputs. However, lack of interpretability also makes the model a black box as it hinders our attempts at understanding and analysis, even when there is full access to all of the structural information and parameters. Modern ML algorithms also commonly learn models with no direct support for strictly enforcing user-specified *constraints* that confine their behavior. Loss functions can incorporate constraints, but they only encourage, rather than enforce, certain behaviors.

Black-box models and those with no direct support for constraints are challenging to rely on when safety, clarity, expert intervention, or compliance with specific requirements is needed. This limits their applicability when tackling safety-critical tasks in which decisions can have significant consequences. Such applications include, but are not limited to, a significant portion of tasks in medical [260, 107], societal [306, 4], and financial [186] domains. Moreover, not understanding a model and not being able to enforce user-specified constraints

makes it inherently incompatible with any problem that is accompanied by a safety or regulatory requirement [277, 288, 262]. All of the aforementioned applications require solutions that the user can trust, either by understanding them or being able to control their behavior. Without such solutions, one can imagine a multitude of ways that could lead to unfortunate outcomes. Examples include false negative predictions when testing for terminal illnesses, hidden biases in sensitive features such as ethnicity or gender impacting societal decisions, and accidents involving self-driving cars. A lack of support for formal constraints also cause inadequacies for some problems where rigorous reasoning is required, such as combinatorial optimization problems [31]. ML models can imitate complex reasoning without constraints, but usually lack guarantees of soundness or completeness [92].

Interpretable and constrained ML allows us to perceive and influence models in well-defined and compact formulations, facilitating application to sensitive tasks. We elaborate on what we mean by *interpretable* and *constrained* below.

- **Interpretable:** An interpretable model enables humans to understand and machines to reason about its decision making. Interpretable ML encompasses various methods aimed at addressing the challenges posed by black-box models [56]. Most relevant to our work is the approach of directly learning inherently interpretable models [134, 136] which are sparse, simulatable, modular, and operate on interpretable features [206]. Contrarily, interpretability can also be achieved in a post-hoc manner by multiple means such as instance-specific explanations [193, 138] or fitting interpretable models [268, 10]. Surprisingly, interpretable models are shown to not cause significant cost to accuracy [237, 177].

- **Constrained:** Imposing user-specified constraints on ML models allows controlling their behavior to strictly adhere to clear expectations. Constraints can be exercised to enforce satisfaction of formal specifications [170] or to integrate domain-specific knowledge to improve accuracy [284, 28, 66]. Similar to achieving interpretability, an attempt to constrain a model can be made during or after training. While the goal in the former approach is to learn a model with the desired properties, the latter approach instead aims to use the model in a framework that directs it toward desired behavior.

Combinatorial optimization refers to a family of problems with discrete components where the solution needs to satisfy a given set of constraints and optimize a given objective function [163]. It includes well-studied problems in areas such as graph theory [189, 85], resource allocation [241, 108], and routing [26]. Most combinatorial problems belong to the NP-hard complexity class, and thus, are not known to have efficient polynomial algorithms. Fast heuristic approaches can yield suboptimal results and naive exhaustive search grows quickly intractable even for small problems. Any NP problem can be reduced to an NP-complete problem and a common approach to solving combinatorial problems is to *formulate* them as other combinatorial problems that are declarative and standardized. These problems, referred to in this dissertation as *declarative* combinatorial optimization problems, are

accompanied by well-developed and efficient search, reasoning, and optimization techniques to solve them. Thus, the formulation approach eliminates the need for custom-made algorithms for each problem and generally enables solving combinatorial problems far faster than the exponential worst-case scenario [106, 1, 214, 44, 212].

Combinatorial optimization problems have been utilized to enable interpretable and constrained ML in various ways. Most commonly, the problem of finding an inherently interpretable model with an objective function based on a given dataset can be formulated into declarative combinatorial optimization problems [19, 124, 37]. The declarative nature enables the encoding of a wide array of interpretable and constrained ML problems, and solving these problems to optimality has been shown to improve accuracy compared to heuristic approaches [37, 124, 64]. Combinatorial components can also be integrated into the inference stages of deep learning algorithms for feedback or supervision [170, 299]. The declarative interface and the capability of complex reasoning with theoretical guarantees provided by the combinatorial component aptly complements the immensely scalable yet statistical models of deep learning. In this dissertation, we aim to explore the benefits of utilizing combinatorial optimization problems towards interpretable and constrained ML in novel and meaningful ways.

> **Thesis Statement.** Formulating machine learning tasks as declarative combinatorial optimization problems enables learning a wide variety of interpretable and constrained machine learning models, providing theoretical guarantees and often improving solution quality.

1.1 Chapters Outline and Contributions

Throughout this dissertation, we explore and investigate different ways in which combinatorial optimization problems can be utilized for interpretable and constrained ML. We propose novel formulations to learn inherently interpretable models in Chapter 3, Chapter 4, and Chapter 5. The approaches presented in Chapter 5 and Chapter 6 support enforcing constraints as formal specifications or semi-supervision on ML models. Chapter 6 proposes a framework where a combinatorial component can be utilized in conjunction with deep learning, rather than independently. In the following, we list the technical chapters of this dissertation and briefly describe their contribution.

Chapter 3: Optimal Decision Tree Classification

In this chapter, we propose a novel SAT-based encoding for the problem of classification with decision trees. Classification is the supervised task of learning a complete classifier function from a set of labeled training data. Decision trees have been learned as classifiers since they provide interpretability, while also being accurate [50, 215, 56]. Our approach is equipped with direct encoding of non-binary features and is shown to support multiple objectives, types of branchings, and pruning constraints. We experimentally show our encoding to

outperform the state-of-the-art approaches in datasets with non-binary features on two well-known objectives. We further demonstrate the capabilities of our encoding to mitigate issues such as overfitting or imbalance in data. To summarize, the key contributions of Chapter 3 are as follows:

1. We introduce a novel SAT-based encoding of decision trees for the problem of classification. Our approach supports direct encoding of numerical features, avoiding an explicit binarization which can lead to a quadratic increase in instance size [19, 281].

2. We use the encoding basis to solve two well-known classification objectives: (1) *min-depth*, i.e., find a tree of minimum depth that correctly classifies all of the training data [19, 39, 209]; and (2) *max-accuracy*, i.e., find a tree given a fixed depth that correctly classifies as many of the training data points as possible [281, 280, 6, 126, 37].

3. We introduce a direct encoding of categorical features. This encoding, like that of numerical features, avoids explicit binarization of features. However, it also enables a more expressive type of branching, referred to as the *power set branching*. Decision trees employing power set branching can achieve higher quality solutions as they are theoretically shown to be more expressive.

4. Through our experiments, we show that: 1) our decision tree approach with direct encoding of numerical features outperforms recent state-of-the-art techniques in both objectives on datasets with mostly numerical features; 2) our encoding produces optimal solutions more quickly, provides higher-quality solutions in case of timeouts, and consumes significantly less memory; 3) power set branching enables more expressive trees that both better fit the training data and generalize better to test data.

The work included in this chapter first appeared in the proceedings of the Twenty-seventh International Conference on Principles and Practice of Constraint Programming (CP 2021) [251]. An extended version, published in the journal, Constraints [253], introduced the new features of cost-sensitive learning and pruning constraints. The key extended contributions are as follows:

1. We add support for cost-sensitive learning by implementing an accuracy objective that allows different misclassification costs for different labels. Cost-sensitive learning facilitates application to problems with imbalanced datasets or varying error costs.

2. We present encoding for two tree pruning constraints: (1) *min-support*, i.e., the minimum number of points that must be contained within each leaf; and (2) *min-margin*, i.e., the minimum number of points that must exist on either side of each branching. Pruning constraints help reduce overfitting to the training examples and allow better generalization to unseen data.

3. Our extended experimental results show: 1) cost-sensitive learning can lead to significant improvement in *balanced accuracy* on imbalanced datasets; 2) tree pruning constraints can help reduce overfitting and lead to higher accuracy on unseen test data.

Chapter 4: Compact Binary Decision Diagrams for Classification

In this chapter, we focus on classification using binary decision diagrams, the more compact counterparts of decision trees. BDDs have historically been utilized as logical components [52, 9, 202, 159]. However, we learn them as interpretable classifiers following recent work [124, 160]. We introduce an encoding of BDDs in SAT, utilizing our approach to directly model non-binary features from Chapter 3. What distinguishes BDDs from decision trees is their compactness and higher emphasis on size optimization. We propose optimizing compactness by initially learning BDDs as trees and modeling the size to which they will be reduced. Alternatively, we propose multi-dimensional branchings, in line with our emphasis that branching nodes should not be limited to simple binary division of points. Multi-dimensional branchings enable direct learning of compact diagrams. To summarize, the key contributions of Chapter 4 are as follows:

1. We introduce a novel SAT-based encoding of maximum accuracy BDDs for classification with direct support for numerical features. Our approach initially models a non-reduced BDD, which can subsequently be reduced by assigning labels to empty terminals and merging equivalent nodes.

2. We extend our encoding by modeling the final size to which the BDD can be reduced. The size objective can be weighted against the accuracy objective in the same stage or be optimized in a secondary post-processing stage.

3. As an alternative to learning non-reduced BDDs and then reducing them, we present a variant of our approach that directly models multi-dimensional BDDs, a family of diagrams that employ multi-dimensional branching. Multi-dimensional BDDs are more expressive than BDDs using the same number of branchings and allow for efficient learning of higher-quality solutions in compact sizes.

4. Through our experiments, we show that our BDD encoding can improve the state of the art in solution accuracy and runtime. Furthermore, we show that encoding and optimizing size in the same stage or a secondary stage allows us to find significantly more compact solutions while maintaining accuracy. Finally, we show that multi-dimensional BDDs scale better than BDDs due to their expressiveness and ability to utilize more features in compact sizes.

The work included in this chapter appeared in the proceedings of the Twenty-ninth International Conference on Principles and Practice of Constraint Programming (CP 2023) [252].

Chapter 5: Constrained Clustering via Decision Trees

In this chapter, we propose a novel SAT-based model for learning decision trees as solutions to clustering, the task of recognizing patterns and grouping a set of unsupervised data into clusters. In addition to classification, decision trees have, to a lesser extent and more recently, been used as clustering solutions in the literature [88, 203, 38, 91]. Our approach guarantees approximation of a well-known bi-criteria objective, hence being the first exact approach to decision tree clustering. We exercise the flexibility in SAT formulations to add support for user-provided constraints, effectively solving the constrained clustering problem using decision trees. Our encoding utilizes our novel non-binary encoding of features, first proposed in Chapter 3. We also introduce a pruning algorithm that detects redundant or infeasible clauses by integrating constraints with the encoding of the objectives. Lastly, we propose multi-stage schemes to address the issues of infeasibility and lack of exploring the Pareto front. To summarize, the key contributions of Chapter 5 are as follows:

1. We present a novel MaxSAT encoding with direct support for numerical features to solve the problem of decision tree clustering. Our encoding guarantees an ϵ-approximation of a Pareto-optimal solution in maximum diameter (MD) and minimum split (MS) criteria. Our approach also supports user-provided constraints of must-links and cannot-links to solve the semi-supervised task of constrained clustering.

2. To reduce the considerable number of clauses required to encode the objective and pairwise constraints, we introduce the smart pairs pruning framework. The encoding of MD and MS, along with must-links and cannot-links, is integrated into the smart pairs algorithm to detect and prune redundant or infeasible clauses.

3. Our extensive experimental results show that: 1) our approach can find interpretable solutions of high quality in short runtimes; 2) employing tree clustering, the [MD,MS] objective, and user-provided constraints improves the solutions in evaluation metrics, with any combination of the three leading to an even greater enhancement; 3) there is a trade-off between solution accuracy and infeasibility in the presence of user-provided constraints; 4) objective approximation and smart pairs pruning significantly boost performance leading to higher quality solutions.

The work included in this chapter first appeared in the proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI 2023) [250]. Work on an extended version, introducing soft constraints and Pareto front generation, is in preparation for journal submission. The key contributions of the extension are as follows:

1. We further extend our encoding by introducing soft constraints whose satisfaction is optimized, rather than required. Soft constraints can address the issue of infeasibility when no tree is found to satisfy all of the hard constraints within a given depth. We propose 1-stage, 2-stage, and 3-stage schemes, with or without linearization, as

different approaches to maximizing the number of satisfied links before optimizing the clustering objective.

2. We detail how our encoding can be applied in an iterative algorithm that finds approximations to all solutions that are Pareto-optimal in [MD,MS], rather than just a single arbitrary one.

3. Our experiments show that soft constraints can address the infeasibility issue and allow the utilization of a higher number of constraints for improved accuracy. Moreover, our experiments reveal that exploring the whole Pareto front leads to finding higher-quality solutions in less constrained instances.

Chapter 6: Neural Sequence Generation with Requirements

In this chapter, we solve the problem of neural sequence generation with constraints. Neural sequence generation is a cornerstone problem in ML which has recently been utilized for solving combinatorial optimization problems [29, 162, 282, 170]. A combinatorial optimization application highlights the need for constrained solutions, as combinatorial problems are often strictly required to satisfy hard constraints [163]. We propose formulating the hard constraint aspect of a combinatorial optimization problem, i.e., a constraint satisfaction problem (CSP) [49, 106, 292, 16, 181], into a declarative combinatorial optimization problem without objectives. Doing so allows us to control a neural sequence model with the goal of enforcing satisfaction of the given set of constraints, referred to as a requirement, a guarantee that purely predictive decoders fail to provide [63]. Our approach augments the beam search procedure with cuts, which are explicit feasibility checks performed on partial solutions. *Beam search with cuts* (BSC) is modular and can combine any pre-trained neural model with a CSP requirement. Even though we perform a case study on Vehicle Routing Problems (VRPs), our implemented requirements are useful in multiple settings and our general framework is applicable to other neural sequence models and CSP requirements. To summarize, the key contributions of Chapter 6 are as follows:

1. We introduce beam search with cuts (BSC), a procedure to decode a pre-trained neural sequence model with the guaranteed satisfaction of a requirement, a set of constraints encoded as a CSP. BSC employs checks, referred to as cuts, preventing the expansion of partial solutions that cannot satisfy the requirement. Cuts are implemented by solving the CSP instances conditioned on a partial solution. Unlike ordinary beam search, BSC is complete, i.e., it is guaranteed to find a feasible solution if one exists.

2. We propose bin packing and regular language acceptance encoded in IP and SAT, respectively, as two types of requirements that find application in multiple settings. We present how adherence to a partial solution can be enforced in each, enabling us to implement them as cuts.

3. We focus on vehicle routing problems as our case study. Specifically, we solve three VRP variants by combining an existing pre-trained neural model [162] and our proposed requirements within a BSC procedure.

4. Our experiments reveal that beam search with cuts can satisfy requirements in a short runtime and without significant cost to solution quality. BSC is also shown to satisfy incrementally tighter requirements until infeasibility is, correctly, detected. Our experiments on comparison against ordinary beam search indicate that BSC scales exponentially better as the length of the solution is increased or the requirement is strengthened. Lastly, we show that an incremental solving of the CSPs significantly lowers the time spent by the solver.

The work included in this chapter is based on a paper that appeared in the proceedings of the Seventeenth International Symposium on Combinatorial Search (SoCS 2024) [249].

1.2 Concluding Remarks

Modern ML techniques can achieve super-human performance in various domains, leading to numerous applications in our daily lives. Such techniques commonly employ complex black-box models that are a risk to use in sensitive tasks such as those with safety or clarity concerns. Interpretable ML aims to bring about the ability to understand models and constrained ML empowers the user to clearly specify and strictly impose constraints.

In this dissertation, we propose to use declarative combinatorial optimization problems to formulate both the problem of learning inherently interpretable models and the constraints that may be imposed on interpretable or non-interpretable models. In Chapter 2, we provide the necessary background information from the literature on interpretable ML, constrained ML, and how combinatorial optimization problems have been utilized to achieve both, setting the stage to present our contribution in the same areas. Chapter 3, Chapter 4, and Chapter 5 propose SAT-based models to learn inherently interpretable models in the form of decision trees or diagrams. More specifically, Chapter 3 and Chapter 4 investigate learning decision trees and diagrams, respectively, as classifiers, while Chapter 5 investigates learning decision tree clustering solutions. In Chapter 6, we propose a framework that solves a combinatorial optimization problem by formulating its constraints into a declarative combinatorial problem that constrains a neural sequence model generating solutions. Lastly, we summarize our contributions and discuss interesting potential extensions of our work in Chapter 7.

Chapter 2

Background

In this chapter, we go over the existing literature that is relevant to this dissertation, i.e., interpretable and constrained ML and the utilization of combinatorial optimization to that end. Most classical approaches to AI such as heuristic search [218, 119], knowledge representation and reasoning [30], logical planning [100, 3], and exact optimization [37, 66, 291] have an algorithmic and symbolic perspective to producing solutions. Given this perspective, the interaction with the solution is strict and well-defined [217], paving the way for the user to understand and influence the solution. Following the rise of ML and statistical approaches to AI, particularly deep learning in the last two decades, the level of reliance on precise formulations and algorithms slowly slipped away and was mostly replaced by black-box models, which utilize large amounts of data to detect and replicate existing patterns [176, 201]. However, there is considerable work on making use of symbolic, combinatorial, and rigorous components to facilitate trust in ML solutions. Such approaches employ alternative interpretable models or complement the black-box models in a hybrid approach, enabling interpretable and constrained solutions. We survey how this line of work has developed and in which applications it has proven to be useful.

2.1 Combinatorial Optimization

Numerous interesting and useful combinatorial problems are known to belong to the NP-complete complexity class [97]. The NP-complete class contains decision problems where feasible solutions are demanded. The optimization variation of these problems demands optimal solutions and belongs to the NP-hard class. Polynomial algorithms can solve approximated [279], randomized [205], parameterized [180], and relaxed [188] variations of NP-hard optimization problems, but are yet to be shown to fully solve them to optimality. Despite the exponential worst-case complexity, methods have been developed to solve combinatorial optimization problems in practical runtimes. These methods are exact in the sense that they employ search, pruning, backtracking, propagation, and relaxation techniques to exhaust the search space and find feasible (optimal) solutions [291, 174, 163].

Rather than directly solving a combinatorial optimization problem with a hard-coded algorithm, we can also formulate it as an intermediate declarative problem and use a unified solving procedure. The declarative problems in the formulation approach are themselves NP-hard. They are also able to easily encode most combinatorial constraints and objectives in a natural way. Declarative problems without an optimization component are commonly referred to as constraint satisfaction problems (CSP) [49]. Once a problem is formulated into a declarative problem, efficient off-the-shelf solvers can allow the solving of relatively difficult instances that have up to millions of variables in some cases. Any-time solvers return the best found solution in case of a timeout but are guaranteed to find an optimal solution given enough time and resources. Formulation approaches allow solving techniques that are independent of problems. They also allow flexibility in problem specification and modularity in the components of each problem. In practice, different declarative combinatorial optimization problems might be more suited to solving different tasks.

In Section 2.1.1 and Section 2.1.2, we present boolean satisfiability (SAT) and integer programming (IP), respectively. SAT and IP are the two declarative problems that have CSP and optimization variants and are used throughout this dissertation. Other well-known declarative combinatorial optimization problems which are not used in this thesis include constraint programming (CP) [16], satisfiability modulo theory (SMT) [27], and dynamic programming (DP) [169].

2.1.1 Boolean Satisfiability Problem

In this section, we define the boolean satisfiability problem (SAT). SAT was the first problem to be proven to be NP-complete and remains an integral part of studying the NP complexity class. A reduction from a more limited SAT variation (3-SAT) is a standard way of proving new problems to be NP-hard, and a reduction to SAT is a common way of solving NP-complete problems by using efficient off-the-shelf SAT solvers [214, 80, 116].

A SAT instance uses propositional variables v_i. A literal l_i is a variable v_i or its logical negation $\neg v_i$. The negation of a variable is `true` if and only if the variable itself is `false`. A clause is a disjunction (logical OR) of literals $(l_1, ..., l_i)$ and is considered to be satisfied when at least one of its literals is `true`. A SAT formula is a conjunction (logical AND) of clauses and is considered to be satisfied when all clauses are. This arrangement of the formula is referred to as the conjunctive normal form (CNF). When presented in CNF, it is NP-hard to decide whether there exists a truth assignment to the variables that satisfies a formula. Disjunctive normal form (DNF) is an alternative structure where clauses are conjunctions of literals and the formula is the disjunction of the clauses. In DNF, it is co-NP-hard to decide whether the formula is a tautology, i.e., it is satisfied under all truth assignments. SAT solvers mostly focus on solving the satisfiability problem for CNF formulae. To solve an NP problem using SAT solvers we: (1) provide a valid reduction from the problem to SAT, i.e., the answer to the original instance is yes if and only if the resulting SAT formula is satisfiable; (2) solve the SAT formula; and (3) decode the satisfying truth assignment (if

any exists) back to a solution for the original problem following the reduction [41, 106].

Maximum satisfiability (MaxSAT) is the optimization variation of SAT. A MaxSAT formula has the same structure in terms of variables, literals, and clauses. However, satisfiability is not the question as the satisfaction of clauses is not mandatory. Instead, the number of satisfied clauses is the objective function to be maximized. MaxSAT is an NP-hard optimization problem and the goal is to find the truth assignment where the number of satisfied clauses is equal or more than any other assignment. Partial MaxSAT [89] is a generalization of both SAT and MaxSAT. In partial MaxSAT, variables and literals are as before, but clauses are divided into hard and soft subgroups. Satisfaction of hard clauses is mandatory, but that of soft clauses is to be optimized. Solving partial MaxSAT has the same complexity as standard MaxSAT and amounts to finding a truth assignment that both satisfies the formula and a number of soft clauses that is the highest amongst all satisfying assignments. Note that using (partial) MaxSAT to solve NP problems also requires reduction, solving, and decoding. Additionally in the reduction, the objective of the original problem should monotonically correspond to the number of soft clauses satisfied to ensure a valid optimization. Hereafter, the simplified term, clause, is used to refer to hard clauses, and MaxSAT is used to refer to the partial MaxSAT problem.

Most SAT solvers are based on the DPLL algorithm [171, 213] where at each iteration: (1) a variable is chosen and given a truth value; (2) satisfied clauses are removed and clauses containing the falsified literal are simplified; (3) if a unit clause is detected, a satisfying value is assigned and the iteration repeats from step 2; (4) if a variable has only one literal representation, a satisfying value is assigned and the iteration repeats from step 2; (5) if all variables are assigned the solution is returned; and (6) if an empty clause is reached a variable assignment is backtracked, unless all possibilities are exhausted which leads to a claim of infeasibility. Modern SAT solvers also employ conflict-driven clause learning (CDCL) [195] where every time an empty clause is reached, the algorithm finds the truth assignments leading to the conflict and learns them as a clause to avoid them without explicit backtracking. MaxSAT instances are commonly solved through iterative translation to SAT instances. Approaches based on linear search start from a satisfying solution and iterate on the lower bound by finding strictly better solutions until it is no longer feasible [196]. Contrarily, core-based approaches start by considering all soft clauses as hard ones and learn infeasible minimal sets of clauses that are iteratively relaxed to find a satisfying solution [33, 72].

To be accommodated in SAT, some non-CNF constraints can be translated to a set of CNF clauses using automated procedures with potentially added auxiliary variables. Most notably, there exists a variety of algorithms with different strengths and weaknesses to translate cardinality constraints into CNF clauses [256, 24, 194]. A cardinality clause $\Sigma_i l_i \leq c$ bounds the number of true literals in its given set. In a valid translation, a truth assignment satisfies the cardinality clauses if and only if it satisfies the resulting set of CNF clauses. Recent work has developed SAT solvers with direct support for cardinality clauses without requiring CNF translation.

Certain SAT solvers can support incremental solving when multiple similar instances are presented to the solver in succession [80]. Incremental solving aims to carry over learned information to future steps in order to avoid repetition and improve the overall performance [80, 207]. The successive instances that are incrementally solved share variables and clauses. They are purely distinguished through a subset of literals, called assumptions a_i. The goal of each step in incremental solving is to find a solution that satisfies the shared formula alongside the particular assumptions of that step. The carry forward information can include clauses, such as those learned in CDCL, under the assumption that they are implied by the hard clauses alone and are not dependent on the assumptions.

2.1.2 Integer Programming

In this section, we define the problem of integer programming (IP) [292]. IP is an NP-hard optimization problem, but comes with efficient off-the-shelf solvers [112, 150]. Due to its expressiveness and well-developed array of solving methods, IP is considered to be a fundamental problem with wide-ranging applications in operations research [211, 129, 313, 244].

An instance of IP consists of an array of variables x, with each individual variable having an integer domain and represented as x_i. Variables are commonly required to have non-negative values $x \geq 0$. However, the variation with support for negative values can be translated into the standard format by representing each variable x_i as the subtraction between two standard variables $x'_i - x'_j$ with $x'_i, x'_j \geq 0$. Constraints in an IP instance are linear inequalities with real-valued coefficients and constants $Ax \leq b$, represented by a matrix A and array b, respectively. The objective is the maximization of the linear expression $c^T x$ where c is a real-valued array representing objective coefficients. A standard formal IP instance in its entirety is presented in Eq. (2.1). To solve IP, one should find values for variables that satisfy all linear constraints and have the highest objective value compared to all other satisfying solutions. IP also has a decision variation by removing the objective component, where the goal is to simply find satisfying values for variables. The decision variation of IP is NP-complete. To solve a problem using IP we: (1) provide a reduction that maintains a yes/no answer for a decision problem and monotonically corresponds to the objective for an optimization problem; (2) solve the IP instance using an off-the-shelf solver; and (3) decode the integer values (if the instance is not found to be infeasible) back to a solution for the original problem by following the reduction in reverse.

$$\begin{aligned} &maximize && c^T x \\ &s.t. && Ax \leq b \\ & && x \geq 0 \end{aligned} \tag{2.1}$$

The problem of linear programming (LP) has the same structure as IP without the lim-

itation to integer values for the variables. Solving LP has numerous applications and is well-established to have very efficient polynomial algorithms [156, 154]. IP has also been extended to multiple more expressive paradigms. Mixed-integer programming (MIP) [1] allows variables with real-valued domains to coexist with integer variables, generalizing both IP and LP. Mixed-integer quadratic programming (MIQP) [175] further enriches the paradigm by adding support for quadratic objective functions. The increase in expressiveness in these variations allow more problems to be modelled but come at a cost to performance.

Transforming an IP instance into an LP one by removing the integer limitation is referred to as relaxation and plays a crucial role in solving IP. Note that since any solution to the IP is a solution to the relaxed instance, the relaxed optimal objective value is guaranteed to be equal or larger than that of the IP, effectively providing an upper bound. Furthermore, the objective value of any feasible solution is effectively a lower bound on the optimal value. The branch-and-bound algorithm [130] utilizes these two values by maintaining a best solution yet, and solving relaxations of current instances at each search node. The lower and upper bounds can be used to prune search nodes that are guaranteed not to contain any solutions better than the current best. When the solution to a relaxation is all integer, it is returned as the optimal case in that particular node. If it is not, a branching is made on the floor and ceiling values of a real-valued variable, adding corresponding constraints to each child node. In a branch-and-bound approach to solving IP, the algorithm can be terminated prematurely and still provide a potentially sub-optimal solution with a guaranteed bound on the optimality gap.

2.2 Interpretable Machine Learning

The work on interpretable ML is focused on building solutions whose inner workings are not, unlike black-box models, a mystery. What constitutes as interpretability is the subject of great debate in the literature [206, 182, 17]. It is acknowledged that interpretability in its essence is a subjective criterion and is highly dependent on user's expectations and the demands of a particular setting. More broadly, Lipton [182] claims that the demand for interpretability arises when formal objectives fail to encompass costs of real-world deployment. However, there have been more concrete attempts at understanding how interpretability notions relate and can be quantified. We mostly rely on the definitions provided in Murdoch et al. [206]. The authors describe the three stages in data-science life cycle, learning a model based on data and preferences, analyzing the model, and reiterating. Interpretability is then defined as the extraction of relevant knowledge from the relationships contained in the data or the model. As the three main desiderata for interpretability, they argue for accuracy in modeling and analysis, i.e., predictive and descriptive accuracy, respectively, and for relevancy, i.e., providing insight for a particular audience into a chosen domain problem. The approaches discussed in Murdoch et al. are categorized into model-based interpretability and post-hoc interpretability, referring to using inherently interpretable models or analyzing black-box models, respectively.

2.2.1 Modelling Stage

Interpretability in the modelling stages is mostly reliant on inherently interpretable models. However, learning interpretable models also has a positive impact on interpretability in the analysis stage, greatly improving descriptive accuracy. Interpretable models are: (1) sparse, employing a small number of parameters; (2) simulatable, having a simple structure and low number of features to be understandable by humans in its entirety; (3) modular, being made up of meaningful decision-making components that can be interpreted independently; and (4) operate on interpretable features engineered based on domain-specific knowledge or (5) by using automated techniques. A well-known family of inherently interpretable models are rule-based models such as decision trees [230, 165, 11], decision lists [234, 307], decision sets [172, 131], and association rules [164, 20]. Such models are compact and humans can usually understand them by following the sequence of activated rules and corresponding features. This simulatability can then be extended to the model as a whole, barring the issues due to the exponential growth of some models such as decision trees. Rule-based models are also highly modular, as the decision corresponding to each rule is independent of the rest. Furthermore, models such as decision trees and lists employ recursive decision-making where each node is a smaller model of the same type. Interpretability of rule-based models is contingent upon the interpretability of their features, and their accuracy is significantly impacted by feature engineering since new features cannot be learned within the model, unlike more complicated models such as neural networks. Rule-based models are also commonly learned using combinatorial optimization problems, contributing to algorithmic transparency which is a less-known aspect of interpretability [182].

Other examples of inherently interpretable models include linear [146] and logistic [191, 151] regression, particularly their sparse variants [263, 185], which can be considered special cases of generalized additive models [293, 312]. Moreover, models that rely on representative points, e.g., prototype-based ones [40, 248, 305], also provide a degree of interpretability. However, a model whose behavior is fully formulated is more interpretable as the interpretation is not dependent on a given input. Lastly, neural-symbolic approaches aim to integrate reasoning mechanisms and knowledge representation as interpretable components into neural networks without fully replacing them [22, 92, 93]. Intuitively, the symbolic aspect of such models represents the cognitive ability to reason about what has been learned by the neural component in semantics such as propositional [270, 96], first-order [86, 246], and temporal [94, 95, 46] logic. Neural-symbolic approaches also lack total interpretability due to the neural aspect.

2.2.2 Analysis Stage

Interpretability in the analysis stage can be achieved in a post-hoc manner for black-box models. However, it is also possible, and easier, for model-based approaches. Many attempts at interpreting an ML model can be categorized as explanations, referred to as prediction-level interpretation in Murdoch et al. Explainability is adjacent to interpretability and de-

mands prediction on a given input to be accompanied by additional instance-dependent information that explains the decision-making [297, 88]. Types of explanations include the previously discussed representative points, in addition to abductive and contrastive explanations [144, 138] which focus on why a decision is made, and why a differing decision is not made, respectively. Inherently interpretable models enable explainability, just as they do simulatability, because their inner workings are fully formulated and structured in a simple way. For example, explanations can be extracted from rule-based models by retrieving the set or sequence of rules that were activated [136, 134]. Explainability can also be achieved for black-box models [109, 245, 197]. Explanation of complex models are often local [110, 233, 226], commonly imitating the behavior in the surrounding space of a given input. Neural-symbolic approaches can also enable explainability in the form of extracting rules that explain part of the model in human-compatible format [14, 286].

The instance-independent variation of post-hoc analysis is to learn post-hoc models. referred to dataset-level interpretation in Murdoch et al. It interprets the model's behavior as a whole and, as such, is mostly irrelevant for the model-based approach where the model is inherently interpretable. Post-hoc models can be inherently interpretable ones with the objective of approximating a given black-box model, rather than fitting a set of training data [166, 10, 268]. However, recent work has demonstrated that, in most cases, there is more utility to learning interpretable models directly compared to learning them for interpreting black-box ones [237].

2.2.3 Combinatorial Optimization for Interpretable ML

Combinatorial Optimization problems can be utilized in model-based approaches to interpretable ML. Learning inherently interpretable models are mostly NP-hard problems [173, 234]. Thus, in practice, they are either learned using heuristics that can yield suboptimal solutions [50, 229, 32, 304, 102] or using exact combinatorial approaches that can be computationally costly. Recent work learns decision trees with search [6, 74] or by using declarative combinatorial problems such as SAT [19, 209, 126], IP [37, 111], and CP [280, 39]. Combinatorial approaches to learning interpretable models for clustering have also been proposed [7, 38], but shown to scale poorly. Problems such as SAT can also be used to learn decision lists [136], sets [301], diagrams [123], and interpretable models based on temporal logic for classifying sequences of observations [55]. Moreover, combinatorial optimization problems can be used to achieve post-hoc interpretability by explaining existing interpretable models [134, 142].

2.3 Constrained Machine Learning

Another approach to making ML solutions more human-compatible is through constraining their behavior to well-formulated specifications. Constraints for an ML solution can embody domain-specific knowledge, regulatory specifications, or learning conditions. Constraints can

be imposed while the model is being learned, concerned with the properties of the model or its behavior with regard to certain data distributions. Training-time constraints can enforce safety properties [277] on the model in problems such as Bayesian optimization [36] and reinforcement learning [300, 222, 65, 187]. Specifications on the model are not limited to safety and can also represent societal concerns by bounding [153] related criteria instead of optimizing [4] them. Constraints in training-time are also used to integrate user-provided knowledge into the training to guide the solution and increase accuracy in problems such as clustering [184, 283] and classification [82]. Lastly, constraints can be utilized to improve the training procedure itself through pruning [280, 98]. Interpretable models can easily accommodate enforcing and verifying additional constraints due to their well-defined structure [208, 231]. In particular, if the learning of an inherently interpretable model is formulated into a declarative combinatorial optimization problem, the same problem can act as a host for encoding additional constraints [280, 37].

Constraints can also be imposed while the model is being utilized, controlling its output. Inference-time constraints can enforce requirements such as adherence to a formal syntax [219] or guaranteed inclusion of elements [13, 179]. Inference-time constraints are independent of the size of training data and how the original model is trained, allowing them to benefit from the scalability of non-combinatorial approaches such as deep learning. Imposing inference-time constraints is necessitated to a higher extent when the model generates sequence solutions, since iteratively navigating a model is more challenging than simply filtering out its unwanted one-time predictions. Note that training-time constraints also impact inference as they shape the model that is being utilized, but the distinction is their involvement in training. Given that constraints are well-defined symbolic components, they can also be used to complement deep learning models in rigorous reasoning capabilities [299]. Contrary to the approach of utilizing deep learning as a heuristic to enhance the performance of algorithms for solving combinatorial optimization [259, 57, 157].

2.3.1 Combinatorial Optimization for Constrained ML

Combinatorial optimization problems can be used to learn ML models with guaranteed satisfaction of training-time constraints. Declarative problems like SAT [71], CP [66, 280], and MIP [4] or more specialized ones like Ising models [64] can be used to solve tasks such as constrained classification or clustering. In these cases, satisfaction of the constraints is integrated into the same formulation that learns the, potentially interpretable, model and the problem is solved as a whole by an exact method. If constraints are inference-time and not intertwined with training the model, they are typically formulated as CSPs, i.e., combinatorial problems that lack the optimization aspect but are still NP-hard. Although less prevalent, combinatorial components can also be responsible for satisfaction of inference-time constraints on the output of pre-trained models, particularly noticeable in the context of neural sequential generation [170, 299].

Chapter 3

Optimal Decision Tree Classification

3.1 Introduction

In this chapter, we present an approach that employs combinatorial optimization problems to learn inherently interpretable models for a problem that is typically solved using black-box models in deep learning. We focus on the problem of classification in which solutions assign class labels to data observations. Learning classification models from a set of training examples is a key task in supervised ML.

The interpretable models that we learn as combinatorial optimization problems are decision trees. Decision trees are among the most popular classification models in ML as they tend to have good performance in addition to providing interpretability. They satisfy the conditions for an interpretable model as presented in Murdoch et al. [206]. They are sparse as they employ a small number of features and parameters overall. They are simulatable as every prediction corresponds to traversing a path from the root to a leaf, with impact from an even smaller number of features. Lastly, they are modular as the branching within each node is independent of what has happened before and will happen next. Decision trees are also transparent and facilitate generation of explanations of why a data point was or was not assigned a label. Moreover, they are easy to analyze with regard to behavioral properties. Traditionally, decision tree classifiers are constructed using greedy heuristic algorithms, such as CART [50], ID3 [230], and C4.5 [229]. However, these algorithms do not provide guarantees on the quality of the resultant trees, which can therefore be unnecessarily large or potentially inaccurate [32].

Alternatively, learning a globally optimal decision-tree classifier was shown to be NP-hard for several optimization criteria [173, 117]. Nevertheless, in recent years a variety of exact techniques have been proposed to solve the problem of constructing optimal decision tree classifiers. One class of techniques focuses on optimizing decision tree size (either the

depth of the tree or the number of nodes in the tree) such that all training examples are correctly classified [19, 39, 209]. Another class of techniques, instead, focuses on maximizing the number of correctly classified training examples, while constraining the maximal depth of the decision tree [281, 280, 6, 126, 37].

The guaranteed optimality provided by an exact approach to solving combinatorial optimization is not necessary in all ML applications and, often, a solution with a high enough quality meets the demands. Therefore, the added computational cost is not always justified, despite the gap in performance becoming narrower recently [139]. Note that it is also possible for a non-exact approach to produce optimal decision trees, but their optimality is not proven, guaranteed, nor bounded, as is the case when using exact approaches with sufficient resources. Optimal approaches are desirable when even a slight improvement in accuracy takes priority over computational tractability. Optimality is particularly of significance since we are intending decision trees as interpretable models. Generally, interpretable modes cannot compensate for the gap in quality by simply expanding further as this will compromise their sparsity and simulatability. This highlights the need for precise and optimal reasoning that considers the model as a whole to find the best possible solution in a small and highly confined format, rather than using a stepwise heuristic approach [50]. Surprisingly, optimality does not cause decision trees to overfit training data [37], but rather leads to a better reflection of the ground truth, improving out-of-sample accuracy [111, 19]. Optimal solutions in similar interpretable models such as decision sets and decision lists also seem to improve generalization [139].

As discussed in Chapter 2, combinatorial optimization problems can be solved by formulation in a declarative problem such as boolean satisfiability (SAT), its optimization variant maximum satisfiability (MaxSAT), integer programming (IP), or constraint programming (CP) and using efficient off-the-shelf solvers. Enabled by the recent advancements that make it possible to solve challenging instances [214, 212], the formulation approach have been successfully utilized to learn decision trees [280, 126, 39, 209]. The formulation approach enables modularity and flexibility in learning different variants with different objectives including minimum depth and maximum accuracy. It further allows constrained ML by enforcing the resulting decision tree to satisfy constraints, such as those related to fairness [4] and those preventing overfitting [165].

Many of the recent state-of-the-art approaches are designed for datasets with binary features [19, 281, 209, 126, 280], and some are further limited to binary class labels (i.e., classification with only two classes) [209, 126]. However, in practice, most real-world datasets contain categorical and numerical features. In order to perform classification in datasets with categorical and numerical features, these approaches convert any non-binary feature into a set of binary features using standard techniques (see discussions in [19, 281]). This conversion often introduces a large number of binary features and may lead to poor performance.

In this chapter, we present and empirically evaluate a novel approach for learning optimal decision-tree classifiers that can directly handle non-binary features without converting them

into binary features. Specifically, we make the following contributions:

1. In Section 3.3, we present a novel SAT encoding of decision trees that directly supports numerical features, and use this encoding to solve two well-known optimization problems in classification tasks: (1) finding a decision tree of a given depth that maximizes the number of correctly classified training examples; and (2) finding a minimum-depth decision tree that correctly classifies all training examples.

2. In Section 3.4, we present an extension of our SAT encoding that directly supports categorical features based on power set branching and show, theoretically and empirically, that the new encoding is more expressive and can lead to decision trees with better solution quality with regard to the two studied optimization problems.

3. In Section 3.5, we present an extension of our SAT encoding for cost-sensitive learning of optimal decision trees that can be used in applications with non-uniform misclassification costs or imbalanced datasets.

4. In Section 3.6, we present SAT-based encodings for two types of tree pruning constraints that can help reduce overfitting to the training dataset.

5. In Section 3.8, we perform extensive experimental analysis based on the setup described in Section 3.7 and show that: 1) Our encoding for decision trees with numerical features significantly outperforms recent state-of-the-art techniques on datasets with numerical features for each of the studied optimization problems and has significantly smaller memory consumption; 2) Our extension for categorical features produces decision trees that better fit the training data and often generalize better to unseen test data; 3) Our extension for cost-sensitive learning can lead to significant improvement in *balanced accuracy* on datasets that suffer from class imbalance; 4) Our tree pruning constraints can help reduce overfitting and lead to higher accuracy on unseen test data.

This chapter is heavily based on the work that first appeared in the proceedings of the Twenty-seventh International Conference on Principles and Practice of Constraint Programming [251]. An extended version was later published in The Constraints journal [253], which added cost-sensitive learning, model enhancements through pruning constraints, new theoretical results, and a significantly expanded experimental evaluation.

3.2 Preliminaries

In this section, we provide the formal definition of decision trees that is used throughout this chapter and occasionally the rest of the dissertation. We define decision trees by first defining branchings and tree structures as a basis. In Sections 3.2.1 and 3.2.2, we discuss some characteristics of decision trees and a theoretical result on their completeness, respectively.

Lastly, we define the classification problems that we solve with decision tree solutions in Section 3.2.3.

Definition 1 (Branching)**.** *Given a finite set of features F, a branching is a division of data points into two groups. A branching can either operate on a binary feature $f_b \in F$ or a binarization of a numerical feature $f_n \in F$ through pairing with a threshold $a \in \mathbb{R}$. For $x[f_n]$ which represents the value of numerical feature f_n for data point x, being smaller or equal to the threshold ($x[f_n] \leq a$) equates to a binary value of* 1 *while being larger ($x[f_n] > a$) equates to* 0*.*

Note that a binary feature can be seen as a special case of a numerical one. Henceforth, we will only use numerical features to be concise, unless noted otherwise. Furthermore, we will only use the more general representation of a branching, which is a feature paired with a threshold.

Definition 2 (Tree Structure)**.** *A tree structure $\mathcal{T}$ is a rooted directed tree. Nodes of $\mathcal{T}$ are divided into leaf nodes $\mathcal{T}_L$ and non-leaf nodes that are referred to as branching nodes $\mathcal{T}_B$ and contain the root node $\delta \in \mathcal{T}_B$. Each branching node has exactly two children orienting away from the root represented by the left and right children function $l, r : \mathcal{T}_B \to (\mathcal{T}_B \cup \mathcal{T}_L)$. The parent function $p : (\mathcal{T}_B \cup \mathcal{T}_L - \{\delta\}) \to \mathcal{T}_B$ represents the edges in reverse $\forall t_p, t_c : p(t_c) = t_p \Leftrightarrow (l(t_p) = t_c \vee r(t_p) = t_c)$.*

Definition 3 (Decision Tree)**.** *Given a set of features F and integer labels $[1..k]$, a decision tree is a tuple $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ where $\mathcal{T} = (\mathcal{T}_B, \mathcal{T}_L, \delta, p, l, r)$ is a tree structure, $\beta : \mathcal{T}_B \to F$ is the feature selection function, $\alpha : \mathcal{T}_B \to \mathbb{R}$ is the threshold selection function, and $\theta : \mathcal{T}_L \to [1..k]$ is the leaf labelling function.*

Next, we describe how decision trees assign labels to data points. This process involves entering the data point at the root and iteratively delivering it to the left or right child based on its values until it reaches a leaf node with a label.

Definition 4 (Decision Tree Label Assignment)**.** *Given a set of features F, integer labels $[1..k]$, a data point $x \in \mathbb{R}^{|F|}$ (an evaluation of F), and a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$, the value of the function $\Theta_{\mathcal{D}}(\delta, x)$ represents the label that $\mathcal{D}$ assigns to x. The function $\Theta_{\mathcal{D}} \in (\mathcal{T}_B \cup \mathcal{T}_L) \times \mathbb{R}^{|F|} \to [1..k]$ is recursively defined as follows:*

$$\Theta(t, x) = \begin{cases} \theta(t) & \text{if } t \in \mathcal{T}_L \\ \Theta(l(t), x) & \text{else if } x[\beta(t)] \leq \alpha(t) \\ \Theta(r(t), x) & \text{otherwise} \end{cases}$$

Every step of the recursion corresponds to a branching. A value of 1 *(*0*) for the branching leads the point to be* directed *to the left (right) child. Every node t invoked with a point x is said to* contain *that point. A leaf node assigns its label to all of the data points that it contains.*

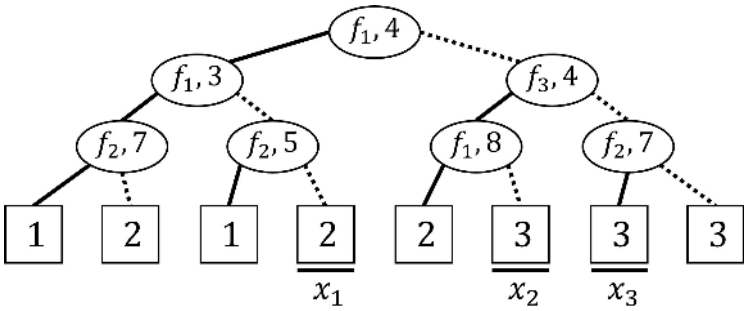


Figure 3.1: A decision tree example. Solid (dotted) lines correspond to the left (right) children. Branching nodes are labeled with with $(\beta(t), \alpha(t))$ values. Leaf nodes are highlighted with the data point that they contain.

We use the simplified notation $\Theta_{\mathcal{D}}(x)$ to denote $\Theta_{\mathcal{D}}(\delta, x)$. When the decision tree $\mathcal{D}$ is intended as the solution to a classification problem, $\Theta_{\mathcal{D}}(x)$ is interpreted as assigning classification labels. The example in Figure 3.1 depicts a decision tree operating on features (f_1, f_2, f_3) assigning labels to $x_1 = (4, 7, 3)$, $x_2 = (9, 5, 2)$, and $x_3 = (6, 1, 8)$.

3.2.1 Decision Tree Properties

In this section, we discuss some properties of decision trees. We first define the depth of a decision tree and the property of being fully balanced.

Definition 5 (Decision Tree Depth)**.** *Given a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$, we recursively define the depth of each node $t \in \mathcal{T}_B \cup \mathcal{T}_L$ as $depth(t) = depth(p(t)) + 1$ with $depth(\delta) = 0$. The depth of $\mathcal{D}$ is then defined to be the maximum depth among its leaf nodes $depth(\mathcal{D}) = \max_{t \in \mathcal{T}_L} depth(t)$.*

Definition 6 (Fully Balanced Decision Tree)**.** *A decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ is said to be fully balanced if all of its leaf nodes have the same depth, i.e., $\forall t_1, t_2 \in \mathcal{T}_L : depth(t_1) = depth(t_2)$.*

Next, we define the support of nodes and the decision tree as a whole.

Definition 7 (Decision Tree Support)**.** *Given a set of features F, a dataset $X \subset \mathbb{R}^{|F|}$, and a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$, the support of a node $t \in \mathcal{T}_B \cup \mathcal{T}_L$ with regard to dataset X is the subset of X that t contains. It can be recursively defined as follows:*

$$\Psi(\delta) = X \tag{3.1}$$

$$\Psi(l(t)) = \{x \mid x \in \Psi(t), \Theta^{\leftarrow}(t, x)\} \qquad \forall t \in \mathcal{T}_B \cup \mathcal{T}_L \tag{3.2}$$

$$\Psi(r(t)) = \{x \mid x \in \Psi(t), \neg\Theta^{\leftarrow}(t, x)\} \qquad \forall t \in \mathcal{T}_B \cup \mathcal{T}_L \tag{3.3}$$

where $\Psi(t)$ represents the support of t and $\Theta^{\leftarrow}(t, x)$ indicates that x is directed to the left child at node t, namely $x[\beta(t)] \leq \alpha(t)$. The support of a decision tree is then defined to be the minimum cardinality support amongst all of its leaf nodes.

3.2.2 Decision Tree Completeness

In this section, we state that decision trees are complete in the sense that they can represent any consistent labeling present in a given dataset.

Definition 8 (Consistent Labelling)**.** *Given a set of features F, integer labels $[1..k]$, and a set of data points $x_i \in X$, a labeling $\gamma : X \to [1..k]$ of dataset X is consistent if and only if there are no two identical points that are assigned different labels, i.e.,*

$$\neg\exists x_1, x_2 \in X : (\forall f \in F : x_1[f] = x_2[f]) \wedge \gamma(x_1) \neq \gamma(x_2). \tag{3.4}$$

Theorem 1. *Given a set of features F and integer labels $[1..k]$, a set of data points $x_i \in X$, and a labelling $\gamma : X \to [1..k]$ such that γ is a consistent labelling of X, there always exists a fully balanced tree $\mathcal{D}$ with sufficiently large depth($\mathcal{D}$) that completely matches the given labelling $\forall x \in X : \Theta_{\mathcal{D}}(x) = \gamma(x)$.*

Proof. We provide a constructive proof for the theorem. Our proof starts with a single root node δ and iteratively deepens the tree until the labels of the points contained in each leaf node are homogeneous. Finally, each leaf node is labeled with the homogeneous label of the points in its support. The procedure is formally described below.

1. Initiate the tree structure with a single root node δ as a leaf and the decision tree with empty α and β functions.[1]
2. If there is no leaf node t such that $\exists x_1, x_2 \in \Psi(t) : \gamma(x_1) \neq \gamma(x_2)$, go to (7).
3. For every leaf node t, replace t with a branching node t_p and two leaf nodes t_l and t_r as its left and right children. If $\exists x_1, x_2 \in \Psi(t) : \gamma(x_1) \neq \gamma(x_2)$, go to (4). If not, go to (5).
4. Due to consistent labeling, we know there exists feature f such that $x_1[f] \neq x_2[f]$. Set $\beta(t_p) = f$ and $\alpha(t_p) = min(x_1[f], x_2[f])$. Go to (6).
5. Set $\beta(t_p)$ and $\alpha(t_p)$ to any arbitrary valid value.
6. Go to (2).
7. For each leaf node t, set $\theta(t) = \gamma(x)$ where $x \in \Psi(t)$. In case the support is empty, label the leaf arbitrarily.
8. Output the constructed decision tree.

This procedure is guaranteed to halt since the number of pairs of data points that are in the same support but have different labels is strictly decreasing at each iteration. Moreover, given how the leaves are labeled, the resulting tree is guaranteed to assign $\gamma(x)$ to every data point x. □

[1] Note that the root is by definition a branching node (Definition 2). However, in this procedure, we treat it as a leaf node at the start. Since for any non-trivial dataset this procedure is guaranteed to at least iterate once, we are guaranteed to replace the leaf root node with a branching root node.

3.2.3 Decision Tree Classifiers

To define the problem of classification, we first define the notion of accuracy for a classifier, such as a decision tree, with regard to a labeled dataset, i.e., a dataset in which the correct class label for each data point is provided.

Definition 9 (Classifier Accuracy)**.** *Given a set of features F, integer labels $[1..k]$, a set of training data points $X \subset \mathbb{R}^{|F|}$, and a labelling $\gamma : X \to [1..k]$, the accuracy of a decision tree $\mathcal{D}$ on X is the fraction of data points in X that are correctly classified with respect to the labelling γ and is defined as follows:*

$$\frac{\sum_{x_i \in X} \mathbb{1}[\Theta_{\mathcal{D}}(x_i) = \gamma(x_i)]}{|X|}, \tag{3.5}$$

where the $\mathbb{1}$ function outputs 1 if the given condition is true and 0 otherwise.

The goal for learning decision tree classifiers is to achieve high accuracy in the training data which in most cases translates to high accuracy in the testing data as well. Due to the emphasis on interpretability, objectives related to the compactness of the solution are also of importance.

We consider two problems representing two different objectives for optimal decision trees. Problem 1 consists of finding a decision tree that maximizes the number of correctly classified data points subject to a constraint on the tree depth. This problem is consistent with the problems considered in a variety of recent works, e.g., in [126, 280, 6].

Problem 1 (Max-Accuracy Decision Tree)**.** *Given a set of features F, integer labels $[1..k]$, a set of training data points $X \subset \mathbb{R}^{|F|}$, a labelling $\gamma : X \to [1..k]$, and a chosen depth d, output a fully balanced decision tree $\mathcal{D}$ of the chosen depth, $depth(\mathcal{D}) = d$, with maximum accuracy (Definition 9).*

Problem 2 consists of finding a decision tree with minimum depth that correctly classifies all data points. This problem is consistent with recent work on decision tree classifiers [19].

Problem 2 (Min-Depth Decision Tree)**.** *Given set of features F, integer labels $[1..k]$, a set of training data points $X \subset \mathbb{R}^{|F|}$, and a labelling $\gamma : X \to [1..k]$, output a fully balanced decision tree $\mathcal{D}$ with minimum depth that correctly classifies all data points $\Theta_{\mathcal{D}}(x_i) = \gamma(x_i)\ \forall x_i \in X$, i.e., has an accuracy of 1.*

Unlike the max-accuracy decision tree problem (Problem 1), each feasible solution to the min-depth decision tree problem (Problem 2) must perfectly classify all examples in the training dataset. Theorem 1 shows that for datasets with consistent labeling (Definition 8), there always exists such a decision tree.

3.3 SAT-based Encoding of Decision Tree Classifiers

In this section, we present our SAT-based approach for optimal decision trees. Our encoding can be used to find solutions for both decision tree classification problems. Specifically, it

can be used as the basis for a partial MaxSAT encoding to find a max-accuracy decision tree (Problem 1) or it can be used in searching for increasingly deeper decision trees that can correctly classify all training examples to find a min-depth decision tree (Problem 2).

We propose a SAT encoding of a decision tree, $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$. Similar to previous works (e.g., [19, 280]), our encoding assumes a tree structure $\mathcal{T}$ (typically a fully balanced tree of some depth d), and decides on the values of β, α, and θ. The depth of the tree is given in advance when learning a max-accuracy decision tree. When learning a min-depth decision tree, we solve a sequence of instances for trees of increasing depth until a solution is found.

In Section 3.3.1, we present the variables that are used to encode a decision tree. In Section 3.3.2, we present the hard clauses that form the basis of decision trees in our encoding. In Section 3.3.3, we show how a satisfying boolean assignment to the SAT instance can be decoded into a decision tree. Lastly, in Section 3.3.4 and Section 3.3.5 we describe how the basis of our encoding can be used to solve Problem 1 and Problem 2.

3.3.1 Variables

The following binary variables are used to represent the different aspects of a decision tree, namely the values of β, α, and θ functions for each possible input:

- $\{a_{t,j} \mid t \in \mathcal{T}_B, j \in F\}$: Represents whether feature j is chosen at branching node t. It is equivalent to $\beta(t) = j$.
- $\{s_{i,t} \mid x_i \in X, t \in \mathcal{T}_B\}$: Represents whether point x_i is directed towards the left child, if it passes through branching node t. It is equivalent to $x_i[\beta(t)] \leq \alpha(t)$.
- $\{z_{i,t} \mid x_i \in X, t \in \mathcal{T}_L\}$: Represents whether point x_i ends up at leaf node t.
- $\{g_{t,c} \mid t \in \mathcal{T}_L, 1 \leq c \leq k\}$: Represents whether label c is assigned to leaf node t. It is equivalent to $\theta(t) = c$.

Note that the number of $a_{t,j}$ variables is in $O(|F||\mathcal{T}_B|)$, $s_{i,t}$ in $O(|X||\mathcal{T}_B|)$, $z_{i,t}$ in $O(|X||\mathcal{T}_L|)$, and $g_{t,c}$ in $O(k|\mathcal{T}_L|)$, making the total number of variables to be in $O(|X||\mathcal{T}_L|)$ given that $|X| \geq |F|$.

3.3.2 Clauses

The following set of hard clauses in CNF guarantee the validity of the recursion in the modelled decision tree, and can consequently be used as the encoding basis for both of the optimization problems we consider. The operator $\bigvee$ is used as a shorthand for the inclusion of a set of literals in the disjunctive clause.

The clauses in Eq. (3.6) and Eq. (3.7) guarantee that exactly one feature is chosen at each branching node $t \in \mathcal{T}_B$.

$$(\neg a_{t,j}, \neg a_{t,j'}) \qquad t \in \mathcal{T}_B, j \neq j' \in F \tag{3.6}$$

$$(\bigvee_{j \in F} a_{t,j}) \qquad t \in \mathcal{T}_B \tag{3.7}$$

For each branching node $t \in \mathcal{T}_B$, we need to make sure that all the data points x_i for which the corresponding feature value $x_i[\beta(t)]$ is less than or equal to the threshold $\alpha(t)$ are directed left and all the data points for which the feature value is greater than the threshold are directed right. We use $\#_j^i$ to denote the index of the i'th data point in X when sorted by feature j in ascending order, assuming ties are broken arbitrarily, and define O_j to be the set of all consecutive pairs in this ordering, i.e., $O_j = \{(\#_j^i, \#_j^{i+1}) \mid 1 \leq i \leq |X|-1\}$. Then, the clauses in Eq. (3.8) guarantee that there are no two points with different feature values where the one with the higher value is directed left while the one with the lower value is directed right. The clauses in Eq. (3.9), together with the clauses in Eq. (3.8), guarantee that points with equal values are directed in a similar manner. The two sets of constraints together guarantee the existence of a valid threshold value $\alpha(t)$, despite the lack of directly including it in the encoding.

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F, (i,i') \in O_j(X) \tag{3.8}$$

$$(\neg a_{t,j}, \neg s_{i,t}, s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F, (i,i') \in O_j(X), x_i[j] = x_{i'}[j] \tag{3.9}$$

For each data point x_i, we need to guarantee the validity of its path in the decision tree. We use $A_l(t)$ (respectively $A_r(t)$) to denote all the ancestors of a leaf node $t \in \mathcal{T}_L$ such that t is a descendant of their left (respectively right) branch. The clauses in Eq. (3.10) and Eq. (3.11) guarantee that each data point that ends up at a leaf node follows the corresponding path. In contrast, the clauses in Eq. (3.12) guarantee that each data point that does not end up in leaf node $t \in \mathcal{T}_L$ has at least one deviation from the corresponding path

$$(\neg z_{i,t}, s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_l(t) \tag{3.10}$$

$$(\neg z_{i,t}, \neg s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_r(t) \tag{3.11}$$

$$(z_{i,t}, \bigvee_{t' \in A_l(t)} \neg s_{i,t'}, \bigvee_{t' \in A_r(t)} s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{3.12}$$

The clauses in Eq. (3.13) guarantee that each leaf node is assigned at most one label. Note that we do not include constraints that prevent leaves with no label. The optimization criteria in Problems 1 and 2 guarantee that in optimal solutions, all leaves that have corresponding training examples will be assigned a label.

$$(\neg g_{t,c}, \neg g_{t,c'}) \qquad t \in \mathcal{T}_L, c \neq c' \in [1..k] \tag{3.13}$$

Finally, we add redundant constraints that help prune the search space. For each branching node, the clauses in Eq. (3.14) guarantee that the data point with the lowest feature value is directed left and the clauses in Eq. (3.15) guarantee that the data point with the highest feature value is directed right.

$$(\neg a_{t,j}, s_{\#_j^1,t}) \qquad t \in \mathcal{T}_B, j \in F \tag{3.14}$$

$$(\neg a_{t,j}, \neg s_{\#_j^{|X|},t}) \qquad t \in \mathcal{T}_B, j \in F \tag{3.15}$$

Note that the number of clauses introduced in Eq. (3.6) and Eq. (3.7) is in $O(|F|^2|\mathcal{T}_B|)$, in Eq. (3.8) and Eq. (3.9) is in $O(|F||X||\mathcal{T}_B|)$, in Eq. (3.10) to Eq. (3.12) is in $O(d|X||\mathcal{T}_L|)$, in Eq. (3.13) is in $O(k^2|\mathcal{T}_L|)$, and in Eq. (3.14) and Eq. (3.15) is in $O(|F||\mathcal{T}_B|)$, respectively. In total, the number of clauses is in $O(|X||\mathcal{T}_L|)$ if $|X| \geq |F| + d + k$. Contrarily, if we employ explicit binarization of features, we can end up with as many as $|F||X|$ features, requiring $O(|X|^2|F|^2|\mathcal{T}_B|)$ clauses to guarantee the validity of feature selection. This blowup in size is common with SAT-based approaches from the literature.

3.3.3 Decoding

Assuming a solution to the SAT encoding, i.e., an assignment to the variables in Section 3.3.1 that satisfy the clauses in Section 3.3.2, we now describe how to extract the feature selection, threshold selection, and leaf labelling functions to make up the decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ alongside the assumed structure.

Decoding β is done by setting $\beta(t) = j$ if the variable $a_{t,j}$ is true. Similarly, decoding θ is done by setting $\theta(t) = c$ if the variable $g_{t,c}$ is true. Note that these procedures are valid since it is guaranteed that at most one $a_{t,j}$ and $g_{t,c}$ variable is set to true for each node, specifically, $\forall t \in \mathcal{T}_B : \sum_{j \in F} a_{t,j} = 1$ and $\forall t \in \mathcal{T}_L : \sum_{c \in [1..k]} g_{t,c} \leq 1$. A leaf node $\hat{t}$ that is assigned no labels, i.e., $\forall c : g_{\hat{t},c} = 0$, will be given an arbitrary label $\hat{c}$ by setting $\theta(\hat{t}) = \hat{c}$ in order to maintain a valid decision tree. Note that the arbitrary assignment does not impact the objective as the unassigned leaf cannot contain any training data points.

Since our SAT encoding does not explicitly compute the threshold for each node, to decode α we have to choose a threshold based on the direction of the data points in each branching node. For a branching node $t \in \mathcal{T}_B$ with $\beta(t) = j$, we set $\alpha(t) = x_i[j]$ where $(i, i') \in O_j(X)$ are consecutive data points according to the ordering of feature j such that $s_{i,t}$ is true and $s_{i',t}$ is false. This pair of points always exists because of the pruning clauses in Eq. (3.14) and Eq. (3.15). Specifically for the binary features, it means that the points with value 0 (1) are always directed left (right). Intuitively, we are using the largest value in the entire dataset directed left as the feature threshold for the node t. Note that the choice is not unique as we can select any value starting from that of the last point directed left to the first point directed right. The selected threshold can impact testing accuracy while maintaining training accuracy. Alternatively, we can only consider the subset of data contained in the branching node, which can be determined in the decoding stage. Doing so

would require a more thorough strategy to select the threshold as the gap between the last value directed to left and the first value directed right is larger.

3.3.4 Max-Accuracy Objective

To find a decision tree of a given depth that maximizes the number of correctly classified training examples, we introduce the following variables to keep track of training examples that are correctly classified:

- $\{p_i \mid x_i \in X\}$: Represents whether point x_i ends up in a leaf node with the same label, i.e., is correctly classified. It is equivalent to $\Theta(x_i) = \gamma(x_i)$.

We use a partial MaxSAT model that includes both hard and soft clauses. We use all the clauses from the basis of our decision tree encoding in Section 3.3.2 as hard clauses. We also add the hard clauses in Eq. (3.16) that guarantee p_i is set to true only when x_i ends up in a leaf node whose label is consistent with the training label.

$$(\neg p_i, \neg z_{i,t}, g_{t,\gamma(x_i)}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{3.16}$$

In order to maximize the number of correctly classified training examples, we add the soft clauses in Eq. (3.17), so that each correctly classified data point contributes a value of 1 to the objective value.

$$(p_i) \qquad x_i \in X \tag{3.17}$$

Therefore, the optimal solution for this partial MaxSAT model is a decision tree that maximizes the number of correctly classified training examples. The number of added variables and clauses (Eq. (3.16) and Eq. (3.17)) for the max-accuracy objective is in $O(|X|)$ and $O(|\mathcal{T}_L||X|)$, respectively.

3.3.5 Min-Depth Objective

To find a minimum-depth decision tree such that all training examples are correctly classified, we add the following hard clauses to the basis of our decision tree encoding described in Section 3.3.2:

$$(\neg z_{i,t}, g_{t,\gamma(x_i)}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{3.18}$$

The clauses in Eq. (3.18) guarantee that the class labels assigned to leaf nodes are consistent with the training set labels of the corresponding data points. If a data point x_i ends in leaf node $t \in \mathcal{T}_L$ then the assigned label of t must match the training label $\gamma(x_i)$. The number of added clauses (Eq. (3.18)) for the min-depth objective is in $O(|\mathcal{T}_L||X|)$.

In order to guarantee that we find the minimum depth decision tree, we follow the technique prevalent in the literature [19]. We start by solving the SAT formula for a tree structure $\mathcal{T}$ of depth 1 and in each iteration increase the depth by 1 until a solution is found. This guarantees that when the SAT solver finds a solution, the obtained decision tree has a minimum depth subject to the constraint that all training examples must be classified correctly.

3.4 Extending Decision Trees to Categorical Features

Categorical features are features that take their value from a set of values representing different categories that do not induce a natural ordering.[2] For example, in a dataset of financial transactions we can have a feature that describes the type of transaction from a set of known transaction types. In order to deal with a categorical feature with K categories, one can convert the feature to K binary features representing a one-hot encoding of the original categorical feature. However, this can lead to unnecessarily deep decision trees as splits may be required for many categories.

In this section, we show that we can easily extend our approach to generate decision trees that support branching on categorical features directly. We employ *power set branching* in which a selected subset of the categories is assigned to the left branch while the subset that contains the rest of the categories is assigned to the right branch.

Definition 10 (Power Set Branching)**.** *Given a finite set of categorical features F_C, a power set branching is a division of data points into two groups. A power set branching operates on a categorical feature $f_c \in F_C$ and a subset of its corresponding categories $a \subseteq dom(f_c)$. For $x[f_c]$ which represents the value of categorical feature f_c for data point x, belonging to the dividing subset ($x[f_c] \in a$) equates to a binary value of* 1 *while not belonging ($x[f_c] \notin a$) equates to* 0*.*

We show how decision trees can be extended to support categorical features and power set branching. We assume a set of features F that includes categorical features F_C and numerical features F_N such that $F = F_C \cup F_N$. In order to support power set branching, we augment the decision tree with a category subset selector α_C, $\mathcal{D}_C = (\mathcal{T}, \beta, \alpha, \alpha_C, \theta)$. We note that the set of branching nodes are divided into those associated with numerical features $\beta(t) \in F_N$ and those associated with categorical features $\beta(t) \in F_C$. The first and second sets are the domains of functions α and α_C, respectively. A well-formed α_C function respects the categories of the corresponding feature, i.e., $\forall t \in \mathcal{T}_B : (\beta(t) \in F_C) \rightarrow (\alpha_C(t) \subseteq dom(\beta(t)))$.

Next, we show how power set branching can be integrated into the recursive label assignment function. The new function, Θ_C, operates recursively as follows:

[2] Some categorical features induce a natural ordering and can therefore be represented as numerical features. For example, a categorical feature with the categories {Low, Medium, High} can be transformed to a numerical feature with the values {1, 2, 3}.

$$\Theta_C(t, x_i) = \begin{cases} \theta(t) & \text{if } t \in \mathcal{T}_L \\ \Theta_C(l(t), x_i) & \text{else if } (\beta(t) \in F_N \wedge x_i[\beta(t)] \leq \alpha(t)) \vee \\ & \qquad (\beta(t) \in F_C \wedge x_i[\beta(t)] \in \alpha_C(t)) \\ \Theta_C(r(t), x_i) & \text{otherwise} \end{cases} \tag{3.19}$$

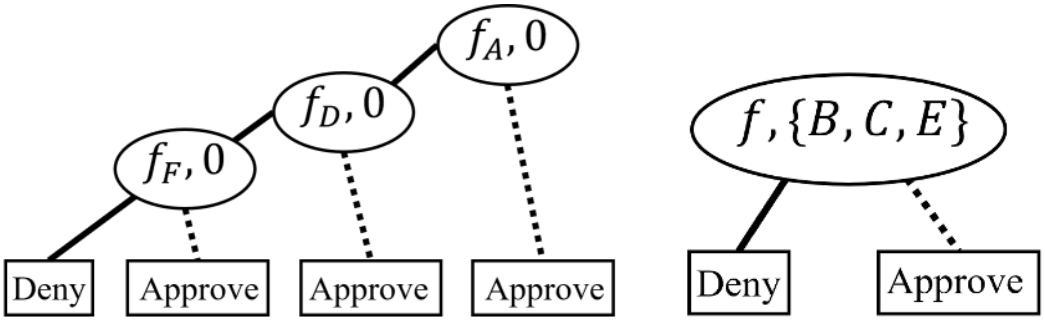
3.4.1 Compactness of Power Set Branching

For the min-depth optimal decision tree problem, augmentation with power set branching can lead to decision trees of smaller depth that can potentially be found earlier. For the max-accuracy optimal decision tree problem, such augmentation can allow more flexible trees that achieve higher accuracy for the same depth. In the following example, we demonstrate how a decision tree employing power set branching can be arbitrarily more compact than an equivalent tree that uses one-hot encoding for categorical features.

Example 1. *To demonstrate the potential benefit of power set branching, consider a simplified dataset of transactions that can either be approved or denied. Each transaction has one categorical feature describing the transaction type with the categories* $\{A, B, C, D, E, F\}$*. Transaction with types* A*,* D*, and* F *are approved, while transactions with types* B*,* C*, and* E *are denied. Figure 3.2a shows a standard decision tree where the categorical feature was converted to six binary features representing a one-hot encoding of the original feature. Figure 3.2b shows the extended variant of decision trees where we can branch directly on categorical features by assigning a subset of the categories to each branch. Feature* f *is the transaction type feature and* f_A*,* f_D*, and* f_F *are one-hot encodings corresponding to values* A*,* D*, and* F*. The standard decision tree requires branching, in sequence, on multiple different categories leading to a tree with a minimum depth of* 3*. In contrast, the extended decision tree that supports branching on categorical features has a depth of* 1*. Similar examples can be synthesized where there are* $2n$ *categories in total and half of them lead to approval. Such examples can show a depth disparity of* 1 *to* n*.*

3.4.2 Encoding of Power Set Branching

Interestingly, the proposed extension involves only minor changes to the decision tree encoding in Section 3.3. Eq. (3.20-3.30) show the modified encoding with the changes highlighted in blue. The clauses in Eq. (3.22) guarantee that data points where the feature value is less or equal to the feature threshold are directed left and vice versa. As this does not apply to categorical features, we restrict Eq. (3.22) to numerical features. Instead, we add Eq. (3.24) that, together with the existing Eq. (3.23), guarantees that data points with the same category are directed in the same direction. Finally, for numerical features we used redundant constraints that guarantee the lowest feature value is directed left and the highest feature value is directed right. For categorical features we do not have such ordering



(a) One-hot encoding. Branching nodes are labelled with $(\beta(t), \alpha(t))$ values.

(b) Power set branching. Branching nodes are labelled with $(\beta(t), \alpha_C(t))$ values.

Figure 3.2: Example equivalent decision trees with and without power set branching for categorical features. Solid (dotted) lines correspond to the left (right) child.

over categories and instead we simply use the clauses in Eq. (3.29) to make sure that some arbitrary category is chosen to be directed left (in categorical features, $\#_j$ represents an arbitrary ordering over the category values of feature j) and we modify Eq. (3.30) such that no value for a categorical feature is forced to go right.[3] Note that unlike previous work that focused on categorical features [111], our approach directly encodes optimal decision trees with both numerical and categorical features.

$$(\neg a_{t,j}, \neg a_{t,j'}) \qquad t \in \mathcal{T}_B, j \neq j' \in F \tag{3.20}$$

$$(\bigvee_{j \in F} a_{t,j}) \qquad t \in \mathcal{T}_B \tag{3.21}$$

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F_N, (i,i') \in O_j(X) \tag{3.22}$$

$$(\neg a_{t,j}, \neg s_{i,t}, s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F, (i,i') \in O_j(X), x_i[j] = x_{i'}[j] \tag{3.23}$$

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F_C, (i,i') \in O_j(X), x_i[j] = x_{i'}[j] \tag{3.24}$$

$$(\neg z_{i,t}, s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_l(t) \tag{3.25}$$

$$(\neg z_{i,t}, \neg s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_r(t) \tag{3.26}$$

$$(z_{i,t}, \bigvee_{t' \in A_l(t)} \neg s_{i,t'}, \bigvee_{t' \in A_r(t)} s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{3.27}$$

$$(\neg g_{t,c}, \neg g_{t,c'}) \qquad t \in \mathcal{T}_L, c \neq c' \in [1..k] \tag{3.28}$$

$$(\neg a_{t,j}, s_{\#_j^1,t}) \qquad t \in \mathcal{T}_B, j \in F \tag{3.29}$$

$$(\neg a_{t,j}, \neg s_{\#_j^{|X|},t}) \qquad t \in \mathcal{T}_B, j \in F_N \tag{3.30}$$

[3] Note that we could add clauses that guarantee that at least one category will go to the right, however it does not provide significant pruning and we therefore opted not to add them.

Decoding of Power Set Branching

Decoding a decision tree extended with power set branching follows the same procedure as described in Section 3.3.3, with one key difference. For nodes that are associated with categorical branching, we need to decode α_C, the category subset selector that indicates the subset of categories directed towards the left child. For a categorical branching node $\beta(t) = j \in F_C$, we define $\alpha_C(t)$ to be the set of categories in feature j for which there are data points in X that are directed left,

$$\alpha_C(t) = \{c \mid \exists x_i \in X : x_i[\beta(t)] = c \wedge s_{i,t} = 1\}. \tag{3.31}$$

Note that unlike our encoding for purely numerical decision trees, our encoding for power set branching allows degenerate nodes for which all categories are directed left with none of the categories directed right. An optimal solution may have degenerate nodes, however we can easily convert such solutions to optimal solutions without degenerate nodes. As a post-optimization step, we arbitrarily select a subset of categories from the left branch and move them to the right branch. Then, we copy all the subtree of the left branch to the right branch, leading to identical predictions on the training examples and maintaining the optimality of the original solution.[4]

3.4.3 Expressiveness of Power Set Branching

Decision trees with power set branching are more expressive than decision trees that use a binarized one-hot encoding of categorical features. Specifically, each decision tree that uses binary branching for categorical features can be transformed into a decision tree with power set branching with identical predictions.

Proposition 1. *Given a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ operating on a set of features $bin(F)$, there exists a decision tree with power set branching on the same structure $\mathcal{D}'_C = (\mathcal{T}, \beta', \alpha, \alpha'_C, \theta)$ operating on the set of features F such that*

$$\forall x_i : \Theta_C(x_i) = \Theta(bin(x_i)) \tag{3.32}$$

where $bin(F)$ is F with its categorical features encoded using binary features in one-hot style and $bin(x_i)$ is x_i with its values encoded according to the binarized feature set $bin(F)$.

Proof. Proposition 1 holds since each branching node in the decision tree $\mathcal{D}$ associated with a binary feature that corresponds to one category c in a categorical feature $j \in F$ can be replaced in $\mathcal{D}'$ with a node that branches directly on the feature j and directs the subset of categories $dom(j) - \{c\}$ to the left and, as a result, $\{c\}$ to the right. □

Note that Proposition 1 does not hold in the opposite direction. Specifically, transforming a decision tree with power set branching into one using binarized features is possible but

[4] Note that we can similarly replace any degenerate node with its left child, however the resultant tree will not be a fully balanced tree (Definition 6).

it may (and often will) require a deeper tree (see Example 1). Furthermore, Proposition 1 implies the following corollaries on the quality guarantees of solutions employing power set branching with regard to each of the optimization problems.

Corollary 1. *Given an instance of Problem 2, the minimum depth found for a decision tree with power set branching is always equal to or less than that of a decision tree with branching based on binarized encoding of categorical features.*

Corollary 2. *Given an instance of Problem 1, the maximum accuracy found for a decision tree with power set branching is always equal to or more than that of a decision tree with branching based on binarized encoding of categorical features.*

Due to brevity, we will continue to work with the standard decision tree definition without power set branching throughout this dissertation. Unless noted otherwise, all of the extensions and applications of decision trees (and binary decision diagrams) in the upcoming sections and chapters are compatible with power set branching in a straightforward manner.

3.5 Cost-sensitive Learning

In many practical applications, the cost of misclassification may not be uniform across classes. For example, in medical diagnosis, the cost of erroneously diagnosing a patient to be healthy may be much higher than that of mistakenly diagnosing a healthy person as being sick [311]. Furthermore, in datasets with imbalanced class distribution in the training set, a uniform misclassification cost will lead to increased emphasis on correctly predicting the dominant class [192]. However, in many classification tasks (e.g., fraud detection and medical diagnosis), the rare classes are those of primary interest.

Cost-sensitive learning allows us to assign different misclassification costs across data points or classes. In particular, it has been used to mitigate the problem of imbalanced datasets by exploiting the fact that the value of correctly identifying the rare classes outweighs the value of correctly identifying the common class [289]. For example, in two-class problems this can be done by associating a greater cost with false negatives than with false positives [289].

In this section, we define cost-sensitive optimal decision trees (Problem 3) and present an extension of our SAT-based approach for learning cost-sensitive optimal decision trees.

Problem 3 (Cost-Sensitive Optimal Decision Tree)**.** *Given a set of features F, integer labels $[1..k]$, a set of training data points $x_i \in X$, a labelling $\gamma : X \to [1..k]$, a chosen depth d, and a valid* [5] *cost matrix $\mathcal{C} \in \mathbb{R}_{\geq 0}^{k \times k}$ such that $\mathcal{C}[c_1, c_2]$ is the cost of assigning the label c_2 to a data point x_i with training label $\gamma(x_i) = c_1$, output a fully balanced decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ of the depth that is given $depth(\mathcal{T}) = d$, such that $\mathcal{D}$ minimizes the classification cost defined by the cost matrix $\mathcal{C}$, i.e., $\sum_{x_i \in X} \mathcal{C}[\gamma(x_i), \Theta(x_i)]$.*

[5] A cost matrix $\mathcal{C}$ is valid if $\forall c : \mathcal{C}[c, c] = 0$ and $\forall c_1 \neq c_2 : \mathcal{C}[c_1, c_2] > 0$

Proposition 2 shows that max-accuracy optimal decision trees (Problem 1) are a special case of cost-sensitive optimal decision trees (Problem 3).

Proposition 2. *Given a set of features F, integer labels $[1..k]$, a set of training data points $x_i \in X$, a labelling $\gamma : X \to [1..k]$, a chosen depth d, and the uniform misclassification cost matrix $\mathcal{C}_\mathcal{U}$ such that*

$$\forall c_1, c_2 \in [1..k] : \mathcal{C}_\mathcal{U}[c_1, c_2] = \begin{cases} 1 & \text{if } c_1 \neq c_2 \\ 0 & \text{if } c_1 = c_2, \end{cases}$$

we have that decision tree $\mathcal{D}$ is an optimal solution to Problem 1 $\iff$ $\mathcal{D}$ is an optimal solution to Problem 3.

Proof. Proposition 2 is correct since the objective of Problem 3 with cost matrix $\mathcal{C}_\mathcal{U}$, that is $\sum_{x_i \in X} \mathcal{C}_\mathcal{U}[\gamma(x_i), \Theta(x_i)]$, is equal to the number of misclassified data points, $\sum_{x_i \in X} \mathbb{1}[\gamma(x_i) \neq \Theta(x_i)]$. Thus, the cost-sensitive objective is equivalent to maximizing the number of correctly classified data points and the two problems have identical optimal solutions. □

We note that costs cannot be incorporated into min-depth optimal decision trees (Problem 2). Since min-depth optimal decision trees perfectly classify all data points, we have that $\Theta(x_i) = \gamma(x_i)\ \ \forall x_i \in X$ and the cost of all feasible solutions is the same ($\sum_{x_i \in X} \mathcal{C}[\gamma(x_i), \gamma(x_i)]$) and is equal to zero.

3.5.1 Encoding of Cost-Sensitive Objectives

To find a decision tree of a given depth that minimizes the classification cost according to a cost matrix $\sum_{x_i \in X} \mathcal{C}[\gamma(x_i), \Theta(x_i)]$, we introduce the following variables to keep track of the label assigned to each data point.

- $\{\hat{p}_{i,c} \mid x_i \in X, 1 \leq c \leq k\}$: Represents whether label c is assigned to point x_i. It is equivalent to $\gamma(x_i) = c$.

In contrast to our encoding of max-accuracy optimal decision tree, we use a *weighted* partial MaxSAT model where each soft clause is associated with a weight. We use all the clauses from the basis of our decision tree encoding in Section 3.3.2 as hard clauses. We add the hard clauses in Eq. (3.33) that guarantee the $\hat{p}_{i,c}$ variables correctly represent the labels assigned to each point.

$$(\neg\hat{p}_{i,c}, \neg z_{i,t}, g_{t,c}) \qquad\qquad t \in \mathcal{T}_L, x_i \in X, c \in [1..k] \tag{3.33}$$

We also add the hard clauses in Eq. (3.34) that guarantee at least one label to be assigned to each point.

$$(\bigvee_{c \in [1..k]} \hat{p}_{i,c}) \qquad\qquad x_i \in \mathcal{X} \tag{3.34}$$

To learn cost-sensitive decision-tree classifiers, we add the soft clauses in Eq. (3.35). Each clause corresponding to data point x_i and label c is satisfied when c is *not* assigned to x_i. We set the weight of each clause (in angle brackets) to be the cost associated with classifying point x_i with label c according to the cost matrix, i.e., $\mathcal{C}[\gamma(x_i), c]$. By maximizing the cost associated with all the classes that we do not assign x_i to, we are effectively minimizing the weighted misclassification cost in Problem 3.

$$\langle \mathcal{C}[\gamma(x_i), c] \rangle \qquad (\neg \hat{p}_{i,c}) \qquad \forall x_i \in X, c \in [1..k] \tag{3.35}$$

3.6 Tree Pruning Constraints

Previous works on both heuristic and optimal decision trees have considered different tree pruning constraints, such as maximum depth and minimum support constraints, in order to produce trees that are less prone to overfitting and tend to generalize better to unseen datasets [165, 37, 111, 280, 6].

Our approach for max-accuracy optimal decision trees inherently includes a constraint on the maximum depth of decision trees based on the provided structure $\mathcal{T}$. In this section, we further present two types of pruning constraints and show how they can be incorporated into our SAT-based encoding for optimal decision trees.

We note that while these constraints can be used in conjunction with both max-accuracy and min-depth decision trees, we only consider them in the context of max-accuracy decision trees. As these constraints prune the space of feasible decision trees they may render min-depth decision trees, that are designed to perfectly fit the training data, infeasible. For max-accuracy decision trees, these constraints may increase the cost of optimal solutions, however the obtained trees are expected to generalize better to unseen data.

3.6.1 Minimum Support Constraints

The support of a decision tree is the smallest number of points that are contained amongst any of its leaf nodes (Definition 7). Minimum support constraints guarantee the support to be larger or equal to a minimum [280, 6, 37]. We show how minimum support constraints are incorporated into our SAT encoding for optimal decision trees.

To impose a minimum support of S for a training dataset X, we use the sequential counter cardinality encoding [256] on the number of points that are contained at each leaf. Namely, we enforce, via a set of new clauses and auxiliary variables, that the set of variables $\{z_{i,t} | x_i \in X\}$ for each leaf node t, should have at least S variables set to true:

$$\forall t \in \mathcal{T}_L : \Psi(t) = \sum_i \#z_{i,t} \geq S \tag{3.36}$$

where $\#z_{i,t}$ represents the numerical interpretation of the Boolean variable $z_{i,t}$. The clauses

produced as the result of the sequential counter encoding will then be added to the set of hard clauses in our SAT encoding.

3.6.2 Minimum Margin Constraints

The margin of a branching (Definition 1) with regard to a dataset X is the smaller number of data points that it assigns a value of 0 or 1. Minimum margin constraints guarantee that the margins for all branchings employed in a decision tree satisfy a lower bound. In other words, for each branching node and assuming that all of X is contained in that node, a minimum number of data points are directed to both left and right children. We formally define the notion of margin and then show how minimum margin constraints are incorporated into our SAT encoding of optimal decision trees.

Definition 11 (Branching Node Margin)**.** *Given a dataset X, the margin of a branching on feature f and threshold a is defined as the size of the smaller half in the division of the points $min(\sum_{x_i \in X} \mathbb{1}[x_i[f] \leq a], \sum_{x_i \in X} \mathbb{1}[x_i[f] > a])$. The margin of a branching node $\Lambda(t)$ is defined as the margin of the branching that it employs and the margin of a decision tree $\Lambda(\mathcal{D})$ is the smallest margin amongst its branching nodes.*

Note that in practice not all data points will arrive at each branching node. Therefore, we consider margin as a less precise notion than support. It is a weaker constraint in most cases and a more strict one when only a few points with marginal values are contained in the node. However, it is simpler to implement and involves no additional variables or clauses (see discussion in Section 3.10).

To impose a minimum margin of M on our encoding, note that our Eq. (3.14) and Eq. (3.15) set of clauses are enforcing at least one data point to be directed towards left and one towards right. Equivalently, they are imposing a margin of 1. Thus, we can impose larger margins by simply strengthening these clauses.

$$(\neg a_{t,j}, s_{\#_j^M,t}) \qquad t \in \mathcal{T}_B, j \in F \tag{3.37}$$

$$(\neg a_{t,j}, \neg s_{\#_j^{|X|-M+1},t}) \qquad t \in \mathcal{T}_B, j \in F \tag{3.38}$$

The clauses in Eq. (3.37) guarantee at least M points to be directed towards left and the clauses in Eq. (3.38) guarantee at least M to be directed towards right, under the assumption that all of X is contained in branching node t. We note that this approach is not compatible with categorical branching due to the fact that the category values are not ordered.

3.6.3 Relation Between Support and Margin

In Proposition 3, we establish a theoretical connection between the support and margin of a decision tree.

Proposition 3. *Given a dataset X and a decision tree $\mathcal{D}$, the support is smaller or equal to the margin $\Psi(\mathcal{D}) \leq \Lambda(\mathcal{D})$.*

Proof. In order to prove by contradiction, we assume $\Psi(\mathcal{D}) > \Lambda(\mathcal{D})$.

Let $t_B \in \mathcal{T}_B$ be the branching node that achieves the minimum margin, namely $\Lambda(\mathcal{D}) = \Lambda(t_B)$. We can state, without loss of generality, that t_B guides $\Lambda(t_B)$ points to its right child t_r. Thus, $|\Psi(t_r)| \leq \Lambda(t_B)$.

Finally, assume t_l to be any of the leaves that have t_r as their ancestor (possibly t_r itself). By definition we have $|\Psi(t_l)| \leq |\Psi(t_r)|$. Which results in the following contradiction.

$$
\begin{aligned}
|\Psi(t_l)| &\leq |\Psi(t_r)| \\
|\Psi(t_l)| &\leq \Lambda(t_B) \\
|\Psi(t_l)| &\leq \Lambda(\mathcal{D}) \\
|\Psi(t_l)| &< \Psi(\mathcal{D})
\end{aligned}
$$

which is impossible since $\Psi(\mathcal{D})$ is defined as the minimum support amongst all leaves. Thus, the support of a decision tree is always smaller or equal to its margin. □

3.7 Experimental Setup

In this section, we describe the setup for running our experiments in terms of tools, data, and similar approach that we intend to use as baselines to compare against.

3.7.1 Implementation

Our encoding for the max-accuracy optimal decision tree problem is implemented in Java and solved using the MaxSAT solver Loandra [34]. Our algorithm for the min-depth optimal decision tree problem is implemented in C++ based on SAT solver MiniSAT [81]. The experiments were run on two 12-core Intel E5-2697v2 CPUs and 128G of RAM.

3.7.2 Baselines

For the max-accuracy optimal decision tree problem (Problem 1), we compare our approach to the following recent state-of-the-art baselines that optimize the accuracy of the obtained tree over a set of training data points for a given tree depth:

- Hu et al.'s [126] MaxSAT encoding for optimal decision trees based on the SAT encoding in Narodytska et al. [209].[6] Similar to our approach, their MaxSAT encoding is solved by the Loandra solver [34].

[6] Code obtained from `https://gepgitlab.laas.fr/hhu/maxsat-decision-trees`.

- Aglin, Nijssen, and Schaus's [6] DL8.5, an approach for optimal decision trees based on itemset mining.[7] They use specialized branch-and-bound algorithm that makes use of a novel caching technique.

For the min-depth optimal decision tree problem (Problem 2), we compare our approach to the recent Avellaneda's SAT-based approach [19], that, similar to our approach solves SAT encodings for increasingly deep decision trees until a tree that perfectly classifies all instances is found. Similar to our implementation for the min-depth problem, Avellaneda uses the MiniSAT solver [81].[8]

Note that other approaches, such as OCT [37], BinOCT [281], DL8 [216], Narodytska et al.'s SAT model [209], and Bessiere, Hebrard, and O'Sullivan's SAT model [39], have been previously compared to one or more of the above baselines in their original publications [126, 19, 280, 6, 19].

All of the selected baselines, and more generally most of the recent state-of-the-art approaches for optimal decision trees, are designed for datasets that contain only binary features. This is particularly true when SAT is used for learning decision trees, as it is a boolean-based paradigm in nature. In order to learn decision-tree classifiers for datasets that have numerical or categorical features, most approaches must first convert any non-binary feature into a set of binary features in a pre-processing step prior to optimization. Following Avellaneda [19], each non-binary feature that can take v different categorical values is converted to v binary features (one-hot encoding [228]), one for each possible value. Furthermore, if the feature is ordered, e.g., numerical features or ordinal features, then it will be converted to $v-1$ binary features that represent the operator $\leq$ as described in the following example based on Avellaneda [19]:

Example 2. *Consider a set of training examples where each example has a single integer feature f, {(1), (3), (4), (5)}. Then, we can transform f into three binary features $\tilde{f}_0, \tilde{f}_1, \tilde{f}_2$. If the feature $\tilde{f}_0$ is true it means that the value of f in the data point is greater than 1. If the feature $\tilde{f}_1$ is true it means that the value of f in the example is greater than 3, etc. Therefore, the transformed dataset will be {(0,0,0), (1,0,0), (1,1,0), (1,1,1)}.*

Note that this conversion to binary features is compatible with the binarization procedure of numerical features in a branching (Definition 1). Specifically, selecting a numerical feature and threshold for a branching is equivalent to selecting the corresponding binary feature from the result of the conversion.

3.7.3 Real Datasets

We run experiments on 15 real datasets with different characteristics from the UCI repository [77]. Table 3.1 reports the statistics for each dataset, including the number of data points $|X|$, the number of class labels k, the number of numerical features $|F_N|$, the number of

[7] Code obtained from `https://github.com/aglingael/dl8.5`.
[8] Code obtained from `https://github.com/FlorentAvellaneda/InferDT`.

binary features[9] $|F_B|$, and the number of categorical features $|F_C|$. It also reports the number of binary features in the dataset used by the baselines $\tilde{f}$, i.e., after all the numerical and categorical features have been converted into binary features.

Type	Name	$\|X\|$	$\|F_N\|$	$\|F_B\|$	$\|F_C\|$	$\tilde{f}$	k
N	Banknote	1372	4	0	0	5016	2
	Breast Cancer	116	9	0	0	891	2
	Cryotherapy	90	5	1	0	93	2
	Immunotherapy	91	6	1	0	166	2
	Ionosphere	351	32	2	0	8114	2
	Iris	150	4	0	0	119	3
	User Knowledge	258	5	0	0	431	4
	Vertebral Column	310	6	0	0	1741	2
	Wine	178	13	0	0	1263	3
C	Credit Approval†	653	6	4	5	1130	2
	Promoter	106	0	0	57	228	2
	Soybean Large†	266	5	16	14	76	15
	Protease Cleavage	746	0	0	8	160	2
	Protease Cleavage(/4)	186	0	0	8	156	2
B	Car‡	1728	6	0	0	15	2
	Monk2	169	4	2	0	11	2

Table 3.1: Description of the datasets used in our experiments. Records with missing values were removed, indicated with †. Classes were merged following Avellaneda [19], indicated with ‡.

Consistent with our focus on non-binary features, we selected datasets with mostly numerical features (type N) or categorical features (type C). Our decision on whether to consider a feature as categorical was made based on the feature description and the type of values (e.g. string or integer). We also included the datasets Monk2 and Car that we consider binary (type B) since they have either binary features or numerical features that only take a small number of different values, i.e., can be converted to a small number of binary features.

The datasets Credit Approval, Promoter, Soybean Large, and Protease Cleavage include a significant number of categorical features and will be used to evaluate our extension for optimal decision trees with categorical features. We also created a smaller version of Protease Cleavage that includes only 25% of the records (by selecting each fourth row).

Feature Interpretability

As discussed in Chapter 2, interpretability of a ML model is highly reliant on the interpretability of the features that it utilizes. For example, a decision tree with a single branching on the output of a complex black-box model has a simple structure, but is not interpretable

[9] In our encoding binary features also include numerical features that take one of two possible values, however we list them separately in Table 3.1 as they are supported by the baseline methods without transformation.

in any way. To maintain interpretability, we should limit feature utilization to those that are direct aspects of the entities at hand, or those that are carefully engineered based on domain-specific knowledge or an automated approach [206]. The features from the UCI repository datasets have been studied in terms of being informative and their impact on accuracy [12, 8]. Particularly, our selected group mostly focuses on cases where there are a sparse number of features each meaningful in their own domain (e.g., Breast Cancer [223], Iris [273], Wine [2], and Car [45]), yielding interpretable models as a result.

3.7.4 Synthetic Datasets

In addition to real datasets, we generated random numerical datasets following the approach described in Guyon [114] and implemented in the library `scikit-learn` [224]. The parameters of the dataset generator include the number of data points $|X|$, the number of informative features $|F^i|$, the number of redundant features $|F^d|$, the number of repeated features $|F^r|$, the noise level η, and the number of classes k. Each class is composed of two Gaussian clusters representing the informative features. Redundant features are linear combinations of informative features, and repeated features are randomly duplicated from the informative and redundant features. The noise is introduced by randomly assigning the class label for a small fraction (η) of the data points.

We created 13 synthetic datasets with different characteristics as described in Table 3.2. Synth1 is generated using a base configuration that includes 400 data points in two classes, ten informative features, zero redundant, zero repeated features, and a noise level of 0.01. The datasets Synth2-13 were then generated by varying different parameters in the base configurations.

Variation	Name	$\|X\|$	$\|F^i\|$	$\|F^d\|$	$\|F^r\|$	η	k
Base	Synth1	400	10	0	0	0.01	2
$\|F\|$	Synth2	400	5	0	0	0.01	2
	Synth3	400	20	0	0	0.01	2
$\|X\|$	Synth4	200	10	0	0	0.01	2
	Synth5	1000	10	0	0	0.01	2
	Synth6	2000	10	0	0	0.01	2
$\|F^i\|, \|F^d\|, \|F^r\|$	Synth7	400	7	3	0	0.01	2
	Synth8	400	7	0	3	0.01	2
	Synth9	400	4	3	3	0.01	2
Noise	Synth10	400	10	0	0	0	2
	Synth11	400	10	0	0	0.05	2
$\|C\|$	Synth12	400	10	0	0	0.01	4
	Synth13	400	10	0	0	0.01	8

Table 3.2: Description of the synthetic datasets used in our experiments.

3.8 Experimental Results

In this section, we present the experimental results to evaluate our SAT-based encoding for learning optimal decision tree classifiers. In Section 3.8.1 we present experimental results for binary and numerical datasets and in Section 3.8.2 we present experimental results for the categorical datasets. Next, in Section 3.8.3 and Section 3.8.4 we present experimental results for cost-sensitive learning and tree pruning constraints, respectively.

3.8.1 Results on Binary and numerical Datasets

In this section, we perform an extensive experimental evaluation of the optimization performance of our SAT-based approach for optimal decision trees on datasets with binary and numerical features. For each of the two optimization criteria, we compare our approach to state-of-the-art combinatorial approaches to learning decision trees that have the same criterion and sets of feasible and optimal solutions. We therefore focus our comparison on the optimization performance, namely the runtime and the objective value (corresponding to the *training* accuracy). We opt against reporting prediction quality on unseen datasets and comparing the two optimization criteria in this aspect.

Previous work on each of the two criteria has already evaluated the quality of decision trees that are optimal with respect to a training dataset on external test datasets [126, 19, 6, 281]. Note that the two criteria are likely to lead to different testing accuracy, and one may be better than the other, depending on the dataset [126, 19]. Min-depth optimal decision trees, and more broadly trees that perfectly fit the training data, require consistent datasets and may lead to overfitting. However, they were found to work well for some datasets [19]. In contrast, max-accuracy optimal trees are likely to be more robust to noise on unseen data, but these decision tree could be seen as less explainable as they do not perfectly account for the patterns in the training data.

Contrarily, in Section 3.8.2, Section 3.8.3, and Section 3.8.4, when we study extensions that augment the set of feasible and optimal solutions (due to different objective or additional constraints), we also analyze the quality of the obtained decision trees based on external test datasets.

Max-Accuracy Objective

We compare our encoding for max-accuracy optimal decision trees to Hu et al.'s MaxSAT encoding [126], Verhaeghe et al.'s CP encoding [280], and Aglin, Nijssen, and Schaus's DL8.5 [6] on the binary and numerical real datasets. Following Hu et al., we set the time limit to 15 minutes and run experiments for three different depth values, $\{2, 3, 4\}$. Table 3.3 shows the time required to find an optimal solution for each approach or timeout ("T/O") if an optimal solution was not found in the time limit. While the time limit of 15 minutes was set for the solvers, we report the *total* runtime (including, for example, model construction). In our approach, as well as Verhaeghe et al., the total time was found to be very close to the

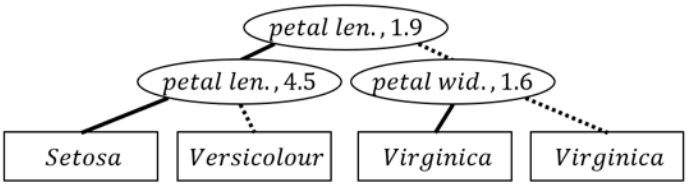


Figure 3.3: A maximum accuracy decision tree classifier of depth 2 for the Iris dataset.

solver running time (the worst observed case was a difference of approximately 40 seconds in Verhaeghe et al., but typically the difference was much shorter). However, in Hu et al. we found that model construction can take significantly longer time periods and in the worst case could take up to approximately 32 minutes for the real datasets and 52 minutes for the synthetic datasets due to the large number of binarized features. We therefore note that for Hu et al., a "T/O" may indicate, in some cases, a much longer total runtime than the 15 minutes allocated to the solver. We also note that Aglin, Nijssen, and Schaus does not use a general-purpose solver and the time limit was set for the custom branch-and-bound tree search.

Table 3.3 also reports the cost of the solutions, i.e., the number of training examples that are *not* correctly classified, for each approach. If an optimal solution was found and proved, then the cost indicates the optimal cost. Otherwise, we report the cost of the best found solution by each approach. If no feasible solution was found in the time limit, we report *UNK*. Note that the datasets Iris, User Knowledge, and Wine could only be solved by our approach and DL8.5, as the other two baselines only support classification problems with two class labels.

In numerical datasets, our approach finds solutions that are at least equal, and in many cases significantly better than the baselines. Furthermore, in all configurations for numerical datasets our approach finds optimal solutions faster than Hu et al. and Verhaeghe et al. When comparing our approach to to DL8.5, we find that DL8.5 tends to find optimal solutions faster for very easy problems (e.g., problems where both approaches find optimal solutions in less than five seconds). However, our approach is able to find optimal solutions much faster for harder instances, e.g., in Banknote, Breast Cancer, User Knowledge and Wine where DL8.5 requires much longer runtimes and in many cases times out. As expected, for binary datasets (Car and Monk2), our approach underperforms relative to the baselines and timed out for depths 3 and 4. Still, it finds the optimal solution for all depths in Car and for depth 2 and 3 in Monk2. In the three datasets with more than two classes, that could not be solved by the MaxSAT and CP baselines, our approach found optimal solutions in seconds and outperformed DL8.5. In Wine, a tree of depth 3 correctly classifies all training examples so we did not run experiments for depth 4. The maximum accuracy decision tree of depth 2 found for Iris is depicted in Figure 3.3. We see that the solution is utilizing the petal length and width features, commonly known as the more meaningful features of Iris for classification [79, 275].

Dataset	d	Solution Cost				Time (s)			
		Ours	SAT [126]	CP [280]	DL [6]	Ours	SAT [126]	CP [280]	DL [6]
Banknote	2	**100**	176	**100**	**100**	**16.83**	T/O	512.21	116.32
	3	**23**	550	100	100	**105.79**	T/O	T/O	T/O
	4	**0**	88	100	100	**18.98**	T/O	T/O	T/O
Breast Cancer	2	**19**	24	**19**	**19**	5.07	T/O	22.19	**2.05**
	3	**9**	25	12	**9**	**242.16**	T/O	T/O	T/O
	4	**0**	18	11	9	**20.79**	T/O	T/O	T/O
Cryotherapy	2	**5**	**5**	**5**	**5**	0.57	3.68	4.21	**0.02**
	3	**1**	**1**	**1**	**1**	0.73	17.57	27.39	**0.69**
	4	**0**	**0**	**0**	**0**	0.75	24.14	7.61	**0.09**
Immunotherapy	2	**8**	**8**	**8**	**8**	0.99	10.53	5.22	**0.06**
	3	**4**	**4**	**4**	**4**	**3.81**	T/O	146.45	6.96
	4	**0**	1	**0**	**0**	1.27	T/O	18.53	**0.67**
Ionosphere	2	**29**	41	**29**	**29**	**155.06**	T/O	T/O	299.19
	3	**21**	186	29	29	T/O	T/O	T/O	T/O
	4	**10**	76	28	28	T/O	T/O	T/O	T/O
Iris	2	**6**	-	-	**6**	0.6	-	-	**0.03**
	3	**1**	-	-	**1**	**0.77**	-	-	1.49
	4	**0**	-	-	**0**	**0.82**	-	-	13.82
User Knowledge	2	**35**	-	-	**35**	1.94	-	-	**0.60**
	3	**10**	-	-	**10**	**3.29**	-	-	168.63
	4	**1**	-	-	8	**3.86**	-	-	T/O
Vertebral Column	2	**45**	46	**45**	**45**	15.79	T/O	67.91	**8.71**
	3	**32**	44	42	**32**	T/O	T/O	T/O	T/O
	4	**15**	39	42	32	T/O	T/O	T/O	T/O
Wine	2	**6**	-	-	**6**	**1.25**	-	-	3.35
	3	**0**	-	-	1	**1.62**	-	-	T/O
Car	2	**250**	**250**	**250**	**250**	12.67	9.2	**2.16**	0.00
	3	**182**	**182**	**182**	**182**	T/O	T/O	5.99	**0.01**
	4	**122**	**122**	**122**	**122**	T/O	T/O	14.09	**0.09**
Monk2	2	**57**	**57**	**57**	**57**	2.74	4.38	**1.38**	0.00
	3	**42**	**42**	**42**	**42**	T/O	826.31	3.6	**0.00**
	4	32	**31**	**31**	**31**	T/O	T/O	8.12	**0.02**

Table 3.3: Results for max-accuracy optimal decision tree problem on real datasets.

Next, we compare our approach for max-accuracy optimal decision trees to Hu et al.'s MaxSAT encoding [126], Verhaeghe et al.'s CP encoding [280], and Aglin, Nijssen, and Schaus's DL8.5 [6] on the synthetic datasets described in Section 3.7.4. Similar to our experiments with real datasets, we set the time limit to 15 minutes and run experiments for three different depth values, $\{2, 3, 4\}$. The results are reported in Table 3.4. We can see that, in general, the synthetic datasets are harder to solve to optimality and optimal solutions were only obtained for decision trees of depth 2. The results show that for the shallowest

trees (depth 2) DL8.5 clearly outperforms our approach in terms of runtime, however our approach still finds the optimal solution in all but two instances. When it comes to deeper trees (depths 3 and 4) all of the approaches are unable to obtain optimal solutions, however our approach finds significantly lower cost solutions compared to the baselines in all cases except for one (Synth5 with depth 3).

Min-Depth Objective

We compare our encoding for min-depth optimal decision trees to Avellaneda's SAT encoding [19] on the binary and numerical datasets. Following Avellaneda's analysis, we set the time limit to 30 minutes. Table 3.5 shows the time to optimal solution for each of the approaches, as well as the tree depth of the optimal solution. As both approaches follow the procedure of solving SAT formulae for increasingly deeper trees until a solution is found, the first solution found is guaranteed to be optimal, i.e., of minimum depth. In case an approach does not find an optimal solution in the time limit, we report "T/O [d]" where d indicates the tree depth of the SAT formula being solved at the moment of timeout. We highlight in bold results for which the alternative approach required at least twice as much run time or timed out.

The results in Table 3.5 show that our approach performs at least as well, and in most cases significantly better on datasets with numerical features. In particular, it finds optimal solutions for Banknote, Breast Cancer and Vertebral Column, for which the baseline timed out. In Ionosphere we find that both methods timed out, however our approach has managed to prove that a solution does not exist for a depth of 3, while the baseline only proved that a solution does not exist for a depth of 2. As expected, in the two binary datasets (Car and Monk2), we find that the baseline outperforms our method. In particular, it manages to find an optimal solution to Car, while our approach timed out.

Next, we compare our approach for min-depth optimal decision trees to Avellaneda's [19] on synthetic datasets. The results are reported in Table 3.6. Again, we find that the synthetic datasets are significantly harder to solve, and both approaches timed out on most instances. Still, our approach shows significant improvement over the baseline as it was able to solve three instances to optimality (compared to zero by the baseline) and in all cases that it was timed out, it manages to prove that a solution does not exist for deeper trees compared to the baseline.

Memory Consumption

To compare the memory consumption of the different approaches, we recorded the peak memory consumption of each approach for each of the datasets. Table 3.7 reports the mean and maximum values for the different approaches for each optimization problem on both real and synthetic datasets. Overall, we observe that our approach has the lowest mean and maximum values in both optimization problems and dataset types.

On real datasets, our maximum memory consumption was approximately 1.3GB. In

Dataset	d	Solution Cost				Time (s)			
		Ours	SAT [126]	CP [280]	DL [6]	Ours	SAT [126]	CP [280]	DL [6]
Synth1	2	**99**	107	**99**	**99**	662.07	T/O	322.36	**67.37**
	3	**69**	134	99	95	T/O	T/O	T/O	T/O
	4	**43**	111	99	94	T/O	T/O	T/O	T/O
Synth2	2	**84**	86	**84**	**84**	33.4	T/O	85.6	**12.81**
	3	**48**	82	82	67	T/O	T/O	T/O	T/O
	4	**21**	53	82	66	T/O	T/O	T/O	T/O
Synth3	2	100	159	100	**95**	T/O	T/O	T/O	**329.48**
	3	**78**	131	95	95	T/O	T/O	T/O	T/O
	4	**48**	151	95	95	T/O	T/O	T/O	T/O
Synth4	2	**44**	55	**44**	**44**	92.69	T/O	79.72	**12.15**
	3	**29**	66	42	35	T/O	T/O	T/O	T/O
	4	**13**	49	42	35	T/O	T/O	T/O	T/O
Synth5	2	**264**	448	282	**264**	T/O	T/O	T/O	**675.82**
	3	292	383	282	**264**	T/O	T/O	T/O	T/O
	4	**201**	417	281	264	T/O	T/O	T/O	T/O
Synth6	2	759	949	**707**	**707**	T/O	T/O	T/O	T/O
	3	**507**	UNK	707	707	T/O	T/O	T/O	T/O
	4	**491**	UNK	706	706	T/O	T/O	T/O	T/O
Synth7	2	**104**	125	**104**	**104**	T/O	T/O	336.59	**67.25**
	3	**86**	125	102	99	T/O	T/O	T/O	T/O
	4	**44**	126	101	98	T/O	T/O	T/O	T/O
Synth8	2	**104**	119	**104**	**104**	534.53	T/O	315.07	**66.86**
	3	**86**	108	104	103	T/O	T/O	T/O	T/O
	4	**48**	125	104	102	T/O	T/O	T/O	T/O
Synth9	2	**71**	77	**71**	**71**	117.3	T/O	300.57	**55.50**
	3	**51**	86	71	70	T/O	T/O	T/O	T/O
	4	**15**	99	71	70	T/O	T/O	T/O	T/O
Synth10	2	**97**	113	**97**	**97**	585.06	T/O	316.8	**65.55**
	3	**82**	135	97	92	T/O	T/O	T/O	T/O
	4	**46**	137	97	92	T/O	T/O	T/O	T/O
Synth11	2	**99**	117	**99**	**99**	571.01	T/O	347.62	**69.78**
	3	**73**	113	99	96	T/O	T/O	T/O	T/O
	4	**43**	140	99	96	T/O	T/O	T/O	T/O
Synth12	2	**198**	-	-	**198**	T/O	-	-	**75.48**
	3	**164**	-	-	193	T/O	-	-	T/O
	4	**132**	-	-	192	T/O	-	-	T/O
Synth13	2	**271**	-	-	**271**	T/O	-	-	**89.02**
	3	**248**	-	-	267	T/O	-	-	T/O
	4	**227**	-	-	267	T/O	-	-	T/O

Table 3.4: Results for max-accuracy optimal decision tree problem in synthetic datasets.

Dataset	Min Depth	Time (s)	
		Ours	SAT [19]
Banknote	4	**5.82**	T/O [4]
Breast Cancer	4	**6.59**	T/O [4]
Cryotherapy	4	**0.08**	0.24
Immunotherapy	4	**0.18**	1.3
Ionosphere	?	T/O [4]	T/O [3]
Iris	4	**0.04**	0.17
User Knowledge	5	**1.31**	59.44
Vertebral Column	5	**87.35**	T/O [5]
Wine	3	**0.11**	14.75
Car	8	T/O [8]	**89.1**
Monk2	6	2.73	**0.28**

Table 3.5: Results for min-depth optimal decision tree problem in real datasets.

Dataset	Min Depth	Time (s)	
		Ours	SAT [19]
Synth1	?	T/O [5]	T/O [4]
Synth2	6	112.56	T/O [4]
Synth3	?	T/O [4]	T/O [3]
Synth4	5	398.09	T/O [4]
Synth5	?	T/O [5]	T/O [3]
Synth6	?	T/O [5]	T/O [3]
Synth7	?	T/O [5]	T/O [4]
Synth8	?	T/O [5]	T/O [4]
Synth9	5	960.22	T/O [4]
Synth10	?	T/O [5]	T/O [3]
Synth11	?	T/O [5]	T/O [4]
Synth12	?	T/O [5]	T/O [4]
Synth13	?	T/O [6]	T/O [3]

Table 3.6: Results for min-depth optimal decision tree problem in synthetic datasets.

comparison, Aglin, Nijssen, and Schaus required approximately 49.7GB in the worst case, making it too big to fit in the memory of many popular workstations or servers. Avellaneda and Hu et al. required more than 10GB in the worst case, almost an order of magnitude higher compared to our worst case. For Verhaeghe et al., we find that the maximum values are approximately within a factor of two from ours, however the mean values are still significantly higher than ours.

On synthetic datasets, all approaches except for ours have significantly higher memory requirements. While in the worst case our approach still requires approximately 1.3GB, Aglin, Nijssen, and Schaus used approximately 50.9GB, Avellaneda used approximately 45.9GB,

and Hu et al. used approximately 37.5GB. While Verhaeghe et al.'s memory requirements were also higher compared to the real datasets, in the worst case it used approximately 5.8GB, within a factor of five from ours.

		Min-Depth		Max-Accuracy			
		Ours	[19]	Ours	[126]	[280]	[6]
Real	Mean	**142.02**	4,527.80	**236.34**	1,646.73	1,176.36	12,627.04
	Max	**1,003.86**	16,684.36	**1,299.09**	10,994.48	2,381.50	49,723.48
Synth	Mean	**425.11**	15,486.00	**273.78**	4,983.97	2,343.20	27,740.05
	Max	**813.52**	45,927.05	**1,307.59**	37,509.36	5,779.00	50,917.67

Table 3.7: Analysis of peak memory consumption (MB) for the different approaches.

3.8.2 Results on Categorical Datasets

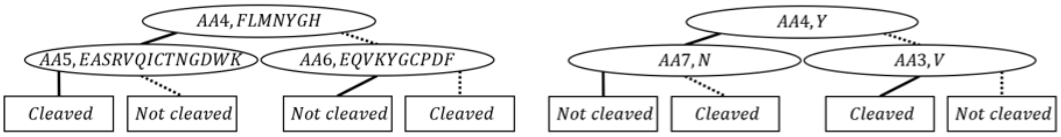
In this section, we present results on datasets with categorical features. We compare our power set branching for categorical features (Ours-PS) against our encoding without power set branching (Ours) and the baselines. As the optimal solution for the different approaches may be different, we do not highlight in bold the lowest run time. For example, our power set branching can fail to find an optimal solution in the time limit, while still obtaining a lower-cost solution compared to an optimal solution that operates on converted binary features.

Max-Accuracy Objective

Table 3.8 and Table 3.9 show the results (solution cost and runtime, respectively) for the max-accuracy optimal decision tree problem. We find that in all of the cases, our power set approach obtains the lowest cost solution and that in almost all cases, these solutions are strictly lower compared to all other approaches. Note that we encountered an error when running the code for Verhaeghe et al. approach on the Credit Approval dataset with a depth of 4. Maximum accuracy classifiers of depth 2 for the Protease Cleavage(/4) dataset with and without power set branching are depicted in Figure 3.4. The Protease Cleavage dataset consists of sequences of amino acids, with each cell taking up to 20 possible categorical values. We see that power set branching enables much more meaningful branchings leading to a significantly lower misclassification cost.

Min-Depth Objective

Table 3.10 reports the results for the min-depth optimal decision tree problem. Note that we report the min depth for each of the approaches as the power set approach can have a lower optimal solution. We find that most categorical datasets could not be solved by any of the methods in the time limit. However, in Protease Cleavage(/4), our encoding that



(a) With power set branching, a misclassification cost of 13 (93% accuracy).

(b) Without power set branching, a misclassification cost of 49 (74% accuracy).

Figure 3.4: Maximum accuracy classifiers of depth 2 for Protease Cleavage(/4) with and without power set branching. AA[N] represents the N-th amino acid and capital letters represent its potential values.

Dataset	d	Cost				
		Ours-PS	Ours	[126]	[280]	[6]
Credit Approval	2	**84**	**84**	**84**	**84**	**84**
	3	**76**	**76**	82	81	77
	4	**60**	65	74	[E]	89
Protease Cleavage	2	**98**	186	186	186	186
	3	**101**	133	149	133	133
	4	**39**	98	136	100	93
Protease Cleavage(/4)	2	**13**	49	49	49	49
	3	**1**	29	29	29	29
	4	**0**	37	40	17	17
Promoter	2	**12**	13	13	13	13
	3	**3**	**3**	4	**3**	**3**
	4	**0**	**0**	**0**	**0**	**0**
Soybean Large	2	**152**	159	-	-	159
	3	**104**	112	-	-	106
	4	**35**	45	-	-	45

Table 3.8: Solution cost for max-accuracy decision trees on datasets with categorical features.

is based on power set branching (Ours-PS) is able to find a solution decision tree of depth 4, while binary branching fails to find a solution after proving that no solution exists for a depth of 6. This demonstrates the expressiveness of our power set branching for categorical features.

Memory Consumption

We also compared the memory consumption of the different approaches on categorical datasets and recorded the peak memory consumption. Table 3.11 reports the mean and maximum values for the different approaches for each optimization problem. On categorical datasets, we observe that our power set encoding has the lowest mean and maximum values, and that our base encoding has the second lowest values. In the worst case, our power set

Dataset	d	Time (s)				
		Ours-PS	Ours	[126]	[280]	[6]
Credit Approval	2	106.26	151.45	T/O	33.06	4.24
	3	T/O	T/O	T/O	T/O	T/O
	4	T/O	T/O	T/O	[E]	T/O
Protease Cleavage	2	T/O	325.35	56.11	4.82	0.04
	3	T/O	T/O	T/O	29.68	3.44
	4	T/O	T/O	T/O	T/O	188.72
Protease Cleavage(/4)	2	18.91	28.58	16.5	4.36	0.04
	3	7.37	575.93	T/O	22.7	2.14
	4	1.27	T/O	T/O	T/O	115.07
Promoter	2	316.71	44.53	316.02	4.35	0.07
	3	T/O	324.77	T/O	63.22	7.20
	4	7.83	57.72	81.08	129.15	24.21
Soybean Large	2	16.22	99.93	-	-	0.02
	3	T/O	T/O	-	-	1.18
	4	T/O	T/O	-	-	32.54

Table 3.9: Runtime for max-accuracy decision trees on datasets with categorical features.

Dataset	Min Depth			Time (s)		
	Ours-PS	Ours	SAT [19]	Ours-PS	Ours	SAT [19]
Credit Approval	?	?	?	T/O [6]	T/O [5]	T/O [5]
Protease Cleavage	?	?	?	T/O [5]	T/O [7]	T/O [6]
Protease Cleavage(/4)	**4**	?	?	0.69	T/O [7]	T/O [6]
Promoter	**4**	**4**	**4**	224.22	498.3	87.69
Soybean Large	?	?	?	T/O [6]	T/O [6]	T/O [6]

Table 3.10: Results for min-depth decision trees on datasets with categorical features.

encoding required approximately 0.7GB of memory and our base encoding approximately 1.9GB. In comparison, the maximum memory consumption for Aghaei, Azizi, and Vayanos is approximately 42.6GB and for Avellaneda approximately 7.6GB. For Hu et al. and Verhaeghe et al. the maximum values are lower, 3.8GB and 2.2GB, respectively, but are still higher compared to our approaches.

	Min-Depth			Max-Accuracy				
	Ours	Ours-PS	[19]	Ours	Ours-PS	[126]	[280]	[6]
Mean	904.73	**382.05**	2,588.05	412.67	**222.61**	728.83	1,332.50	6,228.43
Max	1,865.21	**734.91**	7,580.45	1,451.05	**705.93**	3,838.78	2,180.96	42,610.77

Table 3.11: Analysis of peak memory consumption (MB) on categorical datasets.

Generalization to Unseen Data

In our experiments on numerical datasets (Section 3.8.1), we only compared approaches that have the same sets of feasible and optimal solutions and therefore focused only on the optimization performance (time to find optimal solution or the lowest-cost solution found in case of timeout). In contrast, our new approach for power set branching for categorical features leads to different sets of feasible and optimal solutions. While it does manage to find lower-cost solutions, i.e., solutions that better fit the training data, it could potentially be more prone to overfitting to the training data that would hurt the prediction quality on unseen test data.

In this section, we compare the performance of power set branching encoding vs. our base encoding on unseen test data. To do so, we compute the *testing* accuracy using the same method as in Definition 9 with the set of training data points substituted with testing ones. We use 5-fold cross validation [120] and report the results in Table 3.12. We also report the *total* runtime of the five runs (recall that each run has a time limit of 15 minutes). We restrict our analysis to max-accuracy decision trees as the majority of categorical datasets could not be solved by the min-depth encodings (see Table 3.10). For trees of depth of 2, the power set encoding has at least equal, and in most cases significantly higher, testing accuracy compared to the base encoding. For deeper trees, we observe similar performance for Credit Approval and the two variants of Protease Cleavage. On the Soybean dataset, we observe a significant improvement in performance, while in Promoter, we observe a significant degradation in performance. As we discuss in Section 3.8.4, the degradation in Promoter is likely due to overfitting, and using tree pruning constraints can help mitigate the problem. Finally, we note that for all datasets, the highest testing accuracy (across all configurations) is obtained by the power set encoding (in Credit Approval both encodings obtain the highest testing accuracy).

3.8.3 Cost-Sensitive Learning

In this section, we present experimental results on the cost-sensitive learning approach presented in Section 3.5. To demonstrate the impact of cost-sensitive learning, we choose three datasets that have significant class imbalance and demonstrate how cost-sensitive learning can help mitigate the negative impact of class imbalance:

- **Immunotherapy** is a binary classification dataset with the larger class accounting for approximately 79% of the data points. For this dataset, we assign a higher cost for misclassification of data points in the minority class.
- **Vertebral Column** is a binary classification dataset with the larger class accounting for approximately 68% of the data points. For this dataset, we also assign a higher cost for misclassification of data points in the minority class.
- **Car** is a classification dataset with four classes representing the quality of cars (unacceptable, acceptable, good, very good). The higher-quality classes (good, very good)

Dataset	d	Average Testing Accuracy		Total Time (s)	
		Ours-PS	Ours	Ours-PS	Ours
Credit Approval	2	0.85	0.85	288.00	314.31
	3	0.84	0.83	4506.12	4510.08
	4	0.83	0.84	4510.57	4512.42
Protease Cleavage	2	0.81	0.75	4505.16	1334.82
	3	0.81	0.81	4506.07	4517.60
	4	0.82	0.81	4508.65	4525.03
Protease Cleavage(/4)	2	0.82	0.66	40.68	77.76
	3	0.78	0.81	2.54	1741.67
	4	0.73	0.72	2.52	2968.64
Promoter	2	0.78	0.74	564.22	90.07
	3	0.69	0.77	2014.41	627.15
	4	0.57	0.74	13.52	97.39
Soybean Large	2	0.38	0.32	67.01	184.31
	3	0.60	0.47	4505.94	4507.76
	4	0.79	0.72	4509.17	4510.41

Table 3.12: Results for 5-fold cross-validation on datasets with categorical features.

are significantly smaller and each of them accounts for less than 4% of the points. In all our experiments except for the cost-sensitive experiments in this section, we merge these two classes into the acceptable class following Avellaneda for consistency with previous work. However, in this section we intentionally keep the original class distribution to study the impact of cost-sensitive learning in a highly imbalanced dataset. For this dataset, we assign a higher cost for mistakenly classifying a data point from a higher quality class in a lower quality class.

To analyze the impact of cost-sensitive learning, we analyze the confusion matrix of the obtained classifiers. A confusion matrix M for a classifier is defined as a matrix where each $M_{i,j}$ cell represents the number of data points with true label i that were classified into class j. Table 3.13 and Table 3.14 show the confusion matrix for the training datasets Immunotherapy and Vertebral Column, respectively. We marked the cost of mistake in red using the notation "/e" that represents a cost of e. In each table we present the results for a base configuration where all mistakes have a uniform cost, and a weighted configuration where the cost of misclassification of points from the minority class is doubled. In both datasets, we see that increasing the cost of mistakes in points from the minority class leads to fewer mistakes in minority class and a larger number of mistakes in the majority class. These results demonstrate the ability of cost-sensitive learning to balance the performance across classes.

Table 3.15 shows the confusion matrix for the dataset Car. We can see that in the base configuration (uniform misclassification costs), the smaller classes that represent higher-

True Class	Predicted Class	
	No	Yes
No	15	4/1
Yes	0/1	71
No	17	2/2
Yes	2/1	69

Table 3.13: Confusion Matrix for cost-sensitive learning on Immunotherapy ($d = 3$).

True Class	Predicted Class	
	Negative	Positive
Negative	85	15 /1
Positive	13/1	197
Negative	98	2/2
Positive	34/1	176

Table 3.14: Confusion Matrix for cost-sensitive learning on Vertebral Column ($d = 3$).

quality cars (good and very good) are never predicted. However, as we double the cost of mistakenly classifying a data point from a higher quality class in a lower quality class, we observe that some data points are indeed predicted in the very good class.

True Class	Predicted Class			
	unacc	acc	good	vgood
unacc	1054	156/1	0/1	0/1
acc	26/1	358	0/1	0/1
good	0/1	69/1	0	0/1
vgood	0/1	65/1	0/1	0
unacc	960	242/1	0/1	8/1
acc	0/2	295	0/1	89/1
good	0/2	39/2	0	30/1
vgood	0/2	0/2	0/2	65

Table 3.15: Confusion Matrix for the weighted classification of Car (4 classes).

To evaluate the performance of cost-sensitive learning, we use the Balanced Accuracy metric [115, 204] (also called Average Class Accuracy [155]) which is the average of class-wise accuracy (on balanced datasets, balanced accuracy is equal to accuracy). Given a confusion matrix M, the balanced accuracy is defined as follows.

$$\text{Balanced Accuracy} = \frac{1}{k} \sum_{c \in [1..k]} \frac{M_{c,c}}{\sum M_{c,\cdot}} \tag{3.39}$$

where $\sum M_{c,\cdot}$ represents the total number of data points whose true label is c.

To evaluate the performance of our cost-sensitive learning approach on unseen test data, Table 3.16 shows the balanced testing accuracy, computed using 5-fold cross validation, for the three datasets for a range of cost ratio values (the ratio between the lower-cost classes and the higher-cost classes). We see that we can obtain significantly higher balanced accuracy with cost-sensitive learning by assigning higher cost for mistakes in the smaller classes. Notably, we observe an impressive increase in testing accuracy from 0.44 to 0.64 in Car, a dataset with severe class imbalance.

Cost	Balanced Testing Accuracy			Total Time (s)		
Ratio	Immunotherapy	Vertebral	Car	Immunotherapy	Vertebral	Car
1:1	0.70	0.79	0.44	4.51	4503.55	4510.29
1:2	0.73	0.83	0.64	8.53	4503.33	4513.36
1:3	0.70	0.82	0.64	7.40	4503.27	4514.44
1:4	0.69	0.84	0.59	10.93	4503.20	4514.19
1:5	0.74	0.84	-	8.41	4305.82	-
1:6	0.73	0.83	-	7.03	4128.68	-
1:7	0.69	0.85	-	7.25	3696.10	-

Table 3.16: 5-fold cross-validation results for cost-sensitive learning ($d = 3$).

3.8.4 Tree Pruning Constraints

In this section we present experimental results on the tree pruning constraints in Section 3.6. As we propose these constraints to improve generalization to unseen test datasets, we evaluate the performance using 5-fold cross validation and report the average *testing* accuracy as well as the total run time for the five iterations. Note that in case one or more of the iterations returned infeasible, we report *INF*. In case one or more of the iterations timed out without finding a feasible solution, we report *UNK*.

Minimum Support

Table 3.17 reports the 5-fold cross validation testing accuracy on the numerical and binary datasets for different depth values and different minimum support values ($S = 0$ indicates no minimum support). Table 3.18 reports the results for a similar analysis on categorical datasets based on our power set encoding. We can see that for the shallowest trees (depth 2) adding minimum support constraints does not lead to improvement in most cases, and can even lead to lower accuracy. In contrast, for tree of depth 4, we can see that the testing accuracy for minimum support constraints tends to be equal or higher. For example, using a minimum support $S = 2$ with trees of depth 4 has led to higher testing accuracy in 9 datasets compared to 4 datasets where it resulted in lower accuracy (across all datasets). As we noted in Section 3.8.2, the dataset Promoter with $S = 0$ exhibits degraded testing

accuracy for deeper trees, indicating overfitting to the training data. However, we observe that problem is significantly reduced when introducing constraints for minimum support of $S = 2$.

Table 3.17 and Table 3.18 also report the total runtime (over all five iterations) for each of the configurations. The results show that minimum support constraints with $S = 2$ have comparable run times. As the value of S increases, we observe an increase in run time. The longer runtime may be due to the smaller set of feasible solutions satisfying minimum support constraints with higher S (in some cases there are no feasible solutions at all). It may be due to the larger number of variables and clauses required for the encoding of cardinality constraints with higher S.

Minimum Margin

Table 3.19 report the 5-fold cross validation testing accuracy on the numerical and binary datasets for different depth values and different minimum margin values (recall that minimum margin is not implemented for categorical branching). As with minimum support constraints, it is shown that minimum margin constraints often lead to higher accuracy, but can also lead to lower accuracy. For trees of depth 4, using a minimum margin of $M = 4$ has led to higher testing accuracy in 7 datasets compared to 2 datasets where it resulted in lower testing accuracy. While higher values of minimum margin often lead to faster runtimes, the differences tend to be small.

Discussion

We have shown that the two types of tree pruning constraints presented in Section 3.6 can help improve the accuracy on unseen data. Despite the theoretical connection between the two types (Proposition 3), there are some differences between the approaches. As noted in Section 3.6.2, we consider minimum margin a weaker notion than minimum support as it does not enforce a minimum number of data points to actually go through the node. However, minimum margin constraints require no additional variables or clauses as we simply change the clauses in Eq. (3.14) and Eq. (3.15) to the clauses in Eq. (3.37) and Eq. (3.38). Minimum support represents a stronger notion as it enforces that a given number of data points arrive at a given leaf node. However, each minimum support constraint requires $O(|X| \times S)$ auxiliary variables and $O(|X| \times S)$ additional clauses to be implemented. Note that these values are multiplied by the number of leaves since we have that many minimum support constraints. These differences could potentially account for the longer runtimes observed for minimum support constraints with high values for threshold S.

3.9 Related Work

Related to this work are different approaches for finding optimal decision-tree classifiers. Nijssen and Fromont [215] proposed DL8, one of the earliest approaches for building opti-

Dataset	d	Average Testing Accuracy				Total Time (s)			
		S=0	S=2	S=5	S=10	S=0	S=2	S=5	S=10
Banknote	2	0.91	0.91	0.91	0.91	71.58	68.23	87.46	118.32
	3	0.97	0.97	0.98	0.97	267.32	305.24	467.74	406.72
	4	0.98	0.98	0.98	0.99	44.61	167.80	354.03	702.23
Breast Cancer	2	0.74	0.75	0.75	0.73	14.53	19.36	15.91	24.17
	3	0.77	0.71	0.71	0.73	372.49	319.32	824.56	3039.64
	4	0.62	0.56	0.65	INF	18.99	53.97	4511.27	203.83
Cryotherapy	2	0.94	0.92	0.79	0.79	1.05	2.33	3.26	4.41
	3	0.90	0.89	0.84	INF	1.79	4.83	8.90	1.68
	4	0.86	0.89	INF	INF	1.60	6.78	5.37	2.50
Immunotherapy	2	0.80	0.84	0.81	0.86	1.83	2.95	3.81	4.50
	3	0.78	0.78	0.82	INF	5.21	12.28	39.65	2.22
	4	0.77	0.81	INF	INF	1.92	10.71	15.81	4.66
Ionosphere	2	0.91	0.90	0.90	0.90	471.31	363.29	453.62	461.56
	3	0.87	0.85	0.87	0.87	4507.61	4511.08	4513.78	4519.00
	4	0.83	0.83	0.83	0.81	2902.37	2358.73	2073.45	4543.40
Iris	2	0.93	0.92	0.91	0.95	1.13	2.88	3.83	5.24
	3	0.93	0.95	0.92	0.95	1.45	4.94	7.65	12.88
	4	0.93	0.95	0.92	INF	1.76	9.39	26.93	4.07
User Knowledge	2	0.86	0.84	0.85	0.84	4.52	6.63	8.89	10.30
	3	0.90	0.92	0.91	0.93	10.99	16.00	24.56	38.57
	4	0.90	0.91	0.91	0.91	9.16	35.82	292.84	4527.73
Vertebral Column	2	0.76	0.78	0.77	0.76	42.63	48.41	50.41	55.94
	3	0.80	0.78	0.85	0.83	4504.69	4508.95	4511.17	4517.05
	4	0.79	0.78	0.80	0.81	4505.64	4511.64	4516.80	4533.51
Wine	2	0.93	0.93	0.94	0.93	2.29	3.85	5.35	7.13
	3	0.92	0.93	0.93	0.90	2.93	6.94	9.84	39.93
Car	2	0.86	0.86	0.86	0.86	46.00	53.75	79.91	107.76
	3	0.87	0.87	0.87	0.87	4509.90	4523.90	4555.57	4640.73
	4	0.91	0.93	0.92	0.92	4514.49	4562.40	4677.54	5001.34
Monk2	2	0.60	0.57	0.57	0.60	8.53	9.58	10.04	12.57
	3	0.68	0.66	0.66	0.64	2815.43	2333.87	2280.73	417.40
	4	0.53	0.63	0.62	INF	4505.03	4510.99	4277.60	3.75

Table 3.17: Results for 5-fold cross-validation with minimum support constraints.

mal decision trees based on itemset mining. Later, Bessiere, Hebrard, and O'Sullivan [39] presented the first approach for computing optimal decision trees using discrete optimization. They proposed a SAT encoding and a constraint programming (CP) encoding for computing minimal-size decision trees that perfectly classify all the training examples.

Dataset	d	Average Testing Accuracy				Total Time (s)			
		S=0	S=2	S=5	S=10	S=0	S=2	S=5	S=10
Credit Approval	2	0.85	0.85	0.83	0.82	252.35	283.77	269.46	335.64
	3	0.84	0.84	0.84	0.84	4506.59	4513.07	4519.98	4538.83
	4	0.83	0.81	0.82	0.83	4509.98	4522.63	4549.80	4598.51
Protease Cleavage	2	0.81	0.84	0.87	0.84	4505.30	4509.31	4509.31	4515.39
	3	0.81	0.83	0.81	0.83	4504.65	4510.53	4516.34	4536.92
	4	0.82	0.83	0.82	0.83	4506.41	4517.20	4543.89	4602.82
Protease Cleavage (/4)	2	0.82	0.83	0.81	0.82	40.91	42.61	43.20	49.14
	3	0.78	0.77	0.76	0.76	2.41	5.12	8.01	18.98
	4	0.73	0.69	0.74	UNK	2.22	8.60	10.52	902.86
Promoter	2	0.78	0.75	0.71	0.76	550.07	539.41	581.26	527.34
	3	0.69	0.73	0.72	0.69	2008.56	1843.48	1922.11	4508.18
	4	0.57	0.67	0.63	UNK	12.81	18.23	2502.12	902.93
Soybean Large	2	0.38	0.38	0.37	0.37	68.60	70.65	69.89	79.73
	3	0.60	0.56	0.59	0.57	4505.87	4510.83	4512.36	4515.91
	4	0.79	0.79	0.73	0.53	4507.17	4514.74	4520.88	4534.60

Table 3.18: Results for 5-fold cross-validation with minimum support constraints on datasets with categorical features.

Bertsimas and Dunn [37] proposed OCT, a Mixed Integer Programming (MIP) approach for computing decision trees that minimizes the misclassifications of training examples for a given tree depth. The resulting trees were found to be more accurate on unseen test data than those created by heuristic methods. Verwer and Zhang [281] proposed BinOCT, a MIP approach that is based on binary decisions, and was able to compute solutions faster than OCT. Recently, Aghaei, Azizi, and Vayanos [4] presented a new MIP formulation of decision trees that offers a trade-off between accuracy and fairness.

Narodytska et al. [209] proposed a SAT encoding for optimal decision trees that perfectly classify all training examples and that improved over Bessiere, Hebrard, and O'Sullivan's [39]. More recent works on optimal trees that perfectly fit the training data include Janota and Morgado's [147] alternative SAT encoding that is based on explicit paths in the tree and Avellaneda's [19] SAT encoding that fixes the tree structure and encodes the behavior of each data point. Hu et al. [126] proposed a MaxSAT extension of Narodytska et al.'s SAT encoding that minimizes the misclassification of training examples for a given tree depth.

Verhaeghe et al. [280] recently presented a CP approach for optimal decision trees that minimizes the misclassification of training examples for a given tree depth. Their approach combined global constraints for itemset mining, an AND/OR tree search, and the use of caching. Their approach was able to compute optimal decision trees faster than BinOCT [281] and DL8 [215]. Aglin, Nijssen, and Schaus [6] proposed DL8.5 that improved over the CP approach by using a custom branch-and-bound caching tree search.

Dataset	d	Average Testing Accuracy				Total Time (s)			
		M=1	M=4	M=10	M=20	M=1	M=4	M=10	M=20
Banknote	2	0.91	0.91	0.91	0.91	71.80	62.90	64.09	62.68
	3	0.97	0.97	0.97	0.97	274.42	264.88	191.61	209.57
	4	0.98	0.98	0.98	0.98	45.15	44.69	40.65	44.04
Breast Cancer	2	0.74	0.75	0.75	0.72	15.27	14.58	8.80	7.91
	3	0.77	0.73	0.73	0.70	372.61	317.35	276.37	319.79
	4	0.62	0.66	0.65	0.60	19.34	16.87	10.40	7.34
Cryotherapy	2	0.94	0.94	0.82	0.86	1.07	1.03	1.12	1.12
	3	0.90	0.89	0.91	0.86	1.55	1.63	1.58	1.61
	4	0.86	0.78	0.84	0.86	1.70	1.80	1.84	1.76
Immunotherapy	2	0.80	0.82	0.83	0.78	1.97	1.77	1.68	1.78
	3	0.78	0.77	0.82	0.78	5.28	6.71	6.01	8.52
	4	0.77	0.86	0.82	0.76	1.88	2.15	2.13	2.51
Ionosphere	2	0.91	0.90	0.89	0.90	472.04	451.11	501.34	422.28
	3	0.87	0.86	0.87	0.84	4507.32	4507.30	4507.12	4506.99
	4	0.83	0.85	0.85	0.85	2877.72	2180.39	933.64	1031.55
Iris	2	0.93	0.92	0.90	0.93	1.09	1.06	1.03	1.16
	3	0.93	0.95	0.94	0.95	1.46	1.48	1.38	1.40
	4	0.93	0.93	0.93	0.93	1.93	1.95	1.83	1.88
User Knowledge	2	0.86	0.85	0.85	0.85	4.91	4.97	4.69	3.54
	3	0.90	0.91	0.92	0.93	10.63	10.40	8.78	9.58
	4	0.90	0.91	0.92	0.92	9.19	10.83	7.84	11.57
Vertebral Column	2	0.76	0.76	0.77	0.77	41.06	36.58	36.81	31.10
	3	0.80	0.85	0.81	0.83	4504.80	4504.41	4504.23	4504.48
	4	0.79	0.78	0.82	0.80	4505.03	4505.69	4505.67	4505.48
Wine	2	0.93	0.91	0.94	0.92	2.27	2.00	2.21	2.28
	3	0.92	0.89	0.89	0.91	2.95	2.90	2.52	2.66
Car	2	0.86	0.86	0.86	0.86	46.38	40.94	43.23	39.13
	3	0.87	0.87	0.87	0.87	4509.56	4509.33	4509.28	4509.37
	4	0.91	0.92	0.92	0.92	4513.45	4512.34	4515.24	4511.78
Monk2	2	0.60	0.60	0.60	0.61	8.35	8.52	7.73	7.58
	3	0.68	0.68	0.67	0.67	2814.88	3017.89	3250.32	2796.44
	4	0.53	0.61	0.59	0.64	4504.52	4504.76	4504.49	4504.62

Table 3.19: Results for 5-fold cross-validation with minimum margin constraints.

Günlük et al. [111] recently presented a MIP formulation of optimal decision trees for categorical data. Unlike previous approaches that converted categorical features to a set of binary features, the new MIP formulation directly handles categorical features. However, this approach is restricted to categorical features and cannot directly handle numerical

features.[10]

Previous works have also considered other optimal classification models, including decision lists [238, 15, 303, 302], decision sets [136, 301, 302, 137], and binary decision diagrams [53]. We refer the reader to Ignatiev et al. [139] for an extended survey.

3.10 Conclusion and Future Work

In this chapter, we present our approach that falls in the category of model-based interpretable ML using combinatorial optimization problems. Namely, we introduce and study a novel SAT-based encoding for optimal decision tree classifiers. Using the declarative expressiveness afforded by the formulation approach, our approach supports direct encoding of non-binary (numerical and categorical) features and is used to optimize two popular optimization criteria. We further extend our model with cost-sensitive learning and tree pruning constraints that can help mitigate the negative impact of imbalanced datasets and overfitting to training examples.

The experimental results show that the direct handling of numerical and categorical features leads to better performance compared to state-of-the-art approaches that are designed for binary datasets.[11] The improvement is seen on two different optimization criteria and shows that our approach is preferable regardless of the choice of which objective is more suitable to a given dataset, a decision that can be tuned as a hyper-parameter on unseen data using cross validation. The results support our hypothesis that converting numeric features to a large number of binary features can have significant negative impact on performance, however a more thorough solver-based analysis may lead to deeper insight into the impact of the binarization on performance. Enhanced optimization of the min-depth objective also contributes to interpretability as it yields more sparse and simulatable solutions. Furthermore, our experimental results show that our categorical encoding can lead to decision trees that better fit the training data and often have better performance on unseen test data. The more expressive branching due to the direct encoding of categorical features can also contribute to sparsity and simulatability. Surprisingly, it also enhances modularity as one powerset branching can represent multiple one-hot branchings, leading to meaningful individual nodes that can be understood independent of the surrounding context. Next, we experimentally show the benefits of cost-sensitive learning towards maximizing balanced accuracy in classification tasks that are unbalanced in data or cost. Cost-sensitive learning, as a way to customize the objective function based on an expert's input, allows a greater level of control on the solution. Lastly, our experimental results show that tree pruning constraints are successful in improving generalization to unseen data, proving the importance of constrained ML which combinatorial optimization problems enable.

[10] The paper suggests converting numerical features to categorical features with limited number of categories via thresholding [111], however this conversion does not preserve the optimality of solutions w.r.t. the original numerical values.

[11] Note that all approaches explore the same space of (feasible and) optimal decision-tree solutions.

We believe that the work in this chapter can be extended in a number of ways. Investigating ways to extend our formulation to support different notions of fairness in classification is an interesting direction for future work. Such extensions can range from forbidding or analyzing the appearance of sensitive features in the decision chain of a tree to using cardinality constraints to enforce an equitable treatment of different sensitive groups in training data. Analyzing the impact of different tree structures and the use of our approach as part of an ensemble method are also interesting research directions. We only utilized our encoding to find fully-balanced decision trees. However, there is no theoretical barrier to considering unbalanced trees with a heuristic-based structure. Moreover, our current approach for the min-depth problem consists of iteratively solving our encoding for increasing depths starting from one, to entail the optimality of the first solution. However, alternative approaches such as using a binary search over tree depths can potentially allow us to obtain feasible solutions early while refining our bounds toward an optimal solution. Such directions to extend our work can be helpful in addressing the cases where a solution is not found in the time limit. Finally, we are interested in investigating the applicability of a similar approach based on combinatorial optimization problems for dealing with non-binary features in other classification models such as decision sets [301, 302] and decision lists [303, 302]. Both formats are compact and interpretable ML solutions that can also benefit from the declarative nature of the formulation (into SAT) approach.

Chapter 4

Compact Binary Decision Diagrams for Classification

4.1 Introduction

In this chapter, we focus on binary decision diagrams (BDD) as inherently interpretable models to solve the problem of classification. BDDs, like decision trees, are interpretable yet accurate classifiers, but they are more compact at the cost of being less expressive. Similar to our approach for learning decision tree classifiers (Chapter 3), we learn BDDs directly by formulating the problem into the declarative combinatorial optimization problem of SAT.

Classifiers are complete functions for assigning labels to data points, and are learned from a limited set of supervised training data. A classifier is trained to focus on informative aspects of the input and extract patterns from the data to make decisions. As a result, the size of a classifier has a crucial impact on its accuracy and also its interpretability. In complex black-box classifiers such as deep and convolutional neural networks, it is often challenging to understand the influential features of the data and the inner-workings of the decision-making [268, 308]. In contrast, interpretable classifiers such as decision trees [6, 178] are sparse, simulatable, and modular [206, 56], with surprisingly small cost to accuracy in many applications [237, 177]. Decision diagrams are interpretable models that are even more concise than decision trees [124, 160]. The properties of interpretable solutions make them the perfect candidates for when we need to formally analyze or explain the classifier's behavior [193, 134, 136].

Binary decision diagrams (BDD) are graph representations of functions with binary inputs and are widely used in logical synthesis and formal verification methods [52, 9, 202, 159]. While equivalent to truth tables in purpose, BDDs are more compact since redundant sub-tables can be eliminated and merged [159]. A multi-terminal BDD (MTBDD) is a BDD variant that supports multiple outputs [59].

Decision trees are the most common form of interpretable classifiers (e.g., [50, 230, 229, 19, 280, 126, 37]). However, the number of branchings double at each level of a decision tree, making it challenging to interpret the solution due to the lack of sparsity as depth increases [111]. The large number of branchings in decision trees can hinder the learning performance given that the search space grows exponentially with each level. Further, deep branchings that only affect a small number of data points can cause data fragmentation and overfitting [254, 140, 84]. Lastly, the exponential number of branchings in decision trees restricts their applicability to memory-constrained hardware [254].

We have experimentally observed the issue of exponential growth in decision trees in Chapter 3, Section 3.8.1. Specifically, it has been shown that it is challenging to find perfect classifiers since solving the encoding for deep solutions takes a significant amount of time. The effect is most noticeable for synthetic instances with a large number of informative features, highlighting the relation between sparse classifiers and recognition of informative patterns in data.

In this chapter, we choose BDDs as our interpretable classifiers in order to emphasize sparsity to an even greater degree compared to interpretable approaches based on decision trees (e.g., Chapter 3). The sequential nature of decision-making in BDDs resembles that of decision trees, but unlike decision trees, the same branching is used across a level. This leads to a number of branchings that is linear in depth, and allows BDDs to address the issues of exponential branchings, with a potential cost to expressiveness and accuracy. A BDD is not necessarily a tree as using the same feature across a level allows the merging of branching nodes that are equivalent. The transition from trees to diagrams enhances sparsity but does not hurt the simulatability and modularity of the model as one can still follow the path of activated branchings from the root to a leaf and understand the decisions made at each step independently.

Traditionally, efforts to learn BDDs are focused on optimizing compactness [160]. Learning BDDs is shown to be an NP-hard problem in multiple contexts, including finding a variable ordering that results in the smallest BDD [265]. For the problem of classification, combinatorial optimization problems have been proposed to learn BDDs with the primary objective of maximum accuracy [124]. Nevertheless, the possibility of size optimization as a secondary consideration is maintained. As discussed in Chapter 3, the computation cost of optimality is not always justified, but optimal solutions have been shown to enhance generalization and are of significance when interpretability is demanded. This observation is extended to when BDDs are learned in an exact way [124] rather than heuristically [160, 161]. Furthermore, combinatorial approaches are also shown to be significant when optimizing the size of BBDs rather than their accuracy [54], directly contributing to solution interpretability through sparsity.

Our SAT-based encoding for learning BDDs reuses our direct encoding of branchings over numerical features (Chapter 3) instead of explicitly binarizing them. A straight-forward repurposing of this component and its combination with BDD-specific aspects is a testament to the flexibility provided by the formulation approach. We introduce size optimization that

can be added as a secondary component to the objective or solved separately. We also expand the notion of learning branchings to multiple dimensions in order to learn solutions from a limited but distinctively interpretable family of diagrams.

The main contributions of this chapter are as follows:

1. We present a novel MaxSAT encoding of maximum accuracy BDD classifiers for numerical datasets. Our model represents a non-reduced BDD which can be reduced according to its sequence of terminals.

2. We extend our encoding by modelling the size of the final reduced BDD. The size encoding can be used as a secondary weighted objective, or it can be optimized in a second stage.

3. We present a variant of our encoding which supports direct learning of diagrams through multi-dimensional branchings. Multi-dimensional BDDs enable learning more efficiently as they provide more expressive solutions within the same size.

4. In our experiments, we demonstrate that our base encoding outperforms the state of the art in runtime and accuracy. Additionally, we show that explicitly optimizing size leads to significantly more compact solutions while maintaining high accuracy. Finally, we show that multi-dimensional BDDs scale better than BDDs due to their expressiveness in compact size.

The content of this chapter mirrors that of the work that first appeared in the proceedings of the Twenty-ninth International Conference on Principles and Practice of Constraint Programming [252]. We have revised the notation and terminology to maintain consistency with the rest of the dissertation. Furthermore, certain theoretical contributions and points of discussion are elaborated upon and extended.

4.2 Preliminaries

In this section, we formally define binary decision diagrams (BDDs) as classifiers. Following an approach similar to that used for decision trees (Section 3.2), we first define the underlying structure of BDDs and then proceed to describe how they operate. Next, we define the notions of reduction and equivalency for BDDs in Section 4.2.1. Lastly, we define the max-accuracy classification problem with BDDs as the solutions in Section 4.2.2.

Definition 12 (Binary Diagram Structure)**.** *A binary diagram structure $\mathcal{N}$ is a rooted, directed, and acyclic graph. Nodes of $\mathcal{N}$ are divided into terminals $\mathcal{N}_T$ and non-terminal nodes that are referred to as branching nodes $\mathcal{N}_B$ and contain the root node $\delta \in \mathcal{N}_B$. Each branching node has two outgoing edges leading to its left and right children, corresponding to values 0 and 1, respectively. The left and right children are represented by functions $l, r : \mathcal{N}_B \to (\mathcal{N}_B \cup \mathcal{N}_T)$. The parent function $p : (\mathcal{N}_B \cup \mathcal{N}_T - \{\delta\}) \to \mathcal{N}_B$ represents the edges in reverse $\forall t_p, t_c : p(t_c) = t_p \Leftrightarrow (l(t_p) = t_c \vee r(t_p) = t_c)$.*

Definition 13 (Binary Decision Diagram)**.** *Given a set of features F and integer labels $[1..k]$, a binary decision diagram is a tuple $\mathcal{D} = (\mathcal{N}, \beta, \alpha, \theta)$ where $\mathcal{N}$ is a binary diagram structure, $\beta : \mathcal{N}_B \to F$ is the feature selection function, $\alpha : \mathcal{N}_B \to \mathbb{R}$ is the threshold selection function, and $\theta : \mathcal{N}_T \to [1..k]$ is the terminal labelling function. The* size *of a BDD is the number of branching nodes in its structure.*

Note that in conventional terminology, BDDs are defined as representations of boolean functions with boolean inputs. We adopt standard BDD terminology, with minor exceptions to facilitate discussion of BDDs in a classification context. One difference in our definition is the support for the terminals to have more than two labels (namely, 0 and 1), a variation that is referred to as a multi-terminal BDD in the literature.

We also borrow from the decision tree literature to view the outgoing edges as parent-child relationships. Moreover, we allow the inner nodes in a BDD to perform branchings (Definition 1), which are more general than the boolean features standard BDDs operate on.

We define how BDDs assign labels to data points, akin to how decision trees do so. Namely, the data point is entered from the root and directed based on its values in branchings until a labeling terminal is met.

Definition 14 (BDD Label Assignment)**.** *Given a set of features F, integer labels $[1..k]$, an evaluation of F as a data point $x \in \mathbb{R}^{|F|}$, and a BDD $\mathcal{D} = (\mathcal{N}, \beta, \alpha, \theta)$, the value of the function $\Theta_{\mathcal{D}}(\delta, x)$ represents the label that $\mathcal{D}$ assigns to x. The function $\Theta_{\mathcal{D}} \in (\mathcal{N}_B \cup \mathcal{N}_T) \times \mathbb{R}^{|F|} \to [1..k]$ is recursively defined as follows:*

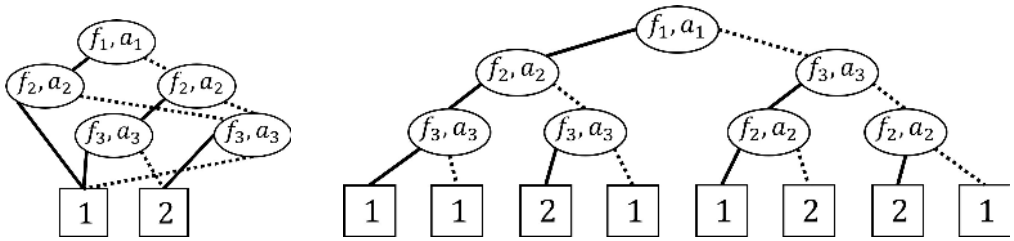
$$\Theta(t, x) = \begin{cases} \theta(t) & \textit{if } t \in \mathcal{N}_T \\ \Theta(l(t), x) & \textit{elseif } x[\beta(t)] \leq \alpha(t) \\ \Theta(r(t), x) & \textit{else} \end{cases}$$

The data point is directed *along the outgoing edge corresponding to its branching value at each step. Once a data point reaches a terminal, it is assigned the label of the terminal. Every node t invoked with a point x is said to* contain *that point,*

Similar to decision trees, we use the shorter notation $\Theta_{\mathcal{D}}(x)$ to denote $\Theta_{\mathcal{D}}(\delta, x)$ for BDDs. Additionally, $\Theta_{\mathcal{D}}(x)$ is interpreted as assigning classification labels when the BDD is intended as a classifier.

4.2.1 Reduced BDDs

The same function, in our case a classifier, can be represented by multiple BDDs. To uniquely represent a function, we define the BDD properties of being *ordered* and *reduced*. The ordered property enforces the same sequence of branchings along each path, effectively making BDDs equivalent to Oblivious Read-Once Decision Graphs (OODGs). The reduced property requires all equivalent parts of the BDD to be merged and all of the redundant



(a) Ordered and reduced. (b) Non-ordered and non-reduced.

Figure 4.1: Equivalent diagram and tree BDDs. Branching nodes are labeled with $(\beta(t), \alpha(t))$ values. Solid (dotted) lines correspond to the value 1 (0).

parts to be removed. Given a function and an ordering of splits, there exists only one ordered and reduced BDD representing the function.

Definition 15 (Ordered BDD and Split Sequence)**.** *A BDD is said to be ordered if the branchings observed along every path from the root to a terminal respect a singular total ordering, called its branching sequence. All branching nodes corresponding to the s-th branching in the sequence are said to be at level s.*

Definition 16 (Node Equivalency and Redundancy)**.** *Consider an ordered binary decision diagram $\mathcal{D}$. Two of its terminals $t_1, t_2 \in \mathcal{N}_T$ are equivalent if $\theta(t_1) = \theta(t_2)$. Moreover, two of its branching nodes $t_1, t_2 \in \mathcal{N}_B$ are equivalent if $\beta(t_1) = \beta(t_2)$, $\alpha(t_1) = \alpha(t_2)$, $l(t_1)$ and $l(t_2)$ are equivalent, and $r(t_1)$ and $r(t_2)$ are equivalent. Furthermore, a branching node $t \in \mathcal{N}_B$ is redundant if its left and right children are equivalent.*

Definition 17 (Reduced BDD)**.** *A BDD is said to be reduced if all of its equivalent nodes are merged and all of its redundant nodes are replaced with their children.*

We assume a BDD to be ordered and reduced unless noted otherwise. Figure 4.1 depicts two BDDs representing the same function, one ordered and reduced and the other one not. It has been shown in the literature that independent of the merging and elimination process, the final result of reducing a BDD is always the same. We formally state the uniqueness of reduced BDDs in Theorem 2 and refer the reader to the BDD literature for more details [159].[1]

Theorem 2. *Given a function and a corresponding set of branchings in total order, there is exactly one ordered and reduced BDD presentation up to renaming.*

4.2.2 BDD Classifiers

In this section, we define the problem of learning BDD classifiers. The accuracy of a BDD classifier is defined similar to that of a decision tree classifier (Section 9). We introduce the

[1] Note that when employing standard BDD terminology, redundant and equivalent nodes are defined through binary sequences called *beads*. Beads correspond to nodes of the uniquely reduced BDD described in Theorem 2.

problem of learning a BDD classifier with maximum accuracy, aligning with recent research on classification via BDDs [124].

Problem 4 (Max-Accuracy Binary Decision Diagram)**.** *Given a set of features F, integer labels $[1..k]$, a set of training data points $X \subset \mathbb{R}^{|F|}$, a labeling $\gamma : X \to [1..k]$, and a maximum number of branchings d, output an ordered binary decision diagram $\mathcal{D}$ with a branching sequence of length d and maximum accuracy (Definition 9).*

Note that the size (number of branching nodes) in a BDD solution can also be optimized as a secondary objective. When optimizing the size, we need to consider the reduced version as it can be significantly smaller than the non-reduced equivalent BDD.

4.3 SAT-based Encoding of BDD Classifiers

In this section, we solve Problem 4 by proposing a MaxSAT encoding to find a BDD classifier with maximum accuracy. Similar to previous work of Hu, Huguet, and Siala [124], we learn an ordered yet non-reduced (tree) BDD first. We then focus on merging and replacing nodes towards a reduced BDD. Inspired by the numerical branching in Chapter 3, we encode branchings (Definition 1) without the need for prior binarization of data which was shown to lead to significant performance degradation in decision trees. We instead directly encode how each branching directs each point, while making sure the order of values is respected given a selected feature.

In Section 4.3.1, we present the variables that are used to encode a BDD. In Section 4.3.2, we present the hard clauses that guarantee the validity of our BDD encoding alongside the soft clauses that model our maximum accuracy objective. Finally in Section 4.3.3, we show how a satisfying boolean assignment to our SAT encoding can be decoded into an unreduced BDD first, and then to a reduced equivalent.

4.3.1 Variables and Structure

Given that we are learning an ordered but non-reduced BDD, it suffices to find a branching sequence of length d and 2^d terminal labels. Specifically, we need to encode the value of β and α functions for a sequence of branchings and the value of θ for each terminal. We use $\beta(s)$ and $\alpha(s)$, respectively, as shorthands for the feature and threshold selected in level s of the branching sequence, i.e., for all branching nodes at level s.

We use a unary encoding to represent feature selection at each branching. Namely, a variable sequence of length $|F|-1$ is used for each branching s, with the number of variables set to true representing the index of the chosen feature. In order for the unary encoding to be well-formed, the values assigned to the sequence of variables should not include the sequence 01 at any point.

The following set of binary variables represents different aspects of modeling the branchings and the terminal, namely the values of β, α for a sequence of length d and θ for 2^d terminals:

- $\{a_{s,j} \mid 0 \leq s < d, j \in F\}$: Represents whether the feature chosen at branching s is or comes before j. It is equivalent to $\beta(s) \leq j$.
- $\{d_{s,i} \mid 0 \leq s < d, x_i \in X\}$: Represents whether point x_i is directed to the left child at branching s. It is equivalent to $x_i[\beta(s)] \leq \alpha(s)$.
- $\{c_{t,l} \mid 1 \leq l \leq k, t \in \mathcal{N}_T\}$: Represents whether output label l is assigned to terminal node t. It is equivalent to $\theta(t) = l$.
- $\{o_i \mid x_i \in X\}$: Represents whether point x_i ends up in a terminal with the same label, i.e., is correctly classified. It is equivalent to $\Theta(x_i) = \gamma(x_i)$.

Note that the number of $a_{s,j}$ variables is in $O(d|F|)$, $d_{s,i}$ in $O(d|X|)$, $c_{t,l}$ in $O(k|\mathcal{N}_T|)$, o_i in $O(|X|)$, resulting in the total number of variables to be in $O(d|X|)$ if $d|X| \geq k|\mathcal{N}_T|$ and $|X| \geq |F|$.

4.3.2 Clauses

We propose the following sets of Conjunctive Normal Form (CNF) clauses for modelling a non-reduced ordered BDD. The clauses in Eq. (4.1) and Eq. (4.2) guarantee that one feature is selected at each branching by enforcing the ordered encoding of $a_{s,j}$ variables. The clauses in Eq. (4.3) and Eq. (4.4) guarantee that for each branching and chosen feature, the points directed to the left (respectively, right) have a lesser or equal (respectively, greater) value compared to a threshold. The clauses in Eq. (4.5) guarantee that each branching directs at least one point to the left. Note that unlike the encoding in Section 3.3, we are not also adding clauses to guarantee at least one point to be directed to the right. We opt against doing so, as forcing a non-trivial branching can interfere with the reduction of a BDD and its potential size optimization. The clauses in Eq. (4.6) and Eq. (4.7) guarantee that one output label is chosen for each terminal node. The clauses in Eq. (4.8) guarantee that a point can only be considered classified if it ends up in a terminal node with the same label as its ground truth. We use $O_j(X)$ to denote the set of all consecutive pairs when the members of X are sorted based on their j values, $O_j^=(X)$ to denote the subset of consecutive pairs with equal j values, i.e., $O_j(X) \cap \{(i_1, i_2) \,|\, x_{i_1}[j] = x_{i_2}[j]\}$, and $\#_j^1$ to denote the index of the point with the smallest j value. Furthermore, we assume $a_{s,j+1}$ for $j = max(F)$ to be the false constant and define $A_R(t)$ ($A_L(t)$) as the set of right (left) ancestors of terminal t.

$$(a_{s,j}, \neg a_{s,j+1}) \qquad s < d, j \in F \tag{4.1}$$

$$(a_{s,0}) \qquad s < d \tag{4.2}$$

$$(\neg a_{s,j}, a_{s,j+1}, d_{s,i_1}, \neg d_{s,i_2}) \qquad s < d, j \in F, (i_1, i_2) \in O_j(X) \tag{4.3}$$

$$(\neg a_{s,j}, a_{s,j+1}, \neg d_{s,i_1}, d_{s,i_2}) \qquad s < d, j \in F, (i_1, i_2) \in O_j^{=}(X) \tag{4.4}$$

$$(\neg a_{s,j}, a_{s,j+1}, d_{s,\#_j^1}) \qquad s < d, j \in F \tag{4.5}$$

$$(\neg c_{t,l_1}, \neg c_{t,l_2}) \qquad t \in \mathcal{N}_T, l_1, l_2 \in [1..k] \tag{4.6}$$

$$(\bigvee_{l \in L} c_{t,l}) \qquad t \in \mathcal{N}_T \tag{4.7}$$

$$(\bigvee_{s \in A_L(t)} \neg d_{s,i}, \bigvee_{s \in A_R(t)} d_{s,i}, c_{t,y_i}, \neg o_i) \qquad t \in \mathcal{N}_T, x_i \in X \tag{4.8}$$

In order to model the objective, we include the soft clauses in Eq. (4.9) with unit weights, which represent correctly classifying as many training points as possible.

$$(o_i) \qquad x_i \in X \tag{4.9}$$

The number of clauses in Eq. (4.1) to Eq. (4.5) is in $O(d|X||F|)$, in Eq. (4.6) and Eq. (4.7) in $O(k^2|\mathcal{N}_T|)$, in Eq. (4.8) in $O(|\mathcal{N}_T||X|)$, and in Eq. (4.9) in $O(|X|)$, resulting in the total number of clauses to be in $O(|X|(d|F| + \mathcal{N}_T))$ if $|X| \geq k^2$.

4.3.3 Decoding

An assignment to the variables in Section 4.3.1 which is satisfying with regard to the hard clauses in Section 4.3.2 is decoded into a reduced BDD in two steps.

In the first step, the assignment is decoded into a non-reduced BDD. We start by extracting the values for the feature selection and threshold selection functions for the branching sequence of length d, and the label selection function for the 2^d terminals. Specifically, for each branching s along the sequence, we set $\beta(s)$ to the number of $a_{s,j}$ variables that are true. With the feature $\beta(s)$ chosen, we set $\alpha(s) = x_i[j]$ where $(i, i') \in O_j(X)$ are consecutive data points according to the ordering of feature j such that $s_{i,s}$ is true and $s_{i',s}$ is false. If no such pair exists, we simply set $\alpha(s)$ to the maximum value of feature $\beta(s)$. For each terminal node t, we set $\theta(t) = l$ if the variable $c_{t,l}$ is true. Note that this assignment is valid since Eq. (4.6) enforces that for each t, no two $c_{t,l}$ variables can be true at the same time. Lastly, the sequence of branchings is expanded into a non-reduced (tree) BDD. Namely, we learn a BDD $\mathcal{D} = (\mathcal{N}, \beta, \alpha, \theta)$ where $\mathcal{N}$ is a tree of depth d, $\beta(t) = \beta(s)$ and $\alpha(t) = \alpha(s)$ for each branching node $t \in \mathcal{N}_B$ in level s, and $\theta(t)$ for each $t \in \mathcal{N}_T$ is directly decoded from the assignment.

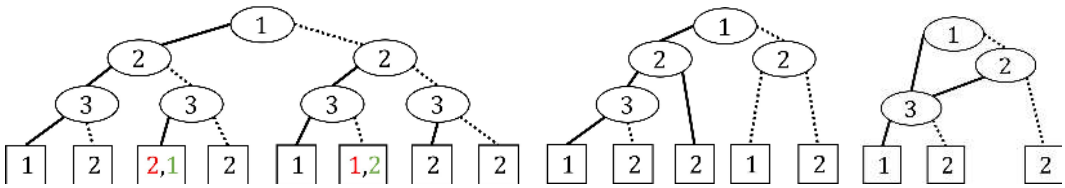
In the second step, the resulting BDD is reduced by merging equivalent nodes and

eliminating redundant nodes. Specifically, we transform $\mathcal{D} = (\mathcal{N}, \beta, \alpha, \theta)$ by iteratively: 1) combining two terminal nodes $t_1, t_2 \in \mathcal{N}_T$ that have the same label $\theta(t_1) = \theta(t_2)$; 2) combining two branching nodes $t_1, t_2 \in \mathcal{N}_B$ that employ the same branching $(\beta(t_1), \alpha(t_1)) = (\beta(t_2), \alpha(t_2))$; 3) replacing a parent node $t \in \mathcal{N}_B$ with its child when its left and right children are the same $l(t) = r(t)$. Note that the reductions are valid since all combined nodes have the same value for β, α, and θ functions and a replaced node has only one child.

4.4 BDD Size Optimization

In the encoding presented in Section 4.3, not all terminals of a solution are guaranteed to contain training points. Thus, we may end up with empty terminals, which we can relabel without affecting the training accuracy. However, the labels of such terminals can affect the prediction on unseen datapoints as well as the size of the final reduced BDD. In Hu, Huguet, and Siala [124], the labels of empty terminals are decided arbitrarily by the solver and they investigate the impact of two post-processing relabelling heuristics on the testing accuracy (i.e., accuracy on unseen data). The first heuristic assigns the majority label of the terminal's first non-empty ancestor. The second heuristic finds and merges equivalent nodes in a greedy top-down search. Hu, Huguet, and Siala's experiments showed that the heuristics do not have significant impact on the testing accuracy. However, we demonstrate how these heuristics can lead to sub-optimal size in Example 3.

Example 3. *In this example, we provide two BDDs and highlight their empty terminals with different options. In Figure 4.2, the BDD is reduced by assigning labels to terminals using the non-empty ancestor heuristic. Reduction by an alternative labelling is then presented to show that the heuristic is not guaranteed to optimize compactness. In Figure 4.3, the BDD is reduced by assigning labels to terminals using the greedy top-down search. Similarly, reduction by an alternative labelling is presented to show that the greedy approach is not guaranteed to result in the lowest size.*

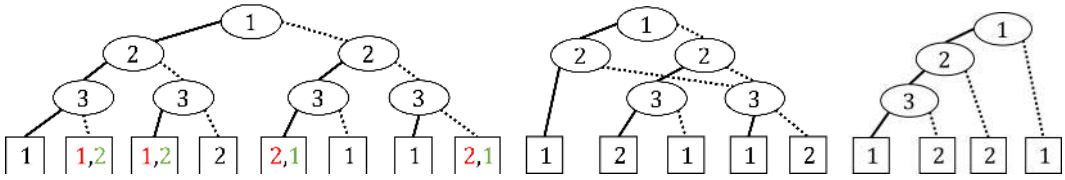


(a) Non-reduced. Black (singular) labels correspond to non-empty terminals, red (left) labels correspond to the heuristic, green (right) labels correspond to the alternative labelling.

(b) Reduced via the non-empty ancestor heuristic labelling.

(c) Reduced via the alternative labelling.

Figure 4.2: A non-reduced BDD, its heuristic-based reduction, and an alternative reduction. Branching nodes are labelled with their level and solid (dotted) lines correspond to the left (right) child. Terminal labels are not combined in the reduction for clarity.



(a) Non-reduced. Black labels correspond to non-empty terminals, red labels correspond to the heuristic, green labels correspond to the alternative labelling.

(b) Reduced via greedy top-down labelling.

(c) Reduced via the alternative labelling.

Figure 4.3: A non-reduced BDD, its greedy-based reduction, and an alternative reduction. Branching nodes are labelled with their level and solid (dotted) lines correspond to the left (right) child. Terminal labels are not combined in the reduction for clarity.

In this section, we propose to use a combinatorial optimization problem for solving the same problem as in Section 4.3, with additional compactness considerations via deciding the labels of empty terminals. Specifically, we aim to model the size of the BDD after the merging of its equivalent nodes and the removal of its redundant nodes. The variables and clauses introduced in this section can be added to the encoding in Section 4.3 for the size to be optimized as a secondary objective (i.e., 1-stage approach). Alternatively, to optimize the size as a post-processing stage (i.e., 2-stage approach), we can use $c_{t,l}$ variables and the clauses in Eq. (4.6) and Eq. (4.7) and disregard the rest of the encoding. In the post-processing stage, we only model the assignment of labels to terminals. For the non-empty terminals, we fix their assignment to the labels that have been decoded for them in the first stage. Consequently, the post-processing stage focuses on optimizing the size of the solution while maintaining its accuracy on the training data. In Section 4.4.1, we introduce variables to represent the size of the reduced BDD. In Section 4.4.2, we present clauses to guarantee the validity of size optimization. In Section 4.4.3, we prove that our size optimization is optimal and show how it affects decoding the solution.

4.4.1 Variables and Structure

To optimize the size of a BDD, we need to model the reduction process. Since it is challenging to model an iterative process in SAT, we equivalently model whether each branching node will be eliminated at some point in the reduction process or not. Note that since the BDD is ordered and the reduction is subject to a branching sequence, that is learned in the same or a prior stage, the equivalency and redundancy of branching nodes are solely dependent on the sequence of terminal labels. A *descendent sequence* of labels corresponding to a branching node is naturally constructed as the order of appearance in the depth-first search of that node with the left outgoing edges prioritized. The sequence label of the BDD is the descendent sequence of the root. If the descendent labels of a branching node consists of two equal substrings, the node is redundant and can be replaced with one of its children. Moreover, if the descendent labels of two branching nodes are equal, they are equivalent and

can be merged. We explicitly define variables to model the redundancy and equivalency of branching nodes. Note that redundancy results in being eliminated in reduction. With a set of equivalent nodes however, all but one are eliminated in the reduction. Thus, we impose an order in our modelling of equivalency so that the first equivalent node is not considered to be eliminated.

We define two sets to help with the structure of the encoding. We first define a set that intuitively contains all pairs of terminals where comparing their labels impacts whether two branching nodes are equivalent. The set E contains all tuples $(\hat{t}_1, \hat{t}_2, t_1, t_2, \underline{t}_1, \underline{t}_2, s)$ with branching nodes $\{\hat{t}_1, \hat{t}_2\}$, terminal nodes $\{t_1, t_2, \underline{t}_1, \underline{t}_2\}$, and $1 \leq s < d$ such that $\hat{t}_1$ ($\hat{t}_2$) is a branching node at level $d - s$ with $\underline{t}_1$ ($\underline{t}_2$) as the first terminal in its descendent sequence and t_1 and t_2 appear in the same position of the two descendent sequences corresponding to $\hat{t}_1$ and $\hat{t}_2$. Members of E are limited to cases where $\underline{t}_2$ appears after $\underline{t}_1$ in the terminal sequence.[2]

$$E = \{(\hat{t}_1, \hat{t}_2, t_1, t_2, \underline{t}_1, \underline{t}_2, s) \mid \exists\, 0 \leq w < 2^s : \underline{t}_1 = l^s(\hat{t}_1), t_1 = \underline{t}_1^{+w}, \underline{t}_2 = l^s(\hat{t}_2), t_2 = \underline{t}_2^{+w}, \underline{t}_1 < \underline{t}_2\}$$

Next, we define a set that intuitively contains all pairs of terminals where comparing their labels impacts whether a branching node is redundant. The set R contains all tuples $(\hat{t}, t_1, t_2, \underline{t}, s)$ with branching node $\hat{t}$, terminal nodes $\{t_1, t_2, \underline{t}\}$, and $1 \leq s < d$ such that $\hat{t}$ is at level $d - s - 1$, t_1 and t_2 appear in the same position of two descendent sequences corresponding to the two children of $\hat{t}$, and $\underline{t}$ appears at the beginning of the descendent sequence of $\hat{t}$ itself:

$$R = \{(\hat{t}, t_1, t_2, \underline{t}, s) \mid \exists\, 0 \leq w < 2^s : t_1 = l^{s+1}(\hat{t})^{+w}, t_2 = l^s(r(\hat{t}))^{+w}, \underline{t} = l^{s+1}(\hat{t})\}$$

The following set of binary variables represents redundancy and equivalency of branching nodes by modelling repetition in terminal labels and their sequences:

- $\{\sigma_{t_1,t_2} \mid (\hat{t}_1, \hat{t}_2, t_1, t_2, \underline{t}_1, \underline{t}_2, s) \in E\}$: Represents whether terminals t_1 and t_2 have been assigned different labels. It is equivalent to $\theta(t_1) \neq \theta(t_2)$.
- $\{b_{\underline{t},\Delta} \mid (\hat{t}, t_1, t_2, \underline{t}, s) \in R, \Delta = 2^{s+1}\}$: Represents whether the sequence of Δ labels starting from terminal node $\underline{t}$ (inclusive) cannot be divided into two equal subsequences. It is equivalent to $\hat{t}$ not being redundant.
- $\{r_{\underline{t}_1,\underline{t}_2,\Delta} \mid (\hat{t}_1, \hat{t}_2, t_1, t_2, \underline{t}_1, \underline{t}_2, s) \in E, \Delta = 2^s\}$: Represents whether the sequence of Δ labels starting from terminal node $\underline{t}_1$ is equal to the sequence of Δ labels starting from terminal node $\underline{t}_2$ (both inclusive). It is equivalent to $\hat{t}_1$ and $\hat{t}_2$ being equivalent.

[2] The notation $l^s(.)$ stands for applying the $l(.)$ function s times. Furthermore, t^{+w} stands for starting from terminal t and moving across the terminal sequence of the BDD by w steps. Lastly, $t_1 < t_2$ stands for t_2 appearing to the right of t_1 in the terminal sequence.

Note that there is redundancy in using R and E to define $b_{\underline{t},\Delta}$ and $r_{\underline{t}_1,\underline{t}_2,\Delta}$ variables, respectively. Namely, multiple members of the two sets with different t_1 and t_2 values correspond to the definition of one variable. Furthermore, note that variables σ_{t_1,t_2}, $b_{\underline{t},\Delta}$, and $r_{\underline{t}_1,\underline{t}_2,\Delta}$ are not defined for all possible index combinations. Specifically, Δ always refers to the length of a branching node's descendent sequence of terminals (a power of 2), and alongside a terminal node as the starting point, has a one-to-one correspondence to a branching node. Consequently, $b_{\underline{t},\Delta}$ is used to represent that the corresponding branching node at level $d - log_2(\Delta)$ is not redundant. Moreover, $r_{\underline{t}_1,\underline{t}_2,\Delta}$ is used to represent that the two corresponding branching nodes at level $d - log_2(\Delta)$ are equivalent. Lastly, σ_{t_1,t_2} is used when there are two branching nodes whose descendent sequences contain t_1 and t_2, respectively, in the same position. Note that the number of σ_{t_1,t_2} variables is in $O(|\mathcal{N}_T|^2)$, $b_{\underline{t},\Delta}$ in $O(|\mathcal{N}_T|)$, and $r_{\underline{t}_1,\underline{t}_2,\Delta}$ in $O(|\mathcal{N}_T|^2)$, resulting in the total number of variables to be in $O(|\mathcal{N}_T|^2)$.

4.4.2 Clauses

We propose the following sets of CNF clauses for modelling the size of a reduced BDD given its terminal labels. The clauses in Eq. (4.10), Eq. (4.11), Eq. (4.12) guarantee that σ_{t_1,t_2} variables correctly represent the difference in labels. The clauses in Eq. (4.13) guarantee that each branching node can only be redundant if all of the corresponding terminal pairs in the descendent sequences of its two children have the same label. The clauses in Eq. (4.14) and Eq. (4.15) guarantee that a pair of branching nodes can only be equivalent if their corresponding descendent terminals have the same label sequences.

$$(\neg c_{t_1,l}, \neg c_{t_2,l}, \neg \sigma_{t_1,t_2}) \qquad (\hat{t}, t_1, t_2, \underline{t}, s) \in R, l \in [1..k] \tag{4.10}$$

$$(\neg c_{t_1,l}, c_{t_2,l}, \sigma_{t_1,t_2}) \qquad (\hat{t}, t_1, t_2, \underline{t}, s) \in R, l \in [1..k] \tag{4.11}$$

$$(c_{t_1,l}, \neg c_{t_2,l}, \sigma_{t_1,t_2}) \qquad (\hat{t}, t_1, t_2, \underline{t}, s) \in R, l \in [1..k] \tag{4.12}$$

$$(b_{\underline{t},2^{s+1}}, \neg \sigma_{t_1,t_2}) \qquad (\hat{t}, t_1, t_2, \underline{t}, s) \in R \tag{4.13}$$

$$(\neg r_{\underline{t}_1,\underline{t}_2,2^s}, \neg c_{t_1,l}, c_{t_2,l}) \qquad (\hat{t}_1, \hat{t}_2, t_1, t_2, \underline{t}_1, \underline{t}_2, s) \in E, l \in [1..k] \tag{4.14}$$

$$(\neg r_{\underline{t}_1,\underline{t}_2,2^s}, c_{t_1,l}, \neg c_{t_2,l}) \qquad (\hat{t}_1, \hat{t}_2, t_1, t_2, \underline{t}_1, \underline{t}_2, s) \in E, l \in [1..k] \tag{4.15}$$

To model our objective, we include the soft clauses in Eq. (4.16) where each clause represents the corresponding branching node $\hat{t}_1$ to be either redundant, or equivalent to another node appearing before it in the indexing order. Maximizing the number of redundant or equivalent branching nodes will consequently minimize the size of our reduced BDD as those are the nodes that will be eliminated in the reduction. Note that if size is being considered as a secondary objective to accuracy, the clauses in Eq. (4.16) can be weighted against the clauses in Eq. (4.9) proportionately. Otherwise, if size is our second-stage

objective, we can use the clauses in Eq. (4.16) with unit weights.

$$\Big(\bigvee_{\underline{t}_2 \in E(\hat{t}_1)} r_{\underline{t}_2,\underline{t}_1,\Delta}, \neg b_{\underline{t}_1,\Delta}\Big) \qquad \hat{t}_1 \in \mathcal{N}_B \tag{4.16}$$

where $\underline{t}_1$ and Δ are uniquely determined by finding a member $(\hat{t}_1, ., ., \underline{t}_1, s)$ in R with $\Delta = 2^{s+1}$, and $E(\hat{t}_1)$ represents the set of all $\underline{t}_2$ where a member $(\hat{t}_1, ., ., ., ., \underline{t}_2, .)$ can be found in E and $\underline{t}_2$ appears before $\underline{t}_1$ in the terminal sequence. Intuitively, $\underline{t}_1$ represents the start of the descendent sequence corresponding to $\hat{t}_1$ and each $\underline{t}_2$ represents the same for each branching node of the same level that appears before $\hat{t}_1$.

The number of clauses in Eq. (4.10) to Eq. (4.15) is in $O(|\mathcal{N}_T|^2)$ and in Eq. (4.16) in $O(|\mathcal{N}_B|)$, respectively. Thus, the total number of added clauses due to size optimization is in $O(|\mathcal{N}_T|^2)$.

4.4.3 Soundness and Decoding

We first show that our size optimization is sound, meaning that maximizing the number of equivalent and redundant nodes is equivalent to minimizing the size of the resulting BDD after reduction.

Proposition 4. *Given a non-reduced (tree) BDD $\mathcal{D} = (\mathcal{N}, \beta, \alpha, \theta)$ and its reduced equivalent $\mathcal{D}' = (\mathcal{N}', \beta', \alpha', \theta')$, the number of branching nodes in $\mathcal{D}$ that are redundant or equivalent to a previous node, is equal to $|\mathcal{N}_B| - |\mathcal{N}'_B|$.*

Proof. To prove Proposition 4, we need to show, in both directions, that each redundant or equivalent node corresponds to an eliminated node in the reduction. Assume that branching node $t \in \mathcal{N}_B$ is eliminated in the reduction:

- If it is replaced with one of its children, the descendent sequences of its two children must be the same. Thus, t was redundant in $\mathcal{D}$.
- If it is merged with one of the branching nodes in the same level, one of them must appear sooner in the indexing order. Thus, the node that appears later, which is considered eliminated, was equivalent to a previous node in $\mathcal{D}$.

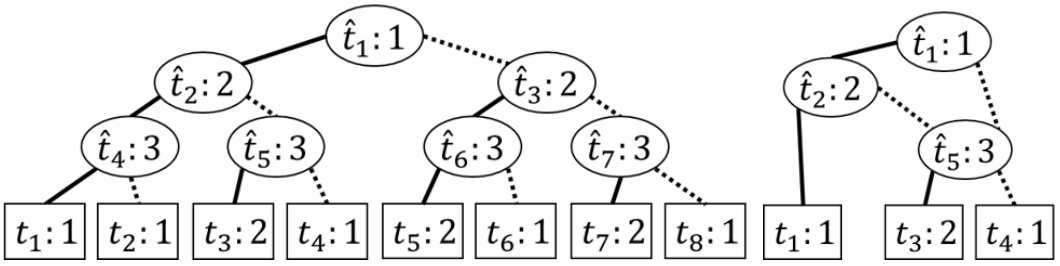
For the opposite direction, we describe a procedure that reduces the BDD by iteratively merging equivalent nodes and removing redundant ones:

- For each set of branching nodes $T \subseteq \mathcal{N}_B$, remove all but the one that appears first in the indexing order.
- For each level $s \in [0..d-1]$, replace each branching node $t \in \mathcal{N}_B$ that has the same left and right children with its child.

Note that the number of node eliminations in this procedure is equal to the number of nodes that are considered redundant or equivalent to a previous node. □

We have shown our size optimization objective equates to the number of nodes eliminated in a full reduction. Based on Theorem 2, we can conclude that our encoding correctly models the final size of a reduced BDD and can be used for achieving optimal compactness. Note that our size optimization is subject to fixed accuracy and non-empty terminals (2-stage approach) or being weighted against accuracy (1-stage approach).

Figure 4.4 shows a non-reduced BDD and its reduced counterpart. Table 4.1 includes the corresponding size optimization variables, their values, and the branching nodes that they allow us to remove in the reduction. Note that the reduced BDD is produced by removing and merging all of the branching nodes indicated by the table.



(a) Non-reduced. (b) Reduced.

Figure 4.4: Example of optimizing the size of a BDD from its non-reduced version to the reduced counterpart.

Variable	$b_{t_1,2}$	$b_{t_1,4}$	$b_{t_1,8}$	$b_{t_3,2}$	$b_{t_5,2}$	$b_{t_5,4}$	$b_{t_7,2}$
Value	False	True	True	True	True	False	True
Remove	$\hat{t}_4$	-	-	-	-	$\hat{t}_3$	-
Variable	$r_{t_1,t_3,2}$	$r_{t_1,t_5,2}$	$r_{t_1,t_7,2}$	$r_{t_3,t_5,2}$	$r_{t_3,t_7,2}$	$r_{t_5,t_7,2}$	$r_{t_1,t_5,4}$
Value	False	False	False	True	True	True	False
Remove	-	-	-	$\hat{t}_6$	$\hat{t}_7$	$\hat{t}_7$	-

Table 4.1: Size optimization variables corresponding to the BDD in Figure 4.4.

Decoding

To decode the solution, we simply need to do as we did in Section 4.3.3, since the additional variables for size encoding are not involved in structuring the BDD. Once we decode and reduce the solution as before, Proposition 4 guarantees that we will end up with a size that corresponds to our objective value. Specifically, we will have one branching node remaining for every unsatisfied soft clause in Eq. (4.16). The decoding for the 2-stage approach differs from that of the 1-stage approach as it re-learns the values for the θ function after the post-processing stage in which the labels might change.

4.5 Multi-Dimensional BDDs

In Section 4.3 and Section 4.4, we presented the approach to learn BDDs by finding trees and then reducing them to diagrams, with or without optimizing the size of the final reduced BDD. However, the number of clauses in Eq. (4.8) is exponential in the length of the branching sequence and multiplied by the size of the dataset. To alleviate this blow-up in our encoding, we consider learning diagrams, rather than trees, directly. This is in line with our goal of utilizing BDDs as a way to mitigate the exponential size of decision trees. A direct encoding of diagrams can help us consider more expressive solutions in a smaller encoding by utilizing more branchings. In this section, we look at a family of diagrams that contain branchings over multiple features at each level, i.e., multi-dimensional branchings. We define multi-dimensional branchings as a division dictated by a directional inner BDD which has 0 and 1 labels.

Definition 18 (Directional Inner BDD)**.** *Given a set of features F and a dimension $\hat{d}$, a directional inner BDD $\mathcal{D}^{\leftrightarrow}$ is an ordered BDD with a branching sequence of length $\hat{d}$ with* 0 *and* 1 *as labels.*

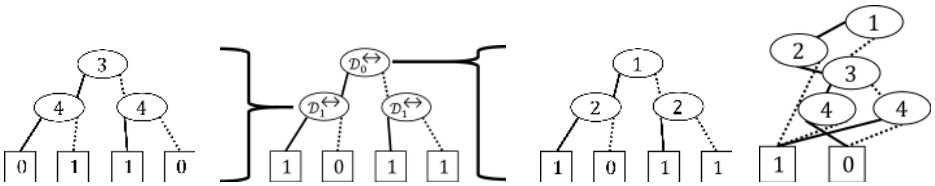
Definition 19 (Multi-dimensional Branching)**.** *Given a finite set of features F, a multi-dimensional branching is a division of data points into two groups using a directional inner BDD $\mathcal{D}^{\leftrightarrow}$. The multi-dimensional branching directs a data point x based on the label assigned by $\mathcal{D}^{\leftrightarrow}$ (*0 *or* 1*). The dimension of a multi-dimensional branching is the dimension of its directional inner BDD.*

A BDD that operates on multi-dimensional branchings is referred to as a multi-dimensional BDD. We formally define multi-dimensional BDDs by augmenting Definition 13 with directional inner BDDs and the χ function which assigns them to branchings.

Definition 20 (Multi-dimensional Binary Decision Diagram)**.** *Given a set of features F and integer labels $[1..k]$, a multi-dimensional binary decision diagram is a tuple $\hat{\mathcal{D}} = (\mathcal{N}, \chi, \theta)$ where $\mathcal{N}$ is a binary diagram structure, the function χ assigns a directional inner BDD to each branching node, and $\theta : \mathcal{N}_T \to [1..k]$ is the terminal labelling function.*

Note that label assignment, size, and reduction properties of multi-dimensional BDDs are defined analogously to standard BDDs. All notions are still applicable since multi-dimensional branchings are the same as standard branchings in dividing the points into two parts. Next, we define the problem of learning multi-dimensional BDDs with maximum accuracy.

Problem 5 (Max-Accuracy Multi-dimensional BDD)**.** *Given a set of features F, integer labels $[1..k]$, a set of training data points $X \subset \mathbb{R}^{|F|}$, a labeling $\gamma : X \to [1..k]$, a maximum number of multi-dimensional branchings d, and a sequence of dimensions $[\hat{d}_0, \hat{d}_1, ..., \hat{d}_{d-1}]$, output a maximum accuracy (Definition 9) ordered multi-dimensional binary decision diagram $\hat{\mathcal{D}}$ with a multi-dimensional branching sequence of length d that corresponds in dimensions to $[\hat{d}_0, \hat{d}_1, ..., \hat{d}_{d-1}]$.*



(a) Directional inner BDD $\mathcal{D}_1^{\leftrightarrow}$. (b) Multi-dimensional BDD $\hat{\mathcal{D}}$. (c) Directional inner BDD $\mathcal{D}_0^{\leftrightarrow}$. (d) Reduced standard BDD $\mathcal{D}$.

Figure 4.5: A multi-dimensional BDD operating on two 2-dimensional directional inner BDDs and its equivalent standard BDD.

A multi-dimensional branching is a generalization of a standard branching, which makes a multi-dimensional BDD at least as expressive as a standard BDD with the same number of branchings. It is not clear to what extent multi-dimensional BDDs are more expressive than standard BDDs. We formally state an upper bound in terms of standard BDDs for the expressiveness of their multi-dimensional variants in Proposition 5.

Proposition 5. *Consider a multi-dimensional BDD $\hat{\mathcal{D}} = (\mathcal{N}, \chi, \theta)$ with a branching sequence of length d corresponding to directional inner BDDs $[\mathcal{D}_0^{\leftrightarrow}, \mathcal{D}_1^{\leftrightarrow}, ..., \mathcal{D}_{d-1}^{\leftrightarrow}]$ of dimensions $[\hat{d}_0, \hat{d}_1, ..., \hat{d}_{d-1}]$. There exists a BDD $\mathcal{D}$ with a branching sequence of length $\sum_{i<d} \hat{d}_i$ that is equivalent to $\hat{\mathcal{D}}$.*

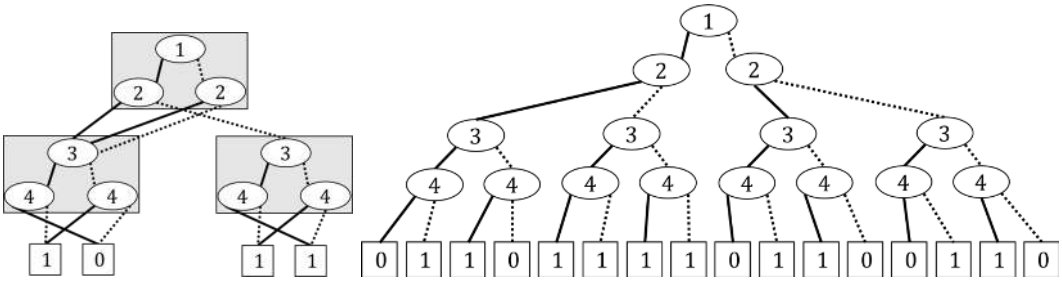
Proof. We prove by construction that $\hat{\mathcal{D}}$ can be unfolded into a standard BDD where branchings correspond to the branchings that are employed within each directional inner BDD.

We let $\mathcal{D}$ be a tree BDD. We extract the β and α values from the branching sequence of each directional inner BDD and concatenate the values to make up the branching sequence of $\mathcal{D}$, we use $\beta(s)$ and $\alpha(s)$ to refer to the branching at position s of the concatenated sequence. Next, we learn the values for the θ function. Given that $\mathcal{D}$ has a tree structure, every terminal node $t \in \mathcal{N}_T$ corresponds to a sequence of 0 and 1 values to branchings that will direct a point to this terminal. For a particular $t \in \mathcal{N}_T$, we take this sequence, divide it into sub-sequences corresponding to each directional inner BDD, retrieve the sequence of labels that each sub-sequence leads to $[l_0, l_1, ..., l_{d-1}]$, feed the sequence of labels as outputs of multi-dimensional branchings to $\hat{\mathcal{D}}$, and retrieve the output label as the $\theta(t)$ value.

Note that our construction guarantees that given the same value for standard branchings, the outputs of $\mathcal{D}$ and $\hat{\mathcal{D}}$ are the same. Therefore, the two BDDs are guaranteed to assign the same label to every data point x.

Note that the resulting tree can still be reduced into a smaller diagram following the standard procedure. □

In Figure 4.5, we depict the procedure from the proof of Proposition 5 which constructs an equivalent standard BDD from a multi-dimensional BDD and its two inner directional BDDs. Figure 4.6 further elaborates on the procedure by showing how the multi-dimensional BDD is operating internally and how it initially unfolds into a non-reduced standard BDD.



(a) Inner operations of the multi-dimensional BDD.

(b) Unfolding of the multi-dimensional BDD.

Figure 4.6: The unfolding procedure for the multi-dimensional BDD from Figure 4.5.

We conclude from Proposition 5 that, in terms of expressiveness, a multi-dimensional BDD with dimensions $[\hat{d}_0, \hat{d}_1, ..., \hat{d}_{d-1}]$ is interposed between two BDDs. The lower bound BDD has a branching sequence of length d and the upper bound BDD has a branching sequence of length $\sum_{i<d} \hat{d}_i$. Previous works have considered other representations that are designed to be more expressive than BDDs, such as sentential decision diagrams [68] or read-k-times branching programs [47, 152].

Next, we present an encoding for learning multi-dimensional BDDs and solve Problem 5. In Section 4.5.1, we introduce variables that encode the sequence of directional inner BDDs. In Section 4.5.2, we describe how the hard clauses in our encoding are modified for leaning multi-dimensional BDDs. In Section 4.5.3, we show how the modified encoding is decoded into a solution.

4.5.1 Variables

In order to encode a multi-dimensional BDD, we use the same set of variables as Section 4.3.1 but remove $a_{s,j}$ variables since a multi-dimensional branching corresponds to a directional inner BDD, not simply a selected feature. To more clearly structure the new variables, we define the set S_H which contains all pairs of indices along the sequence of multi-dimensional branchings, and indices along the branching sequence of the corresponding directional inner BDD:

$$S_H = \{(s, h) \,|\, 0 \leq s < d, 0 \leq h < \hat{d}_s\}$$

We add the following variables to encode the inner-workings of each directional inner BDD, similar to how we encode standard BDDs. Specifically, we introduce variables to encode the values of β_s, α_s, and θ_s functions, for each directional inner BDD that corresponds to position s of the multi-dimensional branching sequence:

- $\{\widehat{a}_{(s,h),j} \mid (s, h) \in S_H, j \in F\}$: Represents whether the feature chosen at branching h of directional inner BDD s is or comes before j. It is equivalent to $\beta_s(h) \leq j$.

- $\{\widehat{d}_{(s,h),i}(s,h) \in S_H, x_i \in X\}$: Represents whether point x_i is directed to left at branching h of directional inner BDD s. It is equivalent to $x_i[\beta_s(h)] \leq \alpha_s(h)$.
- $\{\widehat{c}_{s,t} \mid 0 \leq s < d, t \in \mathcal{N}_T\}$: Represents whether terminal t of directional inner BDD s is assigned the label 1. It is equivalent to $\theta_s(t) = 1$.

Note that since a directional inner BDD only has two output labels, we do not need to have one $\widehat{c}_{s,t}$ variable for each output. It suffices to have one that represents a label of 1 while its negation represents a label of 0. Note that the number of variables required for each directional inner BDD is proportional to a standard BDD with the same number of branchings, i.e., $O(d|X|)$.

4.5.2 Clauses

We discard clauses in Eq. (4.1) to Eq. (4.5) from the original encoding since standard branchings are no longer employed in the multi-dimensional BDD. We keep the clauses Eq. (4.6) to Eq. (4.9) however, since we still need to model labels and terminals containing points.

To complete our encoding, we add CNF clauses for modelling the directional inner BDDs, similar to how we modelled BDDs in our original encoding. The label that is assigned to a data point by a directional inner BDD is then used to determine the direction that the point will be guided towards in a multi-dimensional branching. The clauses in Eq. (4.17) and Eq. (4.18) guarantee that one feature is selected at each branching of each directional inner BDD by enforcing the ordered encoding of $\widehat{a}_{(s,h),j}$ variables. The clauses in Eq. (4.19), Eq. (4.20), and Eq. (4.21) guarantee that each branching of each directional inner BDD conforms to a pairing of selected feature and threshold. The clauses in Eq. (4.22) and Eq. (4.23) guarantee that the direction of a point in a multi-dimensional branching matches the label of its containing terminal in the corresponding directional inner BDD. The clauses in Eq. (4.24) guarantee the leftmost terminal in each directional inner BDD (represented by the shorthand 0) to be assigned the 0 label. These clauses play a role analogous to that of the clauses in Eq. (4.5) from the original encoding, as they are both intended to break symmetry within branchings (see Section 3.3.2 for further discussion). Unlike the original encoding, enforcing a value of 0 here is not guaranteed to impact data points, since the leftmost terminal is not guaranteed to contain any points.

$$(\widehat{a}_{(s,h),j}, \neg\widehat{a}_{(s,h),j+1}) \qquad (s,h) \in S_H, j \in F \quad (4.17)$$

$$(\widehat{a}_{(s,h),0}) \qquad (s,h) \in S_H \quad (4.18)$$

$$(\neg\widehat{a}_{(s,h),j}, \widehat{a}_{(s,h),j+1}, \widehat{d}_{(s,h),i_1}, \neg\widehat{d}_{(s,h),i_2}) \qquad (s,h) \in S_H, j \in F, (i_1,i_2) \in O_j(X) \quad (4.19)$$

$$(\neg\widehat{a}_{(s,h),j}, \widehat{a}_{(s,h),j+1}, \neg\widehat{d}_{(s,h),i_1}, \widehat{d}_{(s,h),i_2}) \qquad (s,h) \in S_H, j \in F, (i_1,i_2) \in O_j^{=}(X) \quad (4.20)$$

$$(\neg\widehat{a}_{(s,h),j}, \widehat{a}_{(s,h),j+1}, \widehat{d}_{(s,h),\#_j^1}) \qquad (s,h) \in S_H, j \in F \quad (4.21)$$

$$(\bigvee_{h \in A_R(s,t)} \widehat{d}_{(s,h),i}, \bigvee_{h \in A_L(s,t)} \neg\widehat{d}_{(s,h),i}, d_{s,i}, \neg\widehat{c}_{s,t}) \qquad 0 \leq s < d, x_i \in X, t \in \mathcal{N}_T^s \quad (4.22)$$

$$(\bigvee_{h \in A_R(s,t)} \widehat{d}_{(s,h),i}, \bigvee_{h \in A_L(s,t)} \neg\widehat{d}_{(s,h),i}, \neg d_{s,i}, \widehat{c}_{s,t}) \qquad 0 \leq s < d, x_i \in X, t \in \mathcal{N}_T^s \quad (4.23)$$

$$(\widehat{c}_{s,0}) \qquad 0 \leq s < d \quad (4.24)$$

where $\mathcal{N}_T^s$ contains the terminals of the directional inner BDD at position s, $A_R(s,t)$ ($A_L(s,t)$) contains the right (left) ancestors of terminal $t \in \mathcal{N}_T^s$, and $\widehat{a}_{(s,h),j+1}$ for $j = max(F)$ is assumed to be the false constant.

Note that the number of clauses in Eq. (4.22) to Eq. (4.24) is in $O(|X| \sum_s |\mathcal{N}_T^s|)$ where $\sum_s |\mathcal{N}_T^s|$ equals the sum of each dimension to the power of two. The number of clauses in Eq. (4.17) to Eq. (4.21) are proportionate to a standard BDD with the same number of branchings.

4.5.3 Decoding

A satisfying assignment to the variables of our multi-dimensional BDD encoding can be decoded into a tree BDD operating on multi-dimensional branchings. Specifically, the variables that we kept from the original encoding are decoded as before to make up the θ function for the multi-dimensional BDD. Moreover, for every s value (a multi-dimensional branching along the sequence) the values of the corresponding variables presented in Section 4.5.1 are decoded into a directional inner BDD, analogous to our decoding of standard BDDs.

In order to further transform the result into a BDD operating on standard branchings, we can transform the multi-dimensional BDD according to the procedure described in the proof of Proposition 5. Note that the number of branchings in the resulting BDD is equal to the sum of dimensions from the multi-dimensional BDD. However, we might be able to reduce the transformed BDD in ways that were not possible in the multi-dimensional format. For example, two directional inner BDDs might employ the exact same branching, leading to the complete elimination of a level when reducing the transformed BDD.

The size optimization presented in Section 4.4 can also be utilized as a secondary stage to learning multi-dimensional BDDs. Namely, after the solution is decoded and transformed into a standard BDD, the resulting BDD can be used to assign labels to all of the training data points to determine the terminals that do not contain any points. Once the empty terminals are detected, we can adjust the size optimization encoding by setting $d = \sum_{i<d} \hat{d}_i$ and

proceed as before. Additionally, we may want to enforce that there is a multi-dimensional BDD employing our learned multi-dimensional branchings, that is equivalent to our standard BDD after optimizing the size. In that case, we can add straight-forward clauses to our size encoding that guarantees transformed terminals that correspond to the same terminal in the original multi-dimensional BDD, are assigned the same label.

4.6 Experimental Setup

In this section, we describe the setup of our experiments to evaluate our BDD encoding. Namely, we discuss implementation details, the state-of-the-art baseline against which we compare, and the datasets for which we learn BDDs.

4.6.1 Implementation

We use the Java programming language to produce encodings and the Loandra MaxSAT solver [34] to solve each instance. We set the solver timeout limit to 15 minutes and use the best found solution in case of a timeout. Our experiments are run on an AMD EPYC 7502 32-core processor and 256GB of RAM.

4.6.2 Baseline

We compare our approach against the recent work on SAT-based learning of BDDs. Hu, Huguet, and Siala [124] aims to find BDD classifiers with maximum accuracy similar to our approach. However, unlike our approach, they only support binary classification and require pre-processing of each numerical feature into a set of binary features. The pre-processing for binarization of features matches that used by the decision tree baselines discussed in Section 3.7.2, and follows the procedure described in Example 2. Hu, Huguet, and Siala focuses solely on optimizing accuracy and does not explicitly model the size of the reduced BDD. To decide the labels of empty terminals, they use the heuristics described in Example 3 alongside arbitrary assignment of labels done by the solver as three candidate methods.

Note that Hu, Huguet, and Siala [124, 125] have compared optimal BDDs against optimal decision trees and heuristic decision trees and found that BDDs are competitive in terms of testing accuracy and have smaller size. Our approach aims to improve the performance of Hu, Huguet, and Siala and learn more compact BDD classifiers with comparable accuracy. We therefore compare our approach to Hu, Huguet, and Siala in terms of runtime, accuracy, and size of BDDs.

4.6.3 Datasets

We run our experiments on 11 datasets from the UCI repository [77] covering different number of labels, both numerical and binary features, and different dataset sizes. Table 4.2 reports the datasets with their number of data points $|X|$, number of numerical features

$|F_N|$, number of binary features $|F_B|$, number of features after explicit binarization in the baseline $\tilde{f}$, and number of class labels k. Following the discussion in Section 3.7.3, we highlight that the datasets for the experiments in this chapter are also selected with feature interpretability in mind.

Name	$\|X\|$	$\|F_N\|$	$\|F_B\|$	$\tilde{f}$	k
Banknote	1372	4	0	5016	2
Breast Cancer	116	9	0	891	2
Cryotherapy	90	5	1	93	2
Immunotherapy	91	6	1	166	2
Ionosphere	351	32	2	8114	2
Iris	150	4	0	119	3
User Knowledge	258	5	0	431	4
Vertebral Column	310	6	0	1741	2
Wine	178	13	0	1263	3
Car	1728	6	0	15	2
Monk2	169	4	2	11	2

Table 4.2: Description of the datasets used in our experiments.

4.7 Experimental Results

In this section, we perform studies to experimentally analyze the performance of our encoding for learning maximum accuracy BDDs (Section 4.3), BDDs with size optimization (Section 4.4), and multi-dimensional BDDs (Section 4.5). We compare the performance of our approach against state-of-the-art BDD learning baseline and investigate the trade-off between size, accuracy, and performance in 1-stage, 2-stage, and multi-dimensional approaches to learning compact diagrams. Throughout the experiments, we seek to find out whether compactness can be achieved without significant compromise, and investigate its impact on testing accuracy. Furthermore, we aim to understand if the added expressiveness of multi-dimensional BDDs allows us to achieve high quality solutions with lower number of branchings.

4.7.1 Comparison Against Baseline

In our first set of results, we compare the performance of our approach in terms of runtime and solution quality against Hu, Huguet, and Siala for different numbers of branchings. Note that while our approach learns multi-terminal BDDs, Hu, Huguet, and Siala's approach learns standard BDDs and is unable to support datasets with more than two labels. Furthermore, since the objective in Hu, Huguet, and Siala does not include compactness considerations, we similarly use our base encoding from Section 4.3 that only optimizes accuracy. As both approaches explore the same space of (feasible and) optimal BDD solutions, we focus our comparison on optimization performance (i.e., training accuracy and runtime).

Dataset	d	Accuracy (%)		Time (s)	
		Ours	Baseline [124]	Ours	Baseline [124]
Banknote	4	96.8	96.8	843.01	TO
	5	97.6	90.4	TO	TO
	6	98.5	91.8	TO	TO
Breast Cancer	4	89.7	89.7	TO	TO
	5	92.2	91.4	TO	TO
	6	94.8	94.8	TO	TO
Cryotherapy	4	97.8	97.8	1.23	2.43
	5	98.9	98.9	3.97	9.41
	6	100	100	0.44	1.64
Immunotherapy	4	95.6	95.6	8.18	26.65
	5	96.7	96.7	74.08	290.99
	6	97.8	97.8	433.35	TO
Ionosphere	4	94.9	90.6	TO	TO
	5	95.2	85.8	TO	TO
	6	96.6	90.6	TO	TO
Iris	4	98.7	-	0.67	-
	5	99.3	-	0.62	-
	6	100	-	0.43	-
User Knowledge	4	94.2	-	54.57	-
	5	95.7	-	828.34	-
	6	97.7	-	TO	-
Vertebral Column	4	88.1	87.7	TO	TO
	5	89.7	90	TO	TO
	6	91	90.3	TO	TO
Wine	4	99.4	-	29.29	-
	5	100	-	2.07	-
	6	100	-	1.14	-
Car	4	92.5	92.5	316.88	TO
	5	92.9	92.9	TO	TO
	6	95.4	95.4	TO	TO
Monk2	4	74.6	74.6	85.49	473.43
	5	84.6	84.6	185.3	416.37
	6	100	100	0.35	1.09

Table 4.3: Results for learning max-accuracy BDDs.

Based on the results presented in Table 4.3, we see that our approach is able to achieve a higher than or equal to Hu, Huguet, and Siala [124] accuracy in all but one case. Furthermore, the runtimes of non-timeout cases show that our approach can also prove optimality much faster. As we expected given our direct encoding of non-binary values, the improvement is most noticeable in datasets with highly numerical features, namely Banknote and Ionosphere. For highly numerical datasets, we have observed Hu, Huguet, and Siala to

struggle even at generating a SAT instance without solving it.

4.7.2 1-Stage and 2-Stage Size Optimization

Next, we evaluate the encoding presented in Section 4.4 to minimize the size of our learned BDDs. We perform size optimization in 1-stage and 2-stage approaches. In the 1-stage approach, we use different values for the weight of the size objective against the accuracy objective. Given a weight ξ and d branchings, the combined size and accuracy objective aims to find a solution $\mathcal{D}$ with maximum $\xi.acc(\mathcal{D}) - \frac{|\mathcal{N}_B^{\mathcal{D}}|}{(2^d-1)}$, where $\mathcal{N}_B^{\mathcal{D}}$ is the set of branching nodes in the reduced version of $\mathcal{D}$ and $acc(\mathcal{D})$ represents its accuracy. We divide the size of $\mathcal{N}_B^{\mathcal{D}}$ by its maximum value to normalize its contribution to the objective function. Both components of the objective have a range of values from 0 to 1.

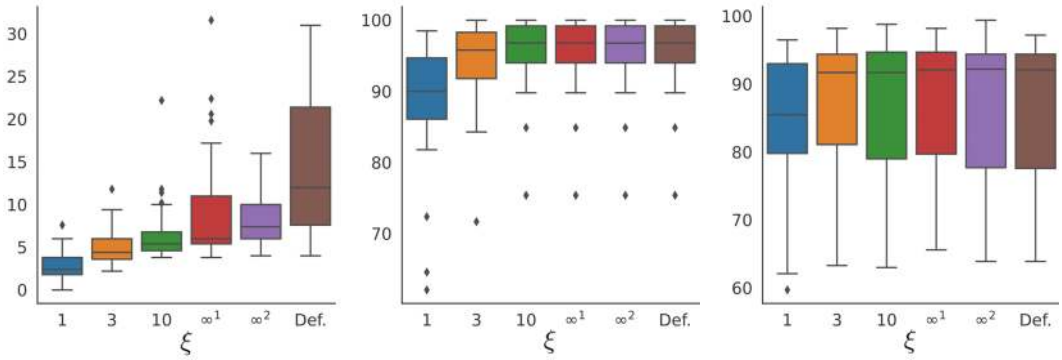
We use $\xi = \infty^2$ to denote the 2-stage approach, $\xi = \infty^1$ to denote that accuracy is completely prioritized over size in the 1-stage approach, and Def. to denote the approach without any size optimization. Note that even in the 1-stage approach, running the second stage can further optimize the size if the first stage have yielded a sub-optimal solution (due to a timeout). However, we opt against mixing the two approaches for the sake of clarity in comparison between the two. In contrast to the previous experiment in which we only focused on training accuracy, we use 5-fold cross validation and report the average value across folds to understand the effects of compactness on generalization and testing accuracy.

The results are presented in Figure 4.7. As expected, we see increase in training accuracy and size as we shift the priority to accuracy by first increasing the ξ value and then changing to ∞^1, ∞^2, and Def., respectively. However, the improvement in accuracy stagnates while the size continues to grow, indicating that a large enough ξ value can act as complete prioritization of accuracy. Interestingly, testing accuracy stagnates sooner and suffers from a larger variability as ξ increases, demonstrating a limited but positive effect of compactness on testing accuracy and generalization. Note that $\xi = \infty^1$ is guaranteed to lead to a size that is equal or lower compared to $\xi = \infty^2$ if all stages are run to optimality. In practice however, $\xi = \infty^2$ and $\xi = \infty^1$ cases lead to competitive sizes due to the fact that most of the 1-stage runs are timed out.

The solutions for lower ξ values in the 1-stage approaches are significantly smaller compared to the $\xi = \infty^2$ case, highlighting the importance of adding size considerations while the solution is being learned towards maximum accuracy. However, optimizing size as a second stage still provides a significant size reduction compared to the case with no size optimization. Table 4.4 provides a detailed report of the results per dataset. Note that in some cases a BDD of size 0 is achieved, which indicates that all branching nodes are redundant and the BDD is equivalent to a singular terminal.

4.7.3 Multi-Dimensional BDDs

In our next set of experiments, we evaluate our approach for directly learning multi-dimensional BDDs. Our goal is to understand whether the added expressiveness of multi-



(a) Size. (b) Training Accuracy. (c) Testing Accuracy.

Figure 4.7: Distributional results for optimizing BDDs in size and accuracy across datasets for different ξ values.

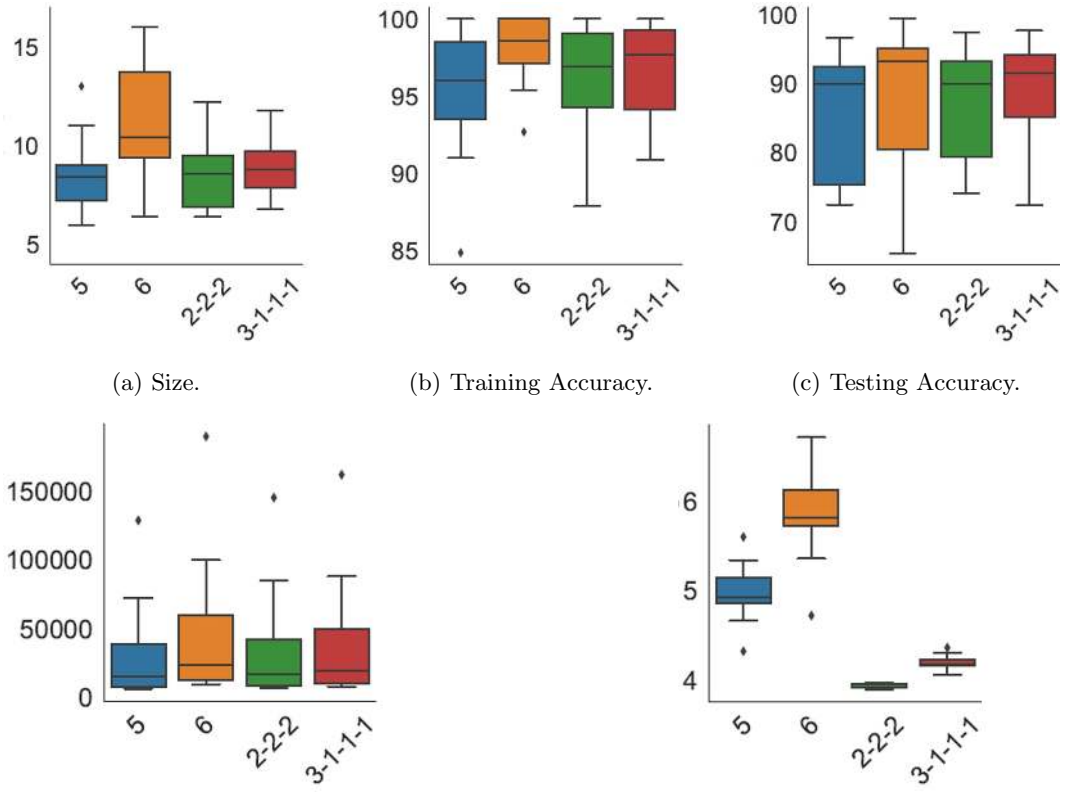
dimensional BDDs allows us to find higher quality solutions in compact sizes. Furthermore, we study how the division of dimensions along a sequence of multi-dimensional branchings can affect the performance and the quality of the solution. We consider one balanced BDD with three multi-dimensional branchings (2-2-2) and one unbalanced BDD with one 3-dimensional branching followed by three 1-dimensional branchings (3-1-1-1). Finally, we use the size encoding in Section 4.4 to optimally decide the labels of empty terminals towards compactness in a second stage. Note that the size of a multi-dimensional BDD is considered to be its number of branching nodes ($|\mathcal{N}_B|$) after it is unfolded and reduced.

The results of the third set of experiments are presented in Figure 4.8 and show that the two multi-dimensional approaches achieve training accuracy that is higher than a standard BDD with 5 branchings and lower than one with 6 branchings. Since the sum of dimensions equal 6 in both cases, the upper bound is expected according to Proposition 5. However, the comparison against the BDD with 5 branchings shows that the multi-dimensional approaches perform better than their theoretical lower bound (respectively, a standard BDD with 3 and 4 branchings) in practice. The same comparison is observed in testing accuracy and size. However, the approach with a dimension sequence of $3-1-1-1$ is able to achieve smaller variability in testing accuracy while still being close to the best approach in size. Finally, we see a lower number of clauses and a significantly shorter average clause length for our multi-dimensional approaches against standard BDDs of similar expressiveness. Table 4.5 presents the detailed results of this experiment with a larger set of dimension sequences considered.

Next, we run experiments for a significantly larger dataset, namely the Adult dataset ($|X| = 32561$, $|F| = 105$, $k = 2$), to investigate the difference in encoding size and performance between standard and multi-dimensional BDDs on large datasets. In this experiment, we avoid 5-fold cross-validation as we focus on the optimization performance of the two methods. Given the scale of the tasks, we increase the timeout limit to 60 minutes. In Figure 4.9, we compare standard (1-dimensional) BDDs against ones employing 2, 3,

Dataset	d	Size					Training Accuracy (%)					Testing Accuracy (%)				
	ξ	1	3	10	∞^1	∞^2	1	3	10	∞^1	∞^2	1	3	10	∞^1	∞^2
Banknote	4	1	3.6	4	4	4.4	85.3	95.8	96.8	96.8	96.8	85.2	95	96.6	96.6	96.4
	5	3	4	5	11	6	94	96.8	97.6	97.5	97.6	93.7	96.4	96.9	96.8	96.7
	6	4	6	5.8	19.8	6.4	96.8	98.6	98.4	98.6	98.6	96.5	97.4	97.4	97.2	97.2
Breast	4	1	4	5.2	6.4	6.8	72.4	88.8	90.7	90.7	90.7	71.5	75.9	75.9	78.4	77.6
	5	3.8	5.4	11.4	11.8	11	88.8	91.8	94	94	94	74.1	70.7	74.1	72.4	72.5
	6	5.6	9.4	22.2	22.4	16	92.9	96.1	97	97.2	97.2	75	71.6	63	71.6	65.5
Cryo.	4	1	3	4.8	4.8	6	86.1	94.7	97.8	97.8	97.8	75.6	88.9	92.2	92.2	94.4
	5	2.4	5.2	5.8	5.8	8	93.1	99.2	99.4	99.4	99.4	83.3	94.4	91.1	91.1	90
	6	4.6	5.8	5.8	5.8	10.4	98.3	100	100	100	100	95.6	87.8	90	90	82.2
Immuno.	4	1	2.2	5.4	5.4	6.8	87.2	91.4	95.6	95.6	95.6	82.2	83.3	83.3	83.3	85.6
	5	1.6	4.4	5.4	5.4	8.4	89.7	95.3	96.7	96.7	96.7	81.1	81.1	78.9	81.1	73.3
	6	3.8	7.2	10	10	14	94.7	98.3	99.2	99.2	99.2	80	76.7	73.3	72.2	75.6
Ionos.	4	1.8	2.4	4	5.6	4	90	92	95.2	94.9	95.2	87.2	90.9	90.9	90.3	92.3
	5	2	3.6	5.2	10.2	8.4	91.2	94.4	95.9	95.7	95.9	89.7	91.7	89.5	85.8	89.2
	6	2.4	6	7	17.2	12.8	92	96.3	96.9	97.2	97.1	90.9	88.6	88	86.9	86.3
Iris	4	2	2.4	3.8	3.8	4.2	96.3	97.3	98.8	98.8	98.8	94	94	94.7	94.7	94
	5	2	2.8	5.2	5.2	6	96.3	98	99.5	99.5	99.5	94	95.3	96.7	96	96
	6	2.8	5.4	5.8	5.8	7	98	99.8	100	100	100	95.3	95.3	94.7	96	94.7
User	4	2.2	4.4	6	6	6	82.8	91.5	94.3	94.3	94.3	79.8	87.6	93.4	93.4	93.4
	5	4	6	6.2	9.2	7	90.5	95.9	96	96	96	86	93	91.9	93.8	92.2
	6	6	7	8	31.6	9.2	96.2	97.2	97.8	97.8	97.8	93	93	95.7	95.7	95.4
Vertebral	4	1.2	2.4	4	5.4	4.6	81.8	86.8	89.8	89.8	89.8	76.8	76.5	79	78.1	77.7
	5	2.2	4.2	5.2	10.6	8.6	86.1	89.4	89.8	90.8	91	74.8	81.6	81.9	79.7	76.1
	6	3.4	6.2	10.2	20.6	15	88.5	91.6	92.4	92.8	92.7	80.3	80.3	79	76.8	78.7
Wine	4	2.2	3	4	5	6.4	93.8	98.5	99.6	99.9	99.9	87.7	96.6	93.3	93.3	93.9
	5	3	3.6	4.6	4.6	9.4	98.5	99.3	100	100	100	95	95	96.7	95.5	92.7
	6	3	4.6	4.6	4.6	10	98.5	100	100	100	100	95	93.3	92.2	93.3	93.3
Car	4	2	4	4	4	5.4	85.5	92.5	92.5	92.5	92.5	85.5	92.5	92.5	92.5	92.5
	5	3.2	4	4.6	6.8	7.4	90	92.5	92.9	93.1	93.1	88.6	92.5	91.7	92.1	92.1
	6	4	6.6	6.8	12.2	9.6	89	95	95	95.4	95.4	87.6	93.8	93.9	94.7	94.7
Monk2	4	0	3.4	5.4	5.4	6.6	62.1	71.7	75.4	75.4	75.4	62.1	63.3	64.5	65.6	63.9
	5	0.6	7.8	9	9	13	64.6	84.3	84.9	84.9	84.9	59.7	79.3	76.4	77.6	74.6
	6	7.6	11.8	11.8	11.8	13.4	86.2	100	100	100	100	81.7	98.2	98.8	98.2	99.4

Table 4.4: Average 5-fold results for learning BDDs with the combined objective of accuracy and compactness.

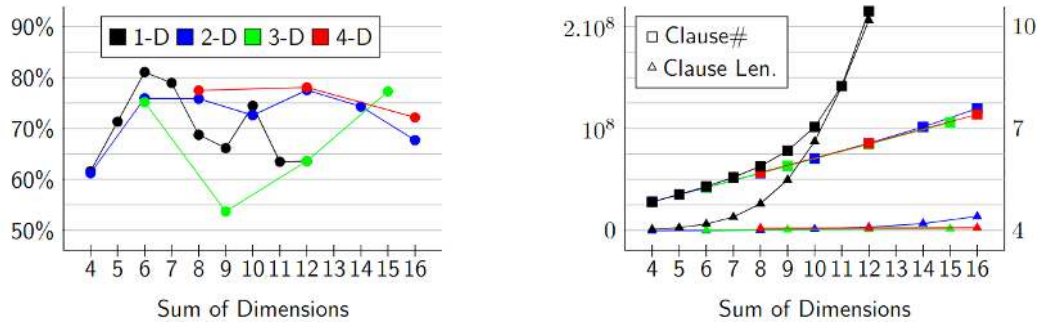


(a) Size. (b) Training Accuracy. (c) Testing Accuracy.

(d) Number of clauses. (e) Average Clause Length.

Figure 4.8: Distributional result for learning BDDs with two number of branchings and Multi-dimensional BDDs with two dimension sequences.

and 4-dimensional branchings that have the same total number of dimensions. The number and length of clauses reported in Figure 4.9 depict the exponential growth of encoding size for the standard approach. The training accuracy presented in Figure 4.9 shows that the exponential size causes the standard approach to decrease in quality and finally cease to produce any solutions due to memory overflow. We observe that by using two-dimensional branchings, we maintain a higher accuracy over a significantly larger portion of dimension sums. Three-dimensional and four-dimensional branchings prove to be more challenging compared to two-dimensional branchings. However, we are still able to find solutions for higher total number of dimensions compared to the 1-dimensional branchings. Note that the sum of dimensions is equal to the total number of features that a multi-dimensional BDD employs. Thus, producing solutions for higher dimension sums equates to being able to exploit more features of the data in solving classification.



(a) Training accuracy.

(b) Number and average length of clauses.

Figure 4.9: Results for learning BDDs and multi-dimensional BDDs for the Adult dataset.

4.8 Related Work

Decision tree classifiers are traditionally constructed via local search and heuristics [50, 230, 229]. Recent advances in tools and techniques for solving combinatorial optimization problems have enabled approaches for finding optimal decision trees via branch-and-bound search [6], SAT-based encodings [126, 19], Constraint Programming [280], or Mixed Integer Programming [37]. Combinatorial approaches produces solutions that are optimal in accuracy, size, or both. An example of learning optimal decision trees in both objectives is our SAT-based approach that was presented in Chapter 3. The size of a decision tree is often measured by its depth or number of nodes, which does not take into account the amount of redundancy between nodes. Decision diagrams address redundancy directly by merging equivalent nodes. Approaches to learn optimal decision diagrams via combinatorial optimization problems usually require a pre-determined skeleton as input.

The work in Florio et al. [84] requires the depth of the diagram and the width for each level to be provided by user. They will then formulate the problem in mixed integer programming (MIP) to learn branchings for each node, connections from each level to the next, and direct connections from each level to terminals. While requiring a skeleton as an input parameter can be limiting, the authors argue that only providing the width of each level is expressive enough and supports a variety of structures. Semi-determining the skeleton prior to learning has the benefits of improved performance and strict control on compactness, but requires involvement of experts and/or heuristics and can lead to sub-optimal solutions. The approach in Florio et al. further supports fairness properties (e.g., demographic parity), size, and stability (i.e., minimum support) to be optimized or regularized as objectives or bounded as constraints, further highlighting the benefits of using declarative combinatorial optimization problems to learn inherently interpretable models. Existing MIP tools also support techniques including warm-starts to improve the solving performance. Moreover, due to the numerical nature of MIP, the authors are able to naturally extend their support to multi-variate branchings. However, their categorical branchings are limited to one-hot encoding which we showed to be less expressive than power set branching in Section 3.8.2.

Contrarily, our SAT-based encoding for learning BDDs can easily be extended to support power set branching as it uses the basis of the encoding presented in Section 3.3. We opt against using Florio et al. as a baseline since it does not enforce the oblivious property and allows different branchings across a decision level. As a result, Florio et al. finds solutions that are more expressive but can be considered less interpretable.

Oblivious decision diagrams encourage compactness even further by requiring all branchings of the same level to be the same, leading to only a linear number of branchings instead of exponential as in ordinary decision trees. Binary Decision Diagrams (BDD) are oblivious decision diagrams that can be reduced and ordered, making them ideal candidates to represent boolean functions. While BDDs are commonly used for hardware synthesis [52, 9, 202, 159], there has been a recent focus on utilizing BDD classifiers as interpretable solutions [124, 53].

Similar to some variants of our approach, the algorithm in Hu, Huguet, and Siala [124] uses a SAT-based encoding to learn BDD classifiers as oblivious ordered decision trees which will then be reduced to diagrams by merging isomorphic subgraphs and removing redundant branchings. They propose to solve the perfect classification problem by solving SAT and the maximum accuracy problem by solving MaxSAT variations of their encoding, the same two objectives that we optimize in our decision tree classification work from Chapter 3. Like many state-of-the-art combinatorial approaches to learn decision trees and diagrams, their approach suffers from a lack of direct support for non-binary features. They provide two ways of encoding branchings, with the latter improving the former in size by explicitly modeling the values of each point for each chosen feature in the diagram. The contrast in performance emphasizes that the same problem can be formulated into a declarative combinatorial optimization problem to different levels of success and it is worthwhile to elegantly design and refine models. Since reduction is done in a secondary stage, the labels of terminals can impact the final size of the BDD, even if they are empty and do not impact the training accuracy. Three biases are introduced to determine the labels of empty terminals after learning a non-reduced tree, but no significant effect is observed with regard to testing accuracy. We included Hu, Huguet, and Siala [124] as our main baseline, to highlight the improvement in performance that a direct encoding of numerical features provides. We further emphasized that the heuristic biases for deciding labels will likely lead to unnecessarily large solutions, and instead proposed multiple approaches to exactly optimize compactness.

Cabodi et al. [53] proposes another SAT encoding for learning BDD classifiers, purely focused on learning perfect classifiers of minimum size that are limited to operate on binary features and assign binary labels. Unlike our approach and those proposed in Florio et al. and Hu, Huguet, and Siala, the encoding in Cabodi et al. is completely flexible to directly model and learn any structure imaginable for a decision diagram. A descriptive paradigm like SAT allows the encoding of parent and child relationships of nodes, in addition to branchings, to learn BDDs without any prior limitation on the structure. The size of the BDDs is then optimized using the encoding by iteratively considering exponentially larger

sizes and performing a binary search once a feasible range of size is detected. However, the flexibility in modeling any diagram comes with poor scalability, even when only considering perfect classifiers, which is a generally easier problem than maximizing accuracy. This leads the authors to also consider a secondary heuristic approach based on techniques from the BDD literature, like the cofactor, that scales better but can yield solutions sub-optimal in size. Given that the problem considered in Cabodi et al. is more general and inherently different than ours, we will not be making a direct comparison against Cabodi et al. as a baseline.

4.9 Conclusion and Future Work

In this chapter, we present a novel MaxSAT encoding for learning Binary Decision Diagrams. Our approach reuses the direct encoding of non-binary features from our SAT-based decision tree encoding (Chapter 3), demonstrating that the formulation approach provides the means to mix and match components that model behaviors, objectives, or constraints. Our BDD encoding learns an oblivious tree which can be reduced to an equivalent diagram. We extend our encoding with optimization for the size of the reduced diagram. The formulation approach provides a natural support for the size objective to be balanced against accuracy in a 1-stage approach or optimized as a second stage. The size can also be bounded in order to produce solutions in a constrained manner. Note that a hard-coded search approach or expertly designed heuristic would have required considerable augmentation to accommodate size optimization in addition to accuracy. Formulation into SAT also allows us to nest our encoding, so that each branching is itself represented by a BDD. This nesting results in a variant of our encoding that uses multi-dimensional branchings represented by inner BDDs. Lastly, note that even though we opt against doing so in this chapter, we can naturally extend our BDD encoding with many features from our decision tree encoding such as power set branching, cost-sensitive learning, and pruning constraints.

Our experiments show that we outperform the state-of-the-art SAT-based BDD learning baseline due to our direct encoding of numerical branchings. We further show that our 1-stage and 2-stage size optimization approaches lead to significantly more compact solutions while maintaining testing accuracy. Following the definition in Murdoch et al. [206], the improved sparsity and, as a result, simulatability contribute to better interpretability. Smaller solutions are also better suited to be used as formal components, especially when the operations performed on the component are computationally expensive. Finally, we show that the expressiveness of our multi-dimensional BDDs allows us to produce high quality solutions in smaller number of branchings, mitigating the exponential growth in the size of the encoding. Employing multi-dimensional branchings can challenge the modularity of an interpretable model as each node is now a more complex component. However, one can argue that multi-dimensional BDDs are still interpretable as each node is still represented by an interpretable model, especially compared to a standard BDD that employs the same number of features but in a less structured manner. From another perspective, multi-dimensional branchings

can be viewed as model-based feature engineering [206], where informative interpretable features are learned to improve model accuracy.

Our work can be extended in a number of ways. Our encoding to optimize the size in a second stage can be extended to model changing the order of branchings in addition to the labels of the empty terminals. Note that a different order will result in a different sequence of empty terminals, affecting the equivalencies and the reduction procedure. Branching ordering (or equivalently feature ordering, in the case of binary branchings) has been shown to significantly impact the size of a BDD [52] and is directly optimized when searching for the most compact form of a given BDD [141, 76], although it is mostly found using heuristics due to its complexity [258, 145]. Note that no extension is required when the size objective is optimized simultaneously with the accuracy, as the branchings are yet to be determined at that stage and the ordering is being optimized to reduce size.

Another interesting direction for future work is to employ multi-dimensional branching in directional BDDs, effectively further nesting the concept of representing branchings with BDDs of their own. Hypothetically, this can allow us to utilize an even larger number of features within a compact size and avoid exponentiation even more. However, the increased cost in computation might reach the point of diminishing returns in expressiveness while undermining interpretability. Our approach can also be equipped with a more thorough strategy for balancing the two objectives of accuracy and size. We can use our encoding to produce all of the Pareto optimal solutions, providing the user with a variety of options to choose from based on expert knowledge or a heuristic. Experimentally, our work can be extended with a more comprehensive analysis of the effects of different parameters. Most notably, a wider array of dimension sequences such as balanced and unbalanced (which were studied in this chapter), random, segmented, and instance-specific can be investigated in terms of their accuracy and performance. Lastly, ideas from our work including the size objective and multi-dimensional branchings can be applied to more expressive BDDs such as free BDDs, which are not constrained to be ordered [243].

Dataset	Dimension Sequence	Accuracy (%) Training	Testing	Size Stage 1	Stage 2	Time (s)
Banknote	2-2-2	**98.6**	97.5	7.4	**6.6**	TO
	3-3	98.4	97.5	**6.6**	**6.6**	TO
	1-2-3	**98.6**	96.9	8.2	6.8	TO
	3-1-1-1	**98.6**	**97.7**	8.8	6.8	TO
Breast Cancer	2-2-2	**94.6**	**74.1**	**8.6**	**8.6**	TO
	3-3	92.9	64.6	8.8	8.8	TO
	1-2-3	94.2	68.1	9.6	9.6	TO
	3-1-1-1	94.2	72.4	11	9.8	TO
Cryotherapy	2-2-2	99.7	86.7	10.6	10.2	**6.18**
	3-3	99.4	**92.2**	10.6	10	25.29
	1-2-3	99.7	90	10.6	9.8	13.66
	3-1-1-1	**100**	86.7	**10**	**9.4**	6.45
Immunotherapy	2-2-2	97.2	75.6	**9.2**	**8.6**	515.83
	3-3	96.9	76.7	10.2	10.2	750.07
	1-2-3	97.2	76.7	11.6	11.2	767.69
	3-1-1-1	**98.1**	**84.4**	9.8	9.6	**489.51**
Ionosphere	2-2-2	96.9	90	**6.6**	**6.6**	TO
	3-3	95.4	88.3	8.8	8.8	TO
	1-2-3	**97**	90.3	7.4	7.4	TO
	3-1-1-1	96.7	**91.5**	8.2	8.2	TO
Iris	2-2-2	99.5	**95.3**	**10.4**	8.4	1.17
	3-3	99.5	94.7	11.2	11.2	0.67
	1-2-3	99.8	94	14.6	**8**	1.2
	3-1-1-1	**100**	93.3	12.4	8.8	**0.44**
User Knowledge	2-2-2	95.6	91.1	**12**	11.6	TO
	3-3	93	88.7	12.2	12.2	**143.96**
	1-2-3	**97.7**	**96.1**	18.2	18.2	494.02
	3-1-1-1	**97.7**	95.4	13.6	**9.8**	783.39
Vertebral Column	2-2-2	91.3	79.4	7.6	**7.2**	TO
	3-3	**91.8**	**80**	**7.4**	7.4	TO
	1-2-3	91.1	79.4	9.4	8.2	TO
	3-1-1-1	90.9	77.4	10.6	8.6	TO
Wine	2-2-2	**100**	93.9	**9**	8.8	3.43
	3-3	**100**	90.5	9.6	9.6	**1.03**
	1-2-3	**100**	93.3	14.2	11.6	5.29
	3-1-1-1	**100**	**95**	**9**	**7.6**	9.03
Car	2-2-2	94	92.5	6.4	6.4	TO
	3-3	93	92.3	6.4	6.4	TO
	1-2-3	92.8	92.6	**6.2**	**5.8**	TO
	3-1-1-1	**94.1**	**93.2**	6.8	6.8	TO
Monk2	2-2-2	87.9	79.3	12.2	12.2	TO
	3-3	89.6	80.5	**10.6**	**10.6**	TO
	1-2-3	90.1	74.6	13	13	TO
	3-1-1-1	**92.9**	**85.8**	11.8	11.8	**695.58**

Table 4.5: Results for learning max-accuracy multi-dimensional BDDs.

Chapter 5

Constrained Clustering via Decision Trees

5.1 Introduction

In this chapter, we shift our attention to learning inherently interpretable models for clustering. We formulate and learn decision trees as clustering solutions, just like we did for the classification problem in Chapter 3. Clustering is the problem of learning existing patterns in data by grouping points into clusters. Decision trees are easy to be understood by humans and reason about by computers. While historically mostly utilized for solving classification, recent approaches to clustering via decision trees have yielded a degree of interpretability through transparency by exposing clustering rationale in the branching structure of the tree [88, 203, 38, 91]. Decision trees are interpretable in the context of clustering for the same reason that they are interpretable classifiers, i.e., they are sparse, simulatable, modular, and make informative decisions when the features are interpretable [206].

Most clustering objectives encourage meaningful groups by using a given distance measure. Ideally, members within a cluster should be semantically close to each other while members of differing clusters should differ. Clustering is a fundamental problem in ML and has wide ranging applications in areas including but not limited to medical sciences [266] and market analysis [295]. Most clustering algorithms either allow an arbitrary assignment of points to clusters or learn black-box functions akin to those employed for classification, e.g., deep neural models, providing no structure or explanation on how the data is divided [298]. Approaches lacking transparency introduce issues for applications in which the satisfaction of a specification is required, clarity in decision making is expected, or an expert needs to be involved in the learning process.

The approaches to learning decision trees are generally divided into local search with heuristics [50, 230, 229] and combinatorial optimization [19, 39, 209, 281, 280, 6, 126, 37]. Local search approaches scale well in terms of performance but suffer from the lack of op-

timality guarantees that combinatorial optimization approaches provides. As discussed in Chapter 2, a sub-category of combinatorial optimization approaches formulate the problem into a declarative standardized problem such as MaxSAT, Integer Programming (IP), and Constraint Programming (CP) and use off-the-shelf solvers to solve the encoded instances. The declarative nature of the formulation approach supports any constraint or specification that can be modelled in the target language [4], enabling constrained ML. It further allows the modularity to interchange and reuse components, as we did by repurposing the non-binary branching encoding from Chapter 3 into the model presented in Chapter 4. Similar to the problem of classification as discussed in Chapter 3 and Chapter 4, it is not always necessary to find optimal decision tree solutions considering the difficulty of doing so. Moreover, optimality can be seen as less important in clustering since objectives typically differ from the goal criterion, unlike for classification where both are primarily predictive accuracy. However, optimality can be demanded in certain settings, particularly when working on small and highly precise scientific datasets [104]. It is also shown to improve evaluation criteria [64] compared to the heuristic approaches [225, 42]. In the context of interpretable clustering solutions, there does not exist a definitive observation on the impact of optimality on solutions, as none of the existing exact approaches are efficient enough to be experimentally evaluated.

The ability to support constraints provided by the formulation approach is particularly of high interest when tackling the problem of constrained clustering. Constrained clustering is a semi-supervised variant of clustering in which bits of domain-specific knowledge is fed into the learning algorithm as constraints to satisfy [42, 225, 184, 64]. The constraints can be concerned with instance-level properties as well as high-level characteristics of clusters including their geometry and density. Instance-level constraints, including pairwise links, are surprisingly expressive given that a number of higher level constraints can be translated to them [70, 183]. Employing clustering constraints has also been shown to improve accuracy in addition to the control that it provides [283, 284].

Approaches that are based on combinatorial optimization problems have been successful in tackling the non-tree variation of the constrained clustering problem [71, 35, 66, 21]. It has also been shown that non-exact approaches can be used to solve decision tree clustering [88, 203, 91]. However, they can lead to suboptimal solutions and lack support for constraints. Combinatorially optimizing decision trees has proven to be an elusive goal in solving clustering or its constrained counterpart. Search-based (e.g., [7]) and formulation-based (e.g., [38]) methods have been proposed but shown to scale extremely poorly compared to analogous approaches for decision tree classification [139].

We propose the first combinatorial optimization approach to learning decision trees for the clustering and the constrained clustering problems. We present a novel SAT-based model that reuses our direct encoding of non-binary features from Chapter 3, optimizes towards a guaranteed approximation of a well-known bi-criteria clustering objective, and supports pairwise clustering constraints. Our ability to integrate two numerical objectives into the SAT paradigm, which is fundamentally binary, is a testament to the novelty of our approach

and the flexibility granted by using the formulation approach to learn interpretable models. Our direct encoding of non-binary features particularly suits the problem of clustering, which mostly has instances involving numerical and categorical data. We also interconnect the encodings of objective and constraints within a pruning technique to reduce the size of the SAT instance. Furthermore, we study feasibility against quality trade-offs and ways to address infeasibility by exploiting the optimization clauses offered by MaxSAT. Lastly, we provide alternative approaches to more comprehensively optimize the bi-criteria approach, showing that combinatorial optimization problems can also be employed in an iterative manner.

Our contributions throughout this chapter are as follows:

1. We present a MaxSAT encoding with a direct encoding of numerical features for solving the problem of decision tree clustering. Our encoding guarantees an ϵ-approximation of a Pareto optimal solution in maximum diameter (MD) and minimum split (MS) criteria. We add support for pairwise constraints as must-links and cannot-links to enable solving the constrained clustering problem.

2. We integrate the encoding of pairwise links with that of the objectives to detect and prune infeasible and redundant clauses, in a novel method referred to as the smart pairs algorithm.

3. We introduce soft pairwise constraints that are encouraged, rather than required, to be satisfied. Soft must-links and cannot-links help address the issue of infeasibility when no solution is found to satisfy all hard constraints in a given depth. We propose 1-stage, 2-stage, and 3-stage schemes to optimizing soft constraints with or without linearizing link satisfaction.

4. We detail how our encoding and support for constraints can be utilized towards producing approximations of all solutions that are Pareto optimal in $[\mathrm{MD}, \mathrm{MS}]$, rather than an arbitrary one.

5. We perform an extensive set of experiments and show that: 1) Our approach is able to find interpretable solutions of high quality in short runtimes; 2) Employing tree clustering, the $[\mathrm{MD}, \mathrm{MS}]$ objective, or user-provided constraints improves the solutions in evaluation metrics, with a combination of two or all of them leading to an even more significant improvement; 3) There is a trade-off between solution accuracy and infeasibility in the presence of user-provided constraints; 4) Soft constraints enable us to benefit from a higher number of constraints and further improve accuracy; 5) Approximating the Pareto front as a whole enables us to find higher quality solutions in less constrained instances; 6) The objective approximation and smart pairs pruning significantly boost performance leading to higher quality solutions.

The content of this chapter is primarily based on the paper that appeared in the proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence [250].

We have expanded upon the original work with introduction of soft constraints, complete approximation of the solutions in the Pareto front, and a more extensive set of experiments. The extended version of this work is in progress towards a journal submission.

5.2 Preliminaries

In this section, we define the problem of constrained clustering using decision trees. We first define user-provided pairwise constraints and the bi-criteria objective. We then define two variations of the constrained clustering problem, with or without restricting the solution to be a decision tree, respectively. We describe external evaluation metrics that can be used to assess the quality of clustering solutions in Section 5.2.1.

The assignment of labels by a decision tree (Definition 4) can be interpreted as a clustering solution. Namely, if a decision tree $\mathcal{D}$ is intended as a clustering solution, $\Theta_{\mathcal{D}}(x)$ is interpreted as the cluster in which x is placed. Two points that are assigned to the same cluster are said to be clustered *together* or *co-clustered*, while two points that are assigned to different clusters are said to be clustered *separately*. Furthermore, given a dataset $X \subseteq \mathbb{R}^{|F|}$, a decision tree $\mathcal{D}$ is said to have non-empty clusters if $\forall k \in [1..k] : \sum_{x_i \in X} \mathbb{1}(\Theta_{\mathcal{D}}(x_i) = k) \geq 1$. A decision tree is assumed to have non-empty clusters unless noted otherwise. Consistent with recent work [88], each cluster is allowed to spread across multiple leaves to improve expressiveness and the ability to satisfy user-provided constraints.

We define how a clustering solution can satisfy pairwise constraints and introduce the two components of our objective. As our main clustering objective, we consider the well-known bi-criteria objective of minimizing the maximum diameter and maximizing the minimum split [66]. Specifically, we consider bounded approximations parameterized by ϵ such that $\epsilon = 0$ indicates a Pareto optimal solution for $[\mathrm{MD}, \mathrm{MS}]$.

Definition 21 (Must-links and Cannot-links)**.** *Given a dataset $X \subset \mathbb{R}^{|F|}$, must-link and cannot-link pairwise constraints are each a set of pairs of points $ML, CL \subseteq X \times X$. A clustering Θ is said to satisfy ML and CL if it clusters must-links together and cannot-links separately,*

$$\forall (x_1, x_2) \in ML : \Theta(x_1) = \Theta(x_2) \tag{5.1}$$

$$\forall (x_1, x_2) \in CL : \Theta(x_1) \neq \Theta(x_2). \tag{5.2}$$

Definition 22 (Maximum Diameter and Minimum Split)**.** *Given a clustering Θ and a dataset $X \subset \mathbb{R}^{|F|}$, a split is the distance between a pair of points that are clustered separately, and a diameter is the distance between a pair of points that are clustered together. The minimum split MS_Θ and maximum diameter MD_Θ values are then defined accordingly as*

follows:

$$\mathrm{MD}_\Theta = max(\{|x_1 - x_2| \,|\, x_1, x_2 \in X, \Theta(x_1) = \Theta(x_2)\}) \tag{5.3}$$

$$\mathrm{MS}_\Theta = min(\{|x_1 - x_2| \,|\, x_1, x_2 \in X, \Theta(x_1) \neq \Theta(x_2)\}). \tag{5.4}$$

We assume $|x_i - x_j|$ to be the Euclidean distance. However, the objectives can also be defined for any other metric space and its corresponding distance function. Our approach throughout the rest of the paper remains applicable in any such setting.

Next, we define the notions of optimality and order when optimizing a bi-criteria objective.

Definition 23 (Pareto Optimality)**.** *A clustering Θ is said to dominate another clustering Θ' if it achieves a better or equal value in one objective and strictly better value in the other.*

$$(\mathrm{MD}_\Theta < \mathrm{MD}_{\Theta'} \wedge \mathrm{MS}_\Theta \geq \mathrm{MS}_{\Theta'}) \vee (\mathrm{MD}_\Theta \leq \mathrm{MD}_{\Theta'} \wedge \mathrm{MS}_\Theta > \mathrm{MS}_{\Theta'})$$

A clustering that is not dominated by any other clustering is said to be Pareto optimal*. Two solutions are* equivalent *if they have the same objective values. A set of Pareto optimal solutions is considered* complete *if it contains at least one equivalent for every Pareto optimal solution. The set of objective value pairs corresponding to a complete Pareto optimal set is referred to as the* Pareto front*.*

Next, we formally define the approximated decision tree clustering problem which aims to find an approximation of a Pareto optimal solution.

Problem 6 (Decision Tree Clustering)**.** *Given a set of numerical features F, clusters $[1..k]$, dataset $X \subset \mathbb{R}^{|F|}$, must-link and cannot-link constraints $ML, CL \subseteq X \times X$, fixed depth d, and approximation parameter ϵ, find a fully balanced decision tree $\mathcal{D}$ of depth d, such that:*

1. *$\Theta_\mathcal{D}$ satisfies the constraints ML and CL;*
2. *$\mathrm{MD}_\mathcal{D} \leq \mathrm{MD}_{\mathcal{D}^*} + \epsilon$;*
3. *$\mathrm{MS}_\mathcal{D} \geq \mathrm{MS}_{\mathcal{D}^*} - \epsilon$,*

where $\mathrm{MD}_\mathcal{D}$ and $\mathrm{MS}_\mathcal{D}$ are shorthands for the maximum diameter and minimum split of the clustering represented by $\mathcal{D}$ and $\mathcal{D}^*$ is an arbitrary Pareto optimal solution to the problem with respect to the bi-criteria of maximizing MS *and minimizing* MD*.*

Note that Problem 6 is not guaranteed to be feasible due to two potential reasons: (1) the sets of must-links and cannot-links are not consistent (e.g., $ML \cap CL \neq \emptyset$); (2) there exists an arbitrary clustering Θ satisfying the constraints but no such clustering can be represented by a decision tree of given depth. However, by using Theorem 1 and setting $\gamma = \Theta$, we conclude that there always exists a deep enough decision tree replicating any arbitrary clustering.

The variation of Problem 6 with no restriction for the solution to conform to a decision tree, is simply referred to as the problem of constrained clustering (CC).

Problem 7 (Constrained Clustering)**.** *Given a set of numerical features F, clusters $[1..k]$, dataset $X \subset \mathbb{R}^{|F|}$, must-link and cannot-link constraints $ML, CL \subseteq X \times X$, and approximation parameter ϵ, find a clustering Θ such that:*

1. *Θ satisfies the constraints ML and CL;*
2. $\text{MD}_\Theta \leq \text{MD}_{\Theta^*} + \epsilon$;
3. $\text{MS}_\Theta \geq \text{MS}_{\Theta^*} - \epsilon$,

where Θ^* is an arbitrary Pareto optimal solution to the problem with respect to the bi-criteria of maximizing the minimum split and minimizing the maximum diameter.

By removing the minimum split component from the bi-criteria objective, we obtain variations of Problem 6 and Problem 7 that solely focus on the maximum diameter. To define the MD-only variations, we can simply remove the third condition (respectively $\text{MS}_\mathcal{D} \geq \text{MS}_{\mathcal{D}^*} - \epsilon$ and $\text{MS}_\Theta \geq \text{MS}_{\Theta^*} - \epsilon$) from the definitions of Problem 6 and Problem 7.

5.2.1 Clustering Evaluation Metrics

In this section, we define metrics for the external evaluation of clustering solutions based on given ground-truth labels. To determine how closely clustering solutions resemble the ground truth, we use two well-known external clustering evaluation metrics: the Adjusted Rand Index (ARI) [132] and the Normalized Mutual Information (NMI) [261].

Given a ground-truth labeling $\hat{\Theta}$ and a clustering solution Θ, the ARI metric is calculated from the number of pairs that are co-clustered by both ($\sigma^{\hat{\Theta},\Theta}$), by neither ($\sigma$), only by $\hat{\Theta}$ ($\sigma^{\hat{\Theta}}$), and only by Θ (σ^{Θ}).

$$\text{ARI} = \frac{2(\sigma^{\hat{\Theta},\Theta}.\sigma - \sigma^{\Theta}.\sigma^{\hat{\Theta}})}{(\sigma^{\hat{\Theta},\Theta} + \sigma^{\hat{\Theta}})(\sigma^{\hat{\Theta}} + \sigma) + (\sigma^{\hat{\Theta},\Theta} + \sigma^{\Theta})(\sigma^{\Theta} + \sigma)}$$

The NMI metric is the normalized version of the Mutual Information metric. It is calculated using the entropy of the ground-truth labeling ($H(\hat{\Theta})$), the entropy of the clustering ($H(\Theta)$), and the conditional entropy of ground-truth given the clustering ($H(\hat{\Theta}|\Theta)$).

$$\text{NMI} = \frac{2.[H(\hat{\Theta}) - H(\hat{\Theta}|\Theta)]}{[H(\hat{\Theta}) + H(\Theta)]}$$

5.3 Distance Classes

In this section, we introduce *distance classes*, a grouping of pairs of data points based on their distance. These classes help us approximate the maximum diameter and minimum split values, leading to a simpler optimization of the objectives.

To encode the ϵ-approximation of the two objectives in Problem 6, we divide the set of all pairs of data points based on their distance into μ non-overlapping intervals called distance classes $\hat{D} = \{D_1, D_2, ..., D_\mu\}$ such that the smallest and largest distances in each class are less than ϵ apart. Any method of grouping pairs that satisfies the requirements is sound with regard to the approximation. However, we employ the simplest approach of sorting all pairs based on their distance and greedily adding them one by one, creating new classes as needed to guarantee that the distances in each class are at most ϵ apart.

The purpose of distance classes are to ensure that all pairs in the same class are treated similarly with regard to being clustered together or not. We consider the index λ^+ such that any pair of data points in distance classes $D_1..D_{\lambda^+}$ must be clustered together (Eq. (5.5)). Similarly, we consider the index λ^- such that any pair in distance classes $D_{\lambda^-+1}..D_\mu$ must be clustered separately (Eq. (5.6)). The pairs in distance classes in-between the two indices $D_{\lambda^++1}..D_{\lambda^-}$ are free to be clustered together or separately on an individual level.

$$((x_1, x_2) \in D_w, w \leq \lambda^+) \rightarrow \Theta_{\mathcal{D}}(x_1) = \Theta_{\mathcal{D}}(x_2) \tag{5.5}$$

$$((x_1, x_2) \in D_w, w > \lambda^-) \rightarrow \Theta_{\mathcal{D}}(x_1) \neq \Theta_{\mathcal{D}}(x_2) \tag{5.6}$$

5.3.1 ϵ-Pareto Optimality

We now use distance classes to describe how we can approximate the MD and MS values. Note that in any feasible clustering, for all λ^+ and λ^- values that satisfy Eq. (5.5) and Eq. (5.6), we have that MS $\geq min(\{|x_1 - x_2| \,|\, (x_1, x_2) \in D_{\lambda^++1}\})$ and MD $\leq max(\{|x_1 - x_2| \,|\, (x_1, x_2) \in D_{\lambda^-}\})$. We therefore maximize λ^+ as a proxy for maximizing the minimum split and minimize λ^- as a proxy for minimizing the maximum diameter in Problem 6. Proposition 6 shows that optimizing λ^+ and λ^- as proxy objectives will lead to guaranteed approximation of [MD, MS].

Proposition 6 (ϵ-approximation of [MD, MS])**.** *Let $\mathcal{D}$ be a decision tree clustering, if $\mathcal{D}$ is a Pareto optimal solution with regard to minimizing λ^- and maximizing λ^+ then $\mathcal{D}$ is an ϵ-Pareto optimal solution with regard to the bi-criteria* [MD, MS] *objective as defined in Problem 6.*

Note that Pareto optimality in λ^+ and λ^- is defined analogous to Pareto optimality in MS and MD (Definition 23).

Proof. We show that there always exists a solution $\mathcal{D}^*$ that is Pareto optimal in MD and MS and is within the ϵ-neighborhood of $\mathcal{D}$ in the two objectives.

- Let $\text{MS}_{\mathcal{D}}$ ($\text{MD}_{\mathcal{D}}$) be the minimum split (maximum diameter) achieved by the solution $\mathcal{D}$. Moreover, let $\lambda^+_{\mathcal{D}}$ and $\lambda^-_{\mathcal{D}}$ be the optimized values of λ^+ and λ^- corresponding to solution $\mathcal{D}$.

- To prove by contradiction, assume that no such $\mathcal{D}^*$ exists. As a result, there should exist another solution $\mathcal{D}'$ that dominates $\mathcal{D}$ and is not in its ϵ neighborhood.
- Assume, without loss of generality, that $\text{MS}_{\mathcal{D}'} > \text{MS}_{\mathcal{D}} + \epsilon$ and $\text{MD}_{\mathcal{D}'} \leq \text{MD}_{\mathcal{D}}$.
- The pairs corresponding to values $\text{MS}_{\mathcal{D}'}$ and $\text{MS}_{\mathcal{D}}$ cannot be in the same distance class due to being more than ϵ far apart. Thus, $\text{MS}_{\mathcal{D}'} \in D_w$ where $w > \lambda^+_{\mathcal{D}} + 1$ and $\lambda^+_{\mathcal{D}} + 1$ is a valid λ^+ value for $\mathcal{D}'$.
- Since $\text{MD}_{\mathcal{D}'} \leq \text{MD}_{\mathcal{D}}$, $\lambda^-_{\mathcal{D}}$ is also a valid λ^- value for $\mathcal{D}'$.
- Therefore, $\mathcal{D}'$ dominates $\mathcal{D}$ in optimizing λ^+ and λ^- values as well. Leading to contradiction as $\mathcal{D}$ is a Pareto optimal solution with regard to minimizing λ^- and maximizing λ^+.
- We conclude that there exists $\mathcal{D}^*$ that satisfies the conditions described in Problem 6. Therefore, $\mathcal{D}$ is an ϵ-approximation of a Pareto optimal solution in MD and MS.

□

5.4 SAT-based Encoding for ϵ-optimal Decision Tree Clustering

In this section we present our approach to solve Problem 6 by encoding it into a SAT instance. We reuse our encoding basis with direct support for numerical features from our decision tree classification approach and remodel the necessary extra aspects. We assume we are given a number of clusters k and a fully balanced tree structure $\mathcal{T}$ of depth d and learn the values of β, α, and θ functions. In Section 5.4.1, we introduce the variables used to encode different aspects of a tree clustering solution. In Section 5.4.2, we present the hard clauses that guarantee the validity of our encoding and the soft clauses that model our objectives. Lastly in Section 5.5, we introduce the smart pairs pruning algorithm to detect the infeasible and redundant clauses of our encoding.

5.4.1 Variables and Structure

We use the following set of boolean variables in our encoding to represent the β, α, and θ functions of the decision tree, the λ^+ and λ^- values, and how the data points are clustered. Variables $a_{t,j}$, $s_{i,t}$, $z_{i,t}$, and $g_{t,c}$ are reused from Section 3.3.1, while the rest are new. We use a unary encoding to represent the cluster assignment to leaves ($g_{t,c}$) and data points ($x_{i,c}$). Namely, the assigned cluster is determined by the number of variables set to true along a sequence of length $k-1$. A well-formed unary encoding should not include the sub-sequence 01 at any point. We use unary encoding to enable symmetry-breaking between equivalent cluster assignments. As an example, if we take all data points from the first cluster, assign them to the second, and vice-versa, we will end up with an equivalent clustering. Adding

symmetry-breaking to the encoding can help avoid consideration of equivalent solutions, and improve the performance as a result. We also use unary encoding b_w^- and b_w^+ variables to model λ^- and λ^+ variables, respectively. Unary encoding helps us easily determine where each distance class is located in the division based on λ^- and λ^+ values.

- $\{a_{t,j} \mid t \in \mathcal{T}_B, j \in F\}$: Represents whether feature j is chosen at branching node t. It is equivalent to $\beta(t) = j$.
- $\{s_{i,t} \mid x_i \in X, t \in \mathcal{T}_B\}$: Represents whether point x_i is directed towards the left child, if it passes through branching node t. It is equivalent to $x_i[\beta(t)] \leq \alpha(t)$.
- $\{z_{i,t} \mid x_i \in X, t \in \mathcal{T}_L\}$: Represents whether point x_i ends up at leaf node t.
- $\{g_{t,c} \mid t \in \mathcal{T}_L, 1 < c \leq k\}$: Represents whether the cluster assigned to leaf t is or comes after c. It is equivalent to $\theta(t) >= c$.
- $\{x_{i,c} \mid x_i \in X, 1 < c \leq k\}$: Represents whether the cluster assigned to point x_i is or comes after c. It is equivalent to $\Theta_{\mathcal{D}}(x_i) >= c$.
- $\{b_w^- \mid 1 \leq w \leq \mu\}$: Represents (the negation of) whether the pairs in distance class D_w should be clustered separately. It is equivalent to $w \leq \lambda^-$
- $\{b_w^+ \mid 1 \leq w \leq \mu\}$: Represents whether the pairs in class D_w should be clustered together. It is equivalent to $w \leq \lambda^+$.

Note that the number of $a_{t,j}$ variables is in $O(|\mathcal{T}_B||F|)$, $s_{i,t}$ in $O(|\mathcal{T}_B||X|)$, $z_{i,t}$ in $O(|\mathcal{T}_L||X|)$, $g_{t,c}$ in $O(k|\mathcal{T}_L|)$, $x_{i,c}$ in $O(k|X|)$, b_w^- in $O(\mu)$, and b_w^+ in $O(\mu)$, bringing the total number to be in $O(|\mathcal{T}_L||X| + \mu)$ if $|X| \geq k$ and $|X| \geq |F|$. The value of the μ parameter can become as large as $|X|^2$ if ϵ converges to 0.

5.4.2 Clauses

We encode the construction of the decision tree (Eq. (5.7) to Eq. (5.15)) following our approach in Section 3.3 which used decision trees as classifiers. Direct support for numerical features in our approach lends itself well to our purposes, since numerical features are prevalent in clustering problems. For a detailed description of the clauses in Eq. (5.7) to Eq. (5.15), see Section 3.3.2.

The following clauses guarantee that exactly one feature is chosen at each branching node (Eq. (5.7) and Eq.(5.8)), the points are directed to the left or right child of each branching node based on their value of the chosen feature (Eq. (5.9) and Eq. (5.10)), the appearance of points at leaves correctly corresponds to the path that they are directed through within the tree (Eq. (5.11), Eq. (5.12), and Eq. (5.13)), and thresholds are non-trivial (Eq. (5.14) and Eq. (5.15)).[1]

[1] The set $O_j(X)$ contains all consecutive pairs of points when ordered according to feature j and $O_j^=(X)$ is its subset limiting the members to only those that have equal j values. The set $A_l(t)$ ($A_r(t)$) contains all nodes that have t as descendent of their left (right) child.

$$(\neg a_{t,j}, \neg a_{t,j'}) \qquad \forall t \in \mathcal{T}_B, j \neq j' \in F \tag{5.7}$$

$$(\bigvee_{j \in F} a_{t,j}) \qquad \forall t \in \mathcal{T}_B \tag{5.8}$$

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad \forall t \in \mathcal{T}_B, j \in F, (i,i') \in O_j(X) \tag{5.9}$$

$$(\neg a_{t,j}, \neg s_{i,t}, s_{i',t}) \qquad \forall t \in \mathcal{T}_B, j \in F, (i,i') \in O_j^{=}(X) \tag{5.10}$$

$$(\neg z_{i,t}, s_{i,t'}) \qquad \forall t \in \mathcal{T}_L, x_i \in X, t' \in A_l(t) \tag{5.11}$$

$$(\neg z_{i,t}, \neg s_{i,t'}) \qquad \forall t \in \mathcal{T}_L, x_i \in X, t' \in A_r(t) \tag{5.12}$$

$$(z_{i,t}, \bigvee_{t' \in A_l(t)} \neg s_{i,t'}, \bigvee_{t' \in A_r(t)} s_{i,t'}) \qquad \forall t \in \mathcal{T}_L, x_i \in X \tag{5.13}$$

$$(\neg a_{t,j}, s_{\#_j^1,t}) \qquad \forall t \in \mathcal{T}_B, j \in F \tag{5.14}$$

$$(\neg a_{t,j}, \neg s_{\#_j^{|X|},t}) \qquad \forall t \in \mathcal{T}_B, j \in F \tag{5.15}$$

The following clauses extend the basic decision tree encoding to support ϵ-optimal clustering trees that satisfy the ML and CL constraints. The clauses in Eq. (5.16) guarantee well-formed unary encoding of cluster labels in each leaf.

$$(g_{t,c}, \neg g_{t,c+1}) \qquad \forall t \in \mathcal{T}_L, c \in [2..k-1] \tag{5.16}$$

The clauses in Eq. (5.17)) and Eq. (5.18) guarantee that the label assigned to each data point matches the label of the leaf that contains the point.

$$(\neg z_{i,t}, \neg g_{t,c}, x_{i,c}) \qquad \forall t \in \mathcal{T}_L, x_i \in X, c \in [2..k] \tag{5.17}$$

$$(\neg z_{i,t}, g_{t,c}, \neg x_{i,c}) \qquad \forall t \in \mathcal{T}_L, x_i \in X, c \in [2..k] \tag{5.18}$$

The clauses in Eq. (5.19) and Eq. (5.20) break the symmetry between the equivalent cluster assignments. We consider two cluster assignments Θ^1 and Θ^2 equivalent if there exists a relabelling, $\Gamma : K \to K$, such that $\Gamma \circ \Theta^1 = \Theta^2$. To break symmetries, we force data points in the ascending order of their index to be assigned to the first empty cluster, if they need a new one. Specifically, Eq. (5.19) guarantees that the $c-1$-th point can only be assigned to the first $c-1$ clusters and Eq. (5.20) guarantees that if all previous points are assigned to the first $c-2$ clusters, the current point can only be assigned to the first $c-1$ clusters. These clauses enforce that there are no two feasible clusterings that are relabellings of each other. Note that we do not eliminate viable solutions, since any clustering can be

renamed into an equivalent one that respects this property.

$$(\neg x_{c-1,c}) \qquad \forall c \in [2..k] \tag{5.19}$$

$$(\neg x_{i,c}, \bigvee_{i'<i} x_{i',c-1}) \qquad \forall x_i \in X, c \in [3..k], c < i \tag{5.20}$$

Assuming $|X| \geq k'$, the clause in Eq. (5.21), alongside the symmetry-breaking ones (Eq. (5.19) and Eq. (5.20)), guarantees that k' clusters are non-empty. Specifically, it enforces that at least one point is not assigned to the first $k'-1$ clusters. Thus, given the ordered assignment of clusters due to symmetry-breaking, there should be at least k' non-empty clusters. While we support any valid value for k', we focus on the setting where the lower bound for the number of non-empty clusters is equal to the upper bound, i.e., $k' = k$, in our experiments.

$$(\bigvee_{i} x_{i,k'}) \tag{5.21}$$

The clauses in Eq. (5.22), Eq. (5.23), and Eq. (5.24), hereafter referred to as the *unconditional separating clauses*, guarantee that pairs in CL are clustered separately.

$$(x_{i,2}, x_{i',2}) \qquad \forall (i,i') \in CL \tag{5.22}$$

$$(\neg x_{i,k}, \neg x_{i',k}) \qquad \forall (i,i') \in CL \tag{5.23}$$

$$(\neg x_{i,c}, \neg x_{i',c}, x_{i,c+1}, x_{i',c+1}) \qquad \forall (i,i') \in CL, c \in [2..k-1] \tag{5.24}$$

The clauses in Eq. (5.25) and Eq. (5.26), hereafter referred to as the *unconditional co-clustering clauses*, guarantee that pairs in ML are clustered together.

$$(\neg x_{i,c}, x_{i',c}) \qquad \forall (i,i') \in ML, c \in [2..k] \tag{5.25}$$

$$(x_{i,c}, \neg x_{i',c}) \qquad \forall (i,i') \in ML, c \in [2..k] \tag{5.26}$$

The clauses in Eq. (5.27), Eq. (5.28), and Eq. (5.29), hereafter referred to as the *conditional separating clauses*, guarantee that a b_w^- variable being set to false forces the pairs in distance class w to be clustered separately.

$$(b_w^-, x_{i,2}, x_{i',2}) \qquad \forall D_w \in \hat{D}, (i,i') \in D_w \tag{5.27}$$

$$(b_w^-, \neg x_{i,k}, \neg x_{i',k}) \qquad \forall D_w \in \hat{D}, (i,i') \in D_w \tag{5.28}$$

$$(b_w^-, \neg x_{i,c}, \neg x_{i',c}, x_{i,c+1}, x_{i',c+1}) \qquad \forall D_w \in \hat{D}, (i,i') \in D_w, c \in [2..k-1] \tag{5.29}$$

The clauses in Eq. (5.30) and Eq. (5.31), hereafter referred to as the *conditional co-clustering clauses*, guarantee that a b_w^+ variable being set to true forces the pairs in distance

class w to be clustered together.

$$(\neg b_w^+, \neg x_{i,c}, x_{i',c}) \qquad \forall D_w \in \hat{D}, (i,i') \in D_w, c \in [2..k] \tag{5.30}$$

$$(\neg b_w^+, x_{i,c}, \neg x_{i',c}) \qquad \forall D_w \in \hat{D}, (i,i') \in D_w, c \in [2..k] \tag{5.31}$$

Lastly, the clauses in Eq. (5.32), Eq. (5.33), and Eq. (5.34) guarantee that b_w^+ and b_w^- variables represent a valid unary encoding and have values that result in well-defined λ^+ and λ^- indices. Specifically, there can exist $\lambda^+ \leq \lambda^-$ values such that $b_1^+..b_{\lambda^+}^+$ and $b_1^-..b_{\lambda^-}^-$ are set to true and the rest of b_w^+ and b_w^- variables are set to false.

$$(\neg b_w^-, b_{w-1}^-) \qquad \forall D_w \in \hat{D}, w > 1 \tag{5.32}$$

$$(\neg b_w^+, b_{w-1}^+) \qquad \forall D_w \in \hat{D}, w > 1 \tag{5.33}$$

$$(\neg b_w^+, b_w^-) \qquad \forall D_w \in \hat{D} \tag{5.34}$$

The number of clauses responsible for the construction of the decision tree are as described in Chapter 3. The number of remaining clauses introduced in Eq. (5.16) is in $O(k|\mathcal{T}_L|)$, in Eq. (5.17) and Eq. (5.18) is in $O(k|\mathcal{T}_L||X|)$, in Eq. (5.19) and Eq. (5.20) is in $O(k|X|)$, in Eq. (5.21) is in $O(1)$, in Eq. (5.22) to Eq. (5.24) is in $O(k|CL|)$, in Eq. (5.25) and Eq. (5.26) is in $O(k|ML|)$, in Eq. (5.27) to Eq. (5.29) is in $O(k|X|^2)$, in Eq. (5.30) and Eq. (5.31) is in $O(k|X|^2)$, and in Eq. (5.32) to Eq. (5.34) is in $O(\mu)$, respectively. Thus, the total number of added clauses is in $O(k|X|^2)$ if $|X| \geq |\mathcal{T}_L|$.

Decoding

In our encoding, most of the variables corresponding to the inner workings of the decision tree are reused from our classification approach. Thus, when given a satisfying assignment to the SAT instance, we can use the procedure described in Section 3.3.3 to decode a decision tree for our tree clustering encoding as well. Specifically, we can learn the values for β and α functions similar to before. With regard to learning the θ function, we consider each leaf node $t \in \mathcal{T}_L$ and set $\theta(t)$ to 1 plus the number of $g_{t,c}$ variables that are true. Note that the rest of variables that are particularly introduced for learning clustering solutions do not directly impact the decision tree and, thus, are not required in the decoding procedure. For detailed instructions on decoding a decision tree solution, see Section 3.3.3.

Objective

In Section 5.3, we established that in order to find an ϵ-approximation of an $[\text{MD}, \text{MS}]$ Pareto optimal solution, we need to find a solution that is Pareto optimal with regards to maximizing λ^+ and minimizing λ^-. Note that λ^+ and λ^- are the number of b_w^+ and b_w^-

variables set to true, respectively.

$$\lambda^+ = \sum_w \mathbb{1}(b_w^+ = true) \tag{5.35}$$

$$\lambda^- = \sum_w \mathbb{1}(b_w^- = true) \tag{5.36}$$

To obtain a Pareto optimal solution, we can optimize any function that is increasing with regard to λ^- and decreasing with regard to λ^+ as objective. We opt for minimizing the simple combination of $\lambda^- - \lambda^+$ and model it using the soft clauses with unit weights (indicated by $\langle\rangle$) in Eq. (5.37) and Eq. (5.38).

$$\langle 1 \rangle (\neg b_w^-) \qquad D_w \in \hat{D} \tag{5.37}$$

$$\langle 1 \rangle (b_w^+) \qquad D_w \in \hat{D} \tag{5.38}$$

The number of soft clauses in Eq. (5.37) and Eq. (5.38) added for encoding the objective is in $O(\mu)$.

5.5 Smart Pairs

Our naive encoding of unconditional and conditional co-clustering and separating clauses (Eq. (5.22) to Eq. (5.31)) leads to a quadratic number of clauses in the size of the dataset $|X|$. This is significant since all the other parts of the encoding are linear in $|X|$. Due to our objectives, the quadratic size is unavoidable, but there are ways to reduce the number of clauses nonetheless. We can employ pruning techniques that see the set of points as nodes in a graph and exploit connections between pairs.

The pruning framework keeps track of connections while our SAT instance is being constructed. At any point, pairs that are already forced to be clustered together are represented by *positive* edges and pairs that are already forced to be clustered separately by *negative* edges. The positive edges imply a set of connected components of points that will be clustered together while a negative edge between nodes in different components indicates that each of the components are mutually exclusive, i.e., each component will be in a different cluster. We incrementally build the set of positive edges (E^+) and the set of negative edges (E^-), with each new edge being classified as *inner* or *crossing*, based on the previous members.

Definition 24 (Inner and Crossing edges)**.** *Given the sets E^+ and E^-, a new edge is said to be an:*

- *Inner edge, if it connects two nodes within an existing connected component based on E^+.*
- *Crossing edge, if it connects two nodes in two connected components based on E^+ that are mutually exclusive based on E^-.*

We will use edges and the pairs of points that they represent interchangeably onward.

Identifying a new edge as inner or crossing while adding co-clustering or separating clauses can help us detect infeasibility or redundancy. Since E^+ (E^-) represents the set of pairs that are forced to be clustered together (separately), an inner pair is forced to have the same label and a crossing pair to have different labels. Thus, it would be redundant to, and we avoid, adding co-clustering clauses (Eq. (5.25), Eq. (5.26), Eq. (5.30), and Eq. (5.31)) for inner pairs and separating clauses (Eq. (5.22), Eq. (5.23), Eq. (5.24), Eq. (5.27), Eq. (5.28), and Eq. (5.29)) for crossing pairs. Furthermore, it would be infeasible to enforce co-clustering for crossing pairs and separation for inner pairs. Our treatment of the unconditional clauses (Eq. (5.22) to Eq. (5.26)) and of the conditional ones (Eq. (5.27) to Eq. (5.31)) differ in the case of infeasibility detection and in the construction of E^+ and E^- sets.

- **Infeasibility Detection:** If the unconditional separation or co-clustering of a pair is infeasible, the problem is infeasible. However, for conditional clauses, detecting infeasibility only leads to fixing the corresponding b_w^+ or b_w^- variable to make the conditional clause non-enforcing by satisfying it. Note that nullifying a conditional clause implies the nullification of all of the subsequent ones. So we can stop adding clauses from a series of conditional ones as soon as one is detected to be infeasible.

- **Construction of E^+ and E^-:** For the unconditional clauses the E^+ and E^- sets are respectively constructed from ML and CL links. For conditional ones however, we also include the pairs that are implied to be co-clustered (separated), due to the order of distance imposed by the MS (MD) objective. Namely, if a pair of longer distance is being co-clustered for optimizing MS, a pair of shorter distance is implied to be already co-clustered. Analogously, if a pair of shorter distance is being separated for optimizing MD, a pair of longer distance is implied to be already separated. Note that implied pairs for the processing of conditional co-clustering clauses are not valid for the processing of conditional separating clauses, and vice-versa.

We construct the sets E^+ and E^- by 1) incrementally adding the members of ML to E^+, corresponding to unconditional co-clustering clauses; 2) incrementally adding the members of CL to E^-, corresponding to unconditional separating clauses; 3) incrementally adding all pairs of the dataset in the ascending order of containing distance classes to E^+, corresponding to conditional co-clustering clauses; 4) resetting E^+ to its state prior to stage 3 and incrementally adding all pairs of the dataset in the descending order of containing distance classes to E^-, corresponding to conditional separating clauses. Note that we reset E^+ to remove all edges corresponding to condition co-clustering clauses, as they are no longer implied to hold when we shift the focus to conditional separating clauses. We add unconditional clauses first since they do not require a reset for E^+ or E^- and can be beneficial in further stages of the algorithm. Moreover, we add unconditional co-clustering clauses before separating ones because constructing E^- while $E^+ = \emptyset$ does not lead to any infeasible or redundant cases. We can use any arbitrary order to add members of ML (CL)

Algorithm 1 Smart Pairs

$E^+ = E^- = \emptyset$
for $(x_i, x_{i'}) \in ML$ sorted in ascending $<^*$ **do**
  if $(x_i, x_{i'})$ is not *inner* **then**
    $E^+ = E^+ \cup \{(x_i, x_{i'})\}$
    add clauses from Eq. (5.25) and Eq. (5.26) for $(x_i, x_{i'})$
  end if
end for
for $(x_i, x_{i'}) \in CL$ sorted in descending $<^*$ **do**
  if $(x_i, x_{i'})$ is *inner* **then**
    Return Infeasible
  end if
  if $(x_i, x_{i'})$ is not *crossing* **then**
    $E^- = E^- \cup \{(x_i, x_{i'})\}$
    add clauses from Eq. (5.22), Eq. (5.23), and Eq. (5.24) for $(x_i, x_{i'})$
  end if
end for
$\hat{E}^+ = E^+$
for $(x_i, x_{i'}) \in X \times X$ sorted in ascending $<^*$ **do**
  set w s.t. $(x_i, x_{i'}) \in D_w$
  if $(x_i, x_{i'})$ is *crossing* **then**
    add clause $(\neg b_w^+)$
    Break
  end if
  if $(x_i, x_{i'})$ is not *inner* **then**
    $E^+ = E^+ \cup \{(x_i, x_{i'})\}$
    add clauses from Eq. (5.30) and Eq. (5.31) for $(x_i, x_{i'})$ and w
  end if
end for
$E^+ = \hat{E}^+$ {Discard implied pairs for conditional co-clusering}
for $(x_i, x_{i'}) \in X \times X$ sorted in descending $<^*$ **do**
  set w s.t. $(x_i, x_{i'}) \in D_w$
  if $(x_i, x_{i'})$ is *inner* **then**
    add clause (b_w^-)
    Break
  end if
  if $(x_i, x_{i'})$ is not *crossing* **then**
    $E^- = E^- \cup \{(x_i, x_{i'})\}$
    add clauses from Eq. (5.27), Eq. (5.28), and Eq. (5.29) for $(x_i, x_{i'})$ and w
  end if
end for

to E^+ (E^-), but we opt for ascending distance for co-clustering clauses and descending distance for separating clauses. Algorithm 1 provides a detailed description of the inner-workings of the Smart Pairs procedure.

5.6 Soft Pairwise Constraints

Due to noise and error in producing must-links (ML) and cannot-links (CL) or the limitations of fixed-depth (d) decision trees, Problem 6 in its general form is not guaranteed to have a feasible solution. However, there always exists a solution if $ML = CL = \emptyset$ and $2^d \geq k$, as we can simply find a tree with arbitrarily bad objective values that contains all labels. This shows that there is a balance to be found in the amount of constraints to satisfy. We observe through our experiments that infeasibility can appear even when adding a moderate number of links, prematurely impeding solution quality enhancement through adding constraints. One approach to potentially tackle infeasibility is to consider incrementally deeper trees. However, simply increasing depth can cause issues in performance. Furthermore, we experimentally observe the benefits of limiting solutions to shallow trees, and arbitrarily increasing the depth can actively work against such advantages. We instead opt for disregarding a subset of links that cause infeasibility and utilizing the rest. Namely, we optimize the number of links that we can satisfy instead of enforcing the satisfaction of all constraints.

In this section, we introduce *soft constraints* to address the issue of infeasibility caused by adding restrictive clustering constraints. We propose an extension of our encoding that works by considering must-links and cannot-links as soft constraints that are encouraged, rather than required, to be satisfied. Treating must-links and cannot-links as soft constraints makes them objectives to be optimized in addition to our primary [MD, MS] objective. Since the number of satisfied constraints is dissimilar from λ^+ and λ^- in nature, it would be challenging to convincingly combine the two goals and weigh them against one another. Hence, our approach focuses on prioritizing the satisfaction of links over the clustering objective.

We introduce multiple schemes for optimizing the pairwise links and the [MD, MS] objective. In Section 5.6.1, we propose the 1-stage approach that optimizes link satisfaction simultaneously with the objective. In Section 5.6.2, we propose the 2-stage scheme that optimizes ML and CL constraints in a pre-processing stage and uses the satisfied links as hard constraints when optimizing [MD, MS]. In Section 5.6.3, we further divide constraint optimization into two stages corresponding to CL and ML sets, respectively. In Section 5.6.4, we propose linearizing the satisfaction of soft pairwise constraints. In Section 5.6.5, we discuss how soft ML and CL sets can be integrated into the smart pairs algorithm.

5.6.1 1-stage Approach

The 1-stage approach optimizes the number of satisfied pairwise links and the [MD, MS] objective in the same stage. It completely prioritizes link satisfaction by using a sufficiently high enough weight. The encoding for the 1-stage scheme is slightly different than the encoding presented in Section 5.4.

We use $\{e^m_{i,i'} \mid (i,i') \in ML\}$ and $\{e^c_{i,i'} \mid (i,i') \in CL\}$ variables in addition to those employed in the original encoding. They, respectively, represent whether the must-link

and cannot-link constraint corresponding to the pair (i, i') is satisfied. We remove the unconditional separating and co-clustering clauses (Eq. (5.22) to Eq. (5.26)) and add the clauses in Eq. (5.39) to Eq. (5.43), that use variables $e^m_{i,i'}$ and $e^c_{i,i'}$ to enforce a must-link or cannot-link only if its corresponding variable is true. We use the soft clauses in Eq. (5.44) and Eq. (5.45) to maximize the number of $e^m_{i,i'}$ and $e^c_{i,i'}$ variables set to true and, as a result, the number of satisfied links. We use $2|\hat{D}| + 1$ as the weight for both (indicated in $\langle\rangle$) to guarantee that link satisfaction is completely prioritized over optimizing [MD, MS]. Note that this is true as all of the soft clauses responsible for optimizing [MD, MS] have a total weight of $2|\hat{D}|$ (Eq. (5.37) and Eq. (5.38)). We decode the solution as before since the inner workings of the solution are encoded the same. Note that this variation of the encoding allows a solution even it does not satisfy any links. Thus, the instance cannot become infeasible due to clustering constraints, independent of the quantity.

$$(\neg e^c_{i,i'}, x_{i,1}, x_{i',1}) \qquad \forall (i, i') \in CL \tag{5.39}$$

$$(\neg e^c_{i,i'}, \neg x_{i,k-1}, \neg x_{i',k-1}) \qquad \forall (i, i') \in CL \tag{5.40}$$

$$(\neg e^c_{i,i'}, \neg x_{i,c}, \neg x_{i',c}, x_{i,c+1}, x_{i',c+1}) \qquad \forall (i, i') \in CL, c \in [2..k-1] \tag{5.41}$$

$$(\neg e^m_{i,i'}, \neg x_{i,c}, x_{i',c}) \qquad \forall (i, i') \in ML, c \in [2..k] \tag{5.42}$$

$$(\neg e^m_{i,i'}, x_{i,c}, \neg x_{i',c}) \qquad \forall (i, i') \in ML, c \in [2..k] \tag{5.43}$$

$$\langle 2|\hat{D}| + 1\rangle (e^c_{i,i'}) \qquad (i, i') \in CL \tag{5.44}$$

$$\langle 2|\hat{D}| + 1\rangle (e^m_{i,i'}) \qquad (i, i') \in ML \tag{5.45}$$

5.6.2 2-stage Approach

To alleviate the computational cost of optimizing the constraints and the objective simultaneously, we can divide them into two stages. The 2-stage approach dedicates the first stage to optimizing the combined number of satisfied must-links and cannot-links as the pre-processing stage and optimizes the [MD, MS] objective in the second stage. Slightly modified versions of the encoding in Section 5.4 are solved at each stage.

Stage 1. We dispose of all of the components in the encoding regarding the modelling and optimization of [MD, MS] bi-criteria objective. Namely, λ^+ and λ^- indices, b^+_w and b^-_w variables, the conditional co-clustering and separating clauses (Eq. (5.27) to Eq. (5.31)), the clauses enforcing the proxy objectives to be well-defined (Eq. (5.32) to Eq. (5.34)), and the soft clauses (Eq. (5.37) and Eq. (5.38)) are removed. Similar to the 1-stage approach, we introduce variables $\{e^m_{i,i'} \mid (i, i') \in ML\}$ and $\{e^c_{i,i'} \mid (i, i') \in CL\}$, and add clauses Eq. (5.39) to Eq. (5.43). Lastly, we maximize the number of $e^m_{i,i'}$ and $e^c_{i,i'}$ variables set to true by adding the soft clauses with unit weights in Eq. (5.46) and Eq. (5.47).

$$\langle 1 \rangle (e^c_{i,i'}) \qquad (i,i') \in CL \tag{5.46}$$

$$\langle 1 \rangle (e^m_{i,i'}) \qquad (i,i') \in ML \tag{5.47}$$

Stage 2. We modify the ML and CL sets by removing any member with corresponding $e^M_{i,i'}$ or $e^C_{i,i'}$ variable set to false, i.e., that is not satisfied in stage 1. We then solve the encoding in Section 5.4 without any changes using the modified sets, and decode the solution as before. Note that the modified set of pairwise constraints are shown to be satisfied in stage 1. Thus, stage 2 is guaranteed to be feasible by using the same solution.

5.6.3 3-stage Approach

The 1-stage and 2-stage schemes consider optimization of must-links and cannot-links simultaneously and weigh them equally. However, it is not guaranteed that the two types of pairwise constraints are equally informative and contribute the same to the accuracy of the solution. The difference is further exacerbated when there is a lack of balance in the size of ML and CL sets. For example, if ML is significantly larger than CL, soft constraint optimization has a trivial solution that satisfies almost all of must-links and almost none of cannot-links. The trivial solution has one dominating cluster that contains almost all of the input dataset. The trivial solution can be the optimal solution if it is challenging to satisfy a considerable subset of both ML and CL. This issue, which we refer to as *ML-domination*, is an example of the potential shortcomings of optimizing must-links and cannot-links together.

We propose the 3-stage scheme to separate optimizing the satisfaction of soft must-links and cannot-links. The 3-stage approach resembles the 2-stage approach (Section 5.6.2) in solving modified versions of the main encoding (Section 5.4), but has two pre-processing stages instead of one. Specifically, it first maximizes cannot-link satisfaction, then must-link satisfaction, and then optimizes the [MD, MS] objective. Prioritizing cannot-links over must-links by optimizing their satisfaction in a previous stage allows us to adhere to a satisfied subset of cannot-links and avoid the trivial ML-dominated solution as a result. Note that we do not consider the alternative approach of prioritizing must-links over cannot-links, as it is still prone to ML-domination.

Stage 1. We dispose of all objective optimization components and unconditional separating and co-clustering clauses, as recounted in Section 5.6.2. We introduce variables $\{e^c_{i,i'} \mid (i,i') \in CL\}$ to represent whether the cannot-link constraint corresponding to the pair (i,i') is satisfied. We add the clauses in Eq. (5.39) to Eq. (5.41) to ensure that all cannot-links are correctly claimed to be satisfied. Finally, we add the soft clauses with unit weights in Eq. (5.46) to maximize the number of satisfied cannot-links.

Stage 2. We modify the CL set by removing any member with the corresponding $e^c_{i,i'}$ variable set to false. Similar to stage 1, we dispose of all objective optimization components and unconditional co-clustering clauses. However, the unconditional separating clauses are kept as we consider cannot-links to be hard constraints at this stage. We introduce variables $\{e^m_{i,i'} \mid (i,i') \in ML\}$ to represent whether the must-link constraint corresponding to the pair (i,i') is satisfied. We add the clauses in Eq. (5.42) and Eq. (5.43) to ensure that all must-links are correctly claimed to be satisfied. Finally, we add the soft clauses with unit weights in Eq. (5.47) to maximize the number of satisfied must-links.

Stage 3. We modify the ML set by removing any member with corresponding $e^m_{i,i'}$ variable set to false. We then solve the encoding in Section 5.4 without any changes using the modified sets from the last two stages, and decode the solution as before. Note that similar to the final stage in Section 5.6.2, the final stage is guaranteed to have a feasible solution.

5.6.4 Chained Method

In Section 5.6.1, Section 5.6.2, and Section 5.6.3, we encoded the satisfaction of pairwise constraints, represented by $e^m_{i,i'}$ and $e^c_{i,i'}$ variables, to be independent of each other. However, similar to how a must-link and a cannot-link might differ in value, there can be an inconsistency across the members of ML or CL in terms of importance and how much they contribute to solution quality. To support differing values amongst links, we can linearize the soft constraint optimization objective by placing links in a total order based on priority.

We introduce the *chained method* to linearize the satisfaction of must-links and cannot-links. Linearizing the objectives means that a pairwise link can be claimed to be satisfied only if all of the previous links according to the total order are also satisfied. Namely, we sort the members of CL and ML according to arbitrary orders $O(CL)$ and $O(ML)$ and consider thresholds λ^{CL} and λ^{ML} to optimize such that the first λ^{CL} members of CL and the first λ^{ML} members of ML are satisfied. The clauses in Eq. (5.48) and Eq. (5.49), respectively, guarantee that $e^c_{i,i'}$ and $e^m_{i,i'}$ variables are satisfied in order and λ^{CL} and λ^{ML} are well-defined.

$$(e^c_{i_1,i'_1}, \neg e^c_{i_2,i'_2}) \qquad ((i_1,i'_1),(i_2,i'_2)) \in O(CL) \tag{5.48}$$

$$(e^m_{i_1,i'_1}, \neg e^m_{i_2,i_2}) \qquad ((i_1,i'_1),(i_2,i'_2)) \in O(ML) \tag{5.49}$$

Note that using the chained method may also improve the performance of soft constraint optimization. When the satisfaction of each link can be individually optimized, there is an exponential number of satisfying truth assignments to $e^c_{i,i'}$ and $e^m_{i,i'}$ variables which directly impact the objective value. By linearizing the objective, we also linearize the number of satisfying truth assignments to $e^c_{i,i'}$ and $e^m_{i,i'}$. This can hypothetically facilitate optimizing the objective. We discuss in Section 5.6.5 how using the chained method can also improve

the performance by allowing the smart pairs algorithm to be used for soft constraints.

The chained method can be used in 1-stage (Section 5.6.1), 2-stage (Section 5.6.2), and 3-stage (Section 5.6.3) schemes to soft pairwise constraints.

- **The 1-stage approach:** We add the clauses in Eq. (5.48) and Eq. (5.49), effectively setting the objective to $\lambda^{CL} + \lambda^{ML}$.
- **Stage 1 of the 2-stage approach:** We add the clauses in Eq. (5.48) and Eq. (5.49), effectively setting the objective to $\lambda^{CL} + \lambda^{ML}$.
- **Stage 1 of the 3-stage approach:** We add the clauses in Eq. (5.48), effectively setting the objective to λ^{CL}.
- **Stage 2 of the 3-stage approach:** We add the clauses in Eq. (5.49), effectively setting the objective to λ^{ML}.

The order used for cannot-links and must-links dictated by $O(CL)$ and $O(ML)$ can be based on the Euclidean distance of pairs. The method is referred to as *Euclidean chained*, when CL (ML) is sorted based on decreasing (increasing) Euclidean distance of pairs. The intuition behind Euclidean chained method is that cannot-links with longer distances and must-links with shorter distances are more likely to conform to a given clustering and be easier to satisfy.

5.6.5 Smart Pairs Integration

In this section, we discuss how transforming pairwise constraints from hard to soft affects their integration into the smart pairs algorithm (Section 5.5). We used the unconditional clauses corresponding to ML and CL sets (Eq. (5.22) to Eq. (5.26)) to be the basis of sets E^+ and E^- in the smart pairs algorithm. However, soft cannot-links and must-links are no longer unconditionally required to be satisfied. Thus, the inclusion of ML (CL) members into E^+ (E^-) is rendered unsound in any stage that its satisfied subset is yet to be determined. Specifically, ML and CL both cannot be integrated in the 1-stage approach, stage 1 of the 2-stage approach, and stage 1 of the 3-stage approach, and ML cannot be integrated in stage 2 of the 3-stage approach.

In case of using the chained method (Section 5.6.4), the satisfaction of a cannot-link (must-link) implies the satisfaction of all of the previous members according to the order $O(CL)$ ($O(ML)$). Thus, the chained method allows us to integrate the members of CL and ML into smart pairs, analogous to how we integrate the conditional co-clustering and separating clauses. Specifically, when optimizing the number of satisfied cannot-links (must-links) with the chained method, we can store the state of E^- (E^+), incrementally add the members of CL (ML) to E^- (E^+) in the order of $O(CL)$ ($O(ML)$), and reset the state of E^- (E^+) afterwards.

5.7 Decision Tree Clustering Pareto Front

In Section 5.4, we proposed an encoding for decision tree clustering whose solution is proven to be an ϵ-approximation of a Pareto optimal solution in [MD, MS]. However, there can exist more than one Pareto optimal solution. Our approximated solution is chosen arbitrarily and is not guaranteed to hold any particular advantage over the other Pareto optimal solutions.

Contrarily, exploring the Pareto front as a whole is beneficial as it allows finding higher quality alternatives, producing sets of diverse choices for the solution, and domain-level expertise to be integrated into solution selection. In this section, we propose an algorithm to solve a generalization of Problem 6 that aims to find approximations of all Pareto optimal solutions in the [MD, MS] objective, rather than an arbitrary one.

Problem 8 (Complete Pareto Optimal Decision Tree Clustering)**.** *Given a set of numerical features F, clusters $[1..k]$, dataset $X \subset \mathbb{R}^{|F|}$, must-link and cannot-link constraints $ML, CL \subseteq X \times X$, fixed depth d, and approximation parameter ϵ, find a set of fully balanced decision trees $\{\mathcal{D}^1, \mathcal{D}^2, ..., \mathcal{D}^P\}$ of depth d, such that:*

1. *For all $p \in [1..P]$, $\Theta^p_{\mathcal{D}}$ satisfies the constraints ML and CL;*
2. *For all $\mathcal{D}^* \in \mathbb{D}^*$, $\exists p : (\text{MD}_{\mathcal{D}^p} \leq \text{MD}_{\mathcal{D}^*} + \epsilon) \wedge (\text{MS}_{\mathcal{D}^p} \geq \text{MS}_{\mathcal{D}^*} - \epsilon)$,*

where $\mathbb{D}^*$ is the set of all Pareto optimal solutions to the problem with respect to the bi-criteria of maximizing the minimum split and minimizing the maximum diameter.

The encoding presented in Section 5.4 has proxy objectives λ^+ and λ^- that approximate MS and MD. We highlighted in Section 5.4.2 that we combine the two proxies by minimizing $\lambda^- - \lambda^+$ and thus, aim for an arbitrary Pareto optimal solution in λ^+ and λ^-. In order to solve Problem 8 however, we solve a series of instances to find a complete Pareto optimal set and the Pareto front in λ^+ and λ^-. In Proposition 7, we show that the Pareto front in λ^+ and λ^- includes approximations of any Pareto optimal solution [MD, MS].

Proposition 7 (Complete ϵ-approximation of [MD, MS])**.** *Let $\{\mathcal{D}^1, \mathcal{D}^2, ..., \mathcal{D}^P\}$ be a complete Pareto optimal set of decision tree clusterings in λ^+ and λ^- objectives. For every $\mathcal{D}^*$ that is a Pareto optimal tree clustering in* [MD, MS]*, there exists p with* $(\text{MD}_{\mathcal{D}^p} \leq \text{MD}_{\mathcal{D}^*} + \epsilon) \wedge (\text{MS}_{\mathcal{D}^p} \geq \text{MS}_{\mathcal{D}^*} - \epsilon)$.

Proof. We show that for any arbitrary Pareto optimal solution in minimum split and maximum diameter $\mathcal{D}^*$, there exists a solution $\mathcal{D}^p$ in its ϵ-neighborhood.

- Let $\text{MS}_{\mathcal{D}^*}$ ($\text{MD}_{\mathcal{D}^*}$) be the minimum split (maximum diameter) achieved by the solution $\mathcal{D}^*$. Furthermore, let the pair (x^s_1, x^s_2) ((x^d_1, x^d_2)) correspond to the minimum split (maximum diameter) distance.
- Set $\lambda^+_{\mathcal{D}^*}$ such that $(x^s_1, x^s_2) \in D_{\lambda^+_{\mathcal{D}^*}+1}$ and $\lambda^-_{\mathcal{D}^*}$ such that $(x^d_1, x^d_2) \in D_{\lambda^-_{\mathcal{D}^*}}$.
- Given that the solution $\mathcal{D}^*$ achieves $\lambda^+_{\mathcal{D}^*}$ and $\lambda^-_{\mathcal{D}^*}$ values in λ^+ and λ^-, any complete Pareto optimal set in λ^+ and λ^- should contain a solution with objective values equal or better than $\lambda^+_{\mathcal{D}^*}$ and $\lambda^-_{\mathcal{D}^*}$. Let $\mathcal{D}^p$ be that solution.

Algorithm 2 Complete Pareto Optimal Set

```
F := ∅
D := Minimize(λ⁻)
while true do
    D := Maximize(λ⁺, with λ⁻ ≤ D_{λ⁻})
    F := F ∪ {D}
    D := Minimize(λ⁻, with λ⁺ > D_{λ⁺})
    if D is null then
        Break
    end if
end while
Return F
```

- Since $\lambda^+_{\mathcal{D}^p} \leq \lambda^+_{\mathcal{D}^*} \wedge \lambda^-_{\mathcal{D}^p} \geq \lambda^-_{\mathcal{D}^*}$, we conclude that $(\text{MD}_{\mathcal{D}^p} \leq \text{MD}_{\mathcal{D}^*} + \epsilon) \wedge (\text{MS}_{\mathcal{D}^p} \geq \text{MS}_{\mathcal{D}^*} - \epsilon)$, proving the proposition for an arbitrary Pareto optimal solution.

□

To find a complete Pareto optimal set in λ^+ and λ^-, we use a well-known method for generating complete Pareto optimal sets in bi-criteria problems [66, 267]. We present the details of using this method for our setting in Algorithm 2. At each step, we need to optimize one objective while the other is being (strictly) bounded by a threshold at each step. We propose a modification to the encoding presented in Section 5.4 that supports having a lower (upper) bound on λ^+ (λ^-) and optimizing λ^- (λ^+). In order to focus on a single object, we dispose of all of the components that are purely responsible for the optimization of the other one, and in order to enforce a bound on an objective, we add all pairs that are implied by the bound to be co-clustered (separated) as part of the ML (CL) set. The details of each case is as follows:

- **When we aim to minimize λ^- with $\lambda^+ \geq \lambda^{lb}$:** We dispose of b^+_w variables, the clauses in Eq. (5.30), Eq. (5.31), Eq. (5.33), and Eq. (5.34), and the soft clauses in Eq. (5.38). Next, we add the pairs belonging to the first λ^{lb} distance classes to ML.
- **When we aim to maximize λ^+ with $\lambda^- \leq \lambda^{ub}$:** We dispose of b^-_w variables, the clauses in Eq. (5.27) to Eq. (5.29), Eq. (5.32) to Eq. (5.34), and the soft clauses in Eq. (5.37). Next, we add the pairs belonging to all but the first λ^{ub} distance classes to CL.

Since our objectives are both integers, we can simply implement strict inequality by increasing λ^{lb} or decreasing λ^{ub} by 1.

Note that Algorithm 2 is compatible with hard pairwise constraints or soft ones that have been previously optimized in 2-stage (Section 5.6.2) or 3-stage (Section 5.6.3) schemes. We only focus on the combination with hard constraints for our experimental studies due to brevity and because: 1) a non-trivial optimization of soft constraints is likely to shrink the search space leading to a small and mostly homogeneous Pareto front; 2) Pareto front

generation allows a domain expert to investigate soft constraint satisfaction and its trade-off against quality without explicitly optimizing it.

5.8 Experimental Setup

In this section, we describe the details of our implementation, baselines that we compare against, method that we use for generating constraints, and how and on which datasets we evaluate our approach.

5.8.1 Implementation

Our approach is implemented in Python and Java and we use the Loandra solver [34] to solve the encodings that we generate. Loandra is an any-time solver that guarantees optimality if run to completion, but can also produce intermediate solutions. We set a time limit of 30 minutes every time a call to the solver is made, a feasible solution is salvaged in case of a timeout if possible. We run experiments on a server with two 12-core Intel E5-2697v2 CPUs and 128G of RAM.

5.8.2 Datasets

We run our experiments on a varied set of datasets ranging in terms of domain, size, and number of features. Our benchmarks include seven real datasets from the UCI repository [77] and four synthetic datasets from FCPS [272]. All of our datasets come with a given number of clusters that we use as an input for our encoding, and a ground-truth labeling that we use to evaluate our solutions. For each dataset, we fix the depth value to the lowest number that provides more than twice as many leaves as there are clusters. We normalize the values of each feature in all datasets to the range $[0, 100]$ so that features with larger values do not dominate the pairwise distances. The properties of each dataset alongside the chosen depth values are provided in Table 5.1. The real datasets are selected with emphasis on their features and how informative and meaningful they are in their corresponding domains. This consideration is important since feature interpretability, per the discussion in Section 3.7.3, lends itself to learning interpretable models.

5.8.3 Constraint Generation

To understand the effects of constraints on our approach, we synthetically generate pairwise clustering constraint sets that vary relative to the size of the dataset. Specifically, for a given κ value with $0 \leq \kappa \leq \frac{|X|-1}{2}$, we generate a set of $\kappa \cdot |X|$ random pairwise constraints following Wagstaff and Cardie [283]: (1) We generate $\kappa \cdot |X|$ pairs of data points without repetition; (2) We add each pair to the must-links (ML) constraint set if both data points share the same ground-truth label and to the cannot-links (CL) set otherwise. To mitigate

Dataset	$\|X\|$	$\|F\|$	k	d
Iris	150	4	3	3
Wine	178	13	3	3
Glass	214	9	7	4
Ionosphere	351	34	2	3
Seeds	210	7	3	3
Libras	360	90	15	5
Spam	4601	57	2	3
Lsun	400	2	3	3
Chainlink	1000	3	2	3
Target	770	2	6	4
WingNut	1016	2	2	3

Table 5.1: Real (top) and synthetic (bottom) datasets and their properties.

the potential bias due to the random generation of constraints, each experiment is run with 20 different seeds with the overall mean being reported.

5.8.4 Baselines

To evaluate the performance of our approach, we compare its results against its variations with regard to structure and objective, in addition to baselines from the literature.

Constrained Clustering

Constrained clustering (Problem 7), i.e., finding a clustering with approximated Pareto optimality of [MD, MS] without restriction to a decision tree, is our first baseline. To solve constrained clustering using our approach, we remove the tree-related components of the encoding. Namely, we remove the variables $a_{t,j}$, $s_{i,t}$, $z_{i,t}$, $g_{t,c}$ and the clauses in Eq. (5.7) to Eq. (5.18). Instead, we introduce the clauses in Eq. (5.50) that guarantee a well-formed unary encoding of labels.[2]

$$(x_{i,c}, \neg x_{i,c+1}) \qquad \forall x_i \in X, c \in [2..k-1] \tag{5.50}$$

While we do not include local search-based approaches for constrained clustering as baselines in our experiments, previous work found that they tend to be outperformed by combinatorial approaches in terms of solution quality and constraint utilization [66].

Maximum Diameter

Our second and third baselines are the variants of Problem 6 and Problem 7 that only focus on optimizing (an approximation of) the maximum diameter objective. In case of

[2] Note that these clauses are redundant in the encoding of tree clustering as the condition is already enforced for the leaves.

Problem 7 and no approximation (i.e., $\epsilon = 0.0$), the variant is equivalent to a well-known baseline formulated in previous works from the literature [67, 66]. In other words, it has the same set of (feasible and) optimal solutions as the baseline.

To solve the MD-only variations, we remove all MS-related components of the two encodings for Problem 6 and Problem 7. Namely, we remove b_w^+ variables, conditional co-clustering clauses in Eq. (5.30) and Eq. (5.31), clauses guaranteeing λ^* to be well-defined in Eq. (5.33) and Eq. (5.34), and soft clauses modelling the MS objective in Eq. (5.38).

Mixed Integer Optimization

To our knowledge, the model in Bertsimas, Orfanoudaki, and Wiberg [38] is the only approach in the literature that solves the problem of decision tree clustering by formulating it as a well-known combinatorial optimization problem, i.e., Mixed Integer Optimization (MIP). Their model is restricted to one leaf per cluster and optimizes the Silhouette objective given sufficient time. However, Bertsimas, Orfanoudaki, and Wiberg note that their model does not scale well, therefore not releasing their code nor presenting any experimental results. They instead focus on a heuristic procedure that does not have any solution quality guarantees and cannot naturally support clustering constraints.

Due to the relevance to our work and the lack of similar approaches in the literature, we aim to provide a comparison against the MIP model. Thus, we have implemented the model using the Gurobi v10 solver. Note that their model did not originally support clustering constraints and we extend it to incorporate the pairwise constraints that we support. Furthermore, we have detected and addressed a series of issues ranging from minor typographical ones to crucial ones that make it impossible to solve the model. We provide the details of our implemented version of Bertsimas, Orfanoudaki, and Wiberg's MIP formulation for decision tree clustering in Appendix A.

5.8.5 Evaluation

We use the ground-truth labels included in our benchmarks to evaluate the quality of clustering solutions obtained by our approach. To determine how closely our learned clustering resembles the ground-truth, we use the Adjusted Rand Index (ARI) and the Normalized Mutual Information (NMI) criteria (Section 5.2.1).

5.9 Experimental Results

In this section, we run an extensive set of experiments to evaluate our SAT encoding for decision tree clustering with support for constraints. We compare our approach against baselines (Section 5.9.1), examine the impact of the approximation parameter (Section 5.9.2), study infeasibility and the impact of the depth parameter (Section 5.9.3), investigate the benefits of soft constraints (Section 5.9.4), explore the Pareto fronts of our objective (Section 5.9.5), and examine the effects of pruning and approximation on performance (Section 5.9.6).

5.9.1 Comparison Against Baselines

In our first set of experimental results, presented in Table 5.2, we solve Problem 6 with $\epsilon = 0.1$ for ML and CL sets generated with various κ values. The results are then compared against three baseline variations achieved through removing the tree-clustering restriction and MS component of the objective, described in Section 5.8.4. We report average ARI and NMI metrics, average runtimes, and the number of feasible instances across 20 seeds. Note that a decision tree clustering problem with (consistent) ML and CL sets might be infeasible for a fixed depth. We exclude infeasible and unknown[3] cases from the computed average ARI and NMI values.

Note that the MIP formulation from Bertsimas, Orfanoudaki, and Wiberg [38], which we revised and implemented, is absent from our comparison. Our experiments with the MIP model found that it is unable to find a feasible solution for any of the datasets and the depths specified in Table 5.2 across multiple runs, both in settings with and without constraints. This result is consistent with Bertsimas, Orfanoudaki, and Wiberg's observation on the limited scalability of their MIP formulation and emphasizes the strong performance of our approach.

The results in Table 5.2 show that our approach can produce high quality interpretable solutions with non-zero ϵ values in the time limit. We observe that finding a tree or non-tree solution that is Pareto optimal in $[\text{MD}, \text{MS}]$ almost always leads to better ARI and NMI scores compared to only optimizing MD. The cases where the MD-only objective achieves better scores mostly correspond to empty ML and CL sets ($\kappa = 0$).

Interestingly, we observe that when the tree clustering problem is feasible, it manages to produce higher quality solutions compared to its non-tree counterpart, constrained clustering (CC), in both objectives. The exceptions are mostly limited to the Chainlink dataset with the $[\text{MD}, \text{MS}]$ objective and the $\kappa = 0$ case with the MD objective. It seems unintuitive for tree clustering to produce higher quality solutions as it is strictly less expressive than CC. We conjecture that both clustering objectives tend to perform better with tree clustering because of its inherently restricted yet structured solution space, resulting in potentially worse objective values but higher ARI scores.

Figure 5.1 summarizes the results from Table 5.2 and highlights the average benefits of tree clustering ($\circ$), the bi-criteria objective (blue and dotted), and constraints across all datasets. In the presence of no constraints, the ARI scores of the four approaches are fairly close. However, adding a moderate number of constraints causes the CC approach with the MD objective to fall behind significantly, highlighting the importance of using either the bi-criteria objective or tree clustering. Furthermore, all approaches employing tree clustering or the bi-criteria objective always achieve better ARI scores when increasing κ. The tree clustering with bi-criteria approach always outperforms the baselines when any constraint is present, with the difference most notable with a low number of constraints. The downside to using the tree clustering approaches, however, is evident in the number of feasible solution

[3] Instances where the solver timed out or crashed without finding a feasible solution are considered "unknown".

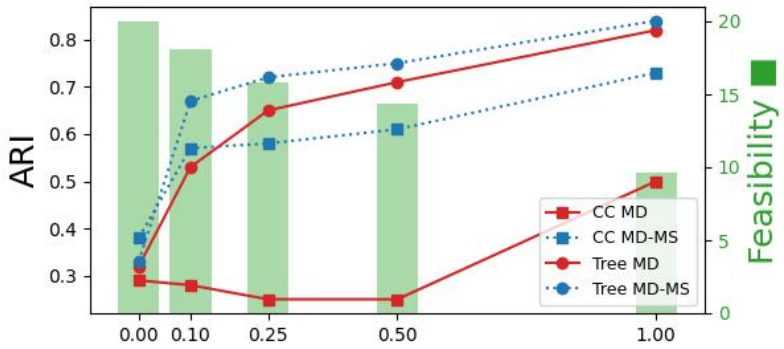


Figure 5.1: ARI and feasibility results from Table 5.2 averaged over 11 datasets.

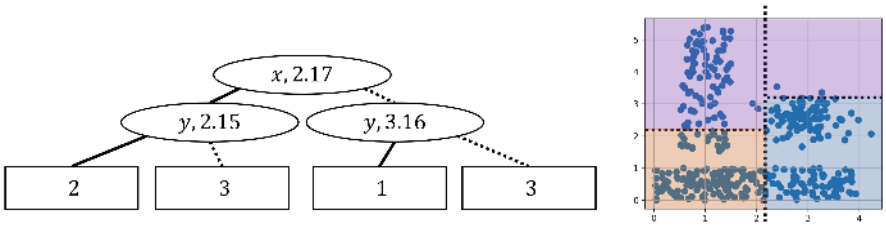


Figure 5.2: A Pareto optimal (in [MD, MS], $\epsilon = 0$) decision tree and its corresponding clustering solution for the Lsun dataset.

decreasing as κ increases, exposing the trade-off between interpretability via decision trees and satisfaction of user-provided clustering constraints. Note that the number of feasible solutions corresponds to the tree clustering with the [MD, MS] objective, but the results for the MD objective are the same in all but two cases.

Figure 5.2 shows a decision tree of depth 2 found as a Pareto optimal ($\epsilon = 0$) clustering solution for the synthetic Lsun dataset without pairwise constraints ($\kappa = 0$). We see that the solution fails to find the intended clusters in the data and instead focuses on a clustering with a significantly smaller maximum diameter. Figure 5.3 shows the solution that is found in a similar setting with just 4 pairwise constraints added ($\kappa = 0.01$). We see that the intended clusters are found and perfect accuracy (ARI of 1.0) is achieved. This example demonstrates how a minimal number of constraints combined with the structure of decision trees can significantly improve solution quality. Note that in non-tree approaches, such as CC or Kmeans with constraints, constraints have lower impact on the points that are not involved directly. Thus, a larger number of them is required to alter the overall structure of the solution in a significant way.

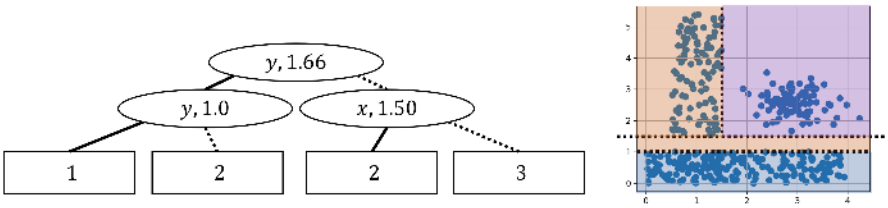


Figure 5.3: A Pareto optimal (in [MD, MS], $\epsilon = 0$) decision tree with $\kappa = 0.01$ and its corresponding clustering solution for the Lsun dataset.

5.9.2 Approximation

Next, we experiment with different values for the approximation parameter ϵ to understand the trade-off between performance and more precise approximation. We solve Problem 6 for each dataset and its corresponding highest κ value from Table 5.2 that resulted in 20 out of 20 feasible solutions. We do so to understand approximation in the presence of constraints but without infeasibility concerns. We choose the [MD, MS] objective and try 4 different values ranging from 0.01 to 10 for the ϵ parameter. Note that we normalize feature values to $[0, 100]$, so any value above 10 for ϵ is, intuitively, seen as too extreme.

The impact of the approximation parameter as shown by the results in Table 5.3 are as expected. Specifically, we see a significant improvement in runtime as ϵ increases for all datasets except for Libras. Moreover, less accurate solutions are found with larger ϵ values in all cases except for Seeds and Chainlink datasets. The negative impact on solution quality is shown to be minimal in most cases, demonstrating the robustness of the tree clustering algorithm to approximation. The robustness can be due to the structure of decision tree solutions minimizing the negative impact of assigning more pairs in the same distance class on the individual treatment of points. Note that since we chose feasible values for κ, we find only feasible solutions in all but 1 case. This particular instance is also feasible, but the solver failed to produce a solution due to a timeout.

5.9.3 Depth and Infeasibility

Our next set of experimental results investigates the effects of the depth parameter on quality and feasibility of decision tree solutions. We solve Problem 6 with three depth values for each dataset, centered around the values specified in Table 5.1. The results presented in Table 5.4 show that increasing depth can help solve the infeasibility issue in almost all of the datasets, but it is not guaranteed to do as it fails to produce any feasible solutions for the Spam dataset even at a depth of four. Furthermore, the results show that increasing the depth beyond the point of feasibility will, in all but one case, lead to lower or equal average ARI and NMI scores. This observation is in line with our claim based on Table 5.2 that the limited structure of trees improves the clustering quality, since shallow trees are more restricted than deeper ones.

5.9.4 Soft Constraints

Results from Table 5.4 have underlined the challenge in balancing solution feasibility and accuracy when adding constraints to a fixed-depth decision tree. One way to approach this issue is to address infeasibility through the use of soft constraints that were introduced in Section 5.6. For our next experiment, we solve Problem 6 with the 1-stage (Section 5.6.1), 2-stage (Section 5.6.2), and 3-stage schemes to soft constraint optimization. For each approach, we use both non-chained and Euclidean chained variations. We use higher values of κ for this experiment, as the 2-stage approach is equipped to deal with the infeasibility due to a higher number of constraints. We report the time spent by the solver in each stage independently to distinguish which optimization is the most challenging in multi-stage schemes.

Table 5.5 and Table 5.6 present the results for the non-chained and chained variations of the 1-stage scheme, respectively. Compared to the results in Table 5.2, the 1-stage approach achieves slightly worse solutions in feasible instances but manages to produce solutions in infeasible instances that are more constrained. By maintaining feasibility, it allows us to continually improve ARI and NMI scores by adding constraints without the risk of infeasibility. The results for the non-chained and chained variations of the 2-stage scheme are presented in Table 5.7 and Table 5.8, respectively. We observe that the 2-stage scheme is also similarly enabling a higher utilization of clustering constraints to improve accuracy. When comparing against the 1-stage in feasibility and accuracy, we see that each approach is superior in some cases. The instability in comparison could be due to the 1-stage approach being able to achieve higher objective values in theory but being more computationally demanding in practice. Table 5.9 and Table 5.10 present the results for the non-chained and chained variations of the 3-stage scheme, respectively. Similar to before, we see a higher utilization of constraints that lead to higher quality solutions and an instability when comparing against the 1-stage and 2-stage schemes. Note that any comparison between different schemes should be done in caution as they employ different number of calls to the solver and, as a result, vary in the total time limit that they are given.

In comparison against the non-chained variations, we see that employing the chained method will usually result in lower or equal average scores. Exceptions include the Spam dataset for the 1-stage approach and the Ionosphere dataset for the 3-stage approach. The chained variations are also able to produce less feasible solutions. The number of cases in which less than 20 feasible solutions are produced increases from 4 to 13 for 1-stage, 4 to 6 for 2-stage, and 5 to 7 for 3-stage schemes. We also report the percentage of must-links and cannot-links satisfied by each approach. We see that most schemes maintain an above 90% satisfaction for both types of pairwise constraints. The exceptions are 21 cases in non-chained 1-stage, 26 in chained 1-stage, 20 in non-chaind 2-stage, 28 in chained 2-stage, 24 in non-chained 3-stage, and 39 in chained 3-stage. A comparison shows that chained variations are able to satisfy less constraints, which matches our expectation as a linearized satisfaction of links is more limited.

We observe that the chained method outperforms its non-chained counterpart in terms of runtime for Chainlink in 2-stage and 3-stage approaches and for Ionosphere and Seeds across schemes. However, we also see ML-domination, as described in Section 5.6.3, occurring in cases with a considerable intersection. Specifically, we see the ML and CL satisfaction rates to converge to 100% and 0%, respectively, as κ is increased for Ionosphere and Spam in 1-stage and 2-stage and Chainlink in 2-stage schemes, nullifying the improvement in runtime. In fact, a shorter runtime translates to a higher quality solution only for Ionosphere in the 3-stage scheme. Overall, our observations suggest that while the chained method is useful for prioritizing link satisfaction, it is not suitable to be employed for improving the performance. It can also cause ML-domination where as the non-chained variation do not. Although, the ML-domination can be addressed and the accuracy can be improved by using the 3-stage scheme, as evident by Ionosphere, Spam, and Chainlink datasets.

In Figure 5.4, we summarize and combine the ARI scores from Table 5.2, Table 5.5, Table 5.6, Table 5.7, Table 5.8, Table 5.9, and Table 5.10. We compare in an ascending order of κ values the multiple soft constraint schemes and their chained and non-chained variations against the original approach of treating links as hard constraints. We report the number of feasible solutions within the figure if any of the instances fail to produce a solution, with a cross mark indicating a failure across the board. The trends visible in the figure matches our observations from the tables. Namely, all soft constraint approaches can improve solution quality by incorporating a higher number of constraints without infeasibility. Moreover, the non-chained methods perform better than their chained counterparts in accuracy and number of feasible solutions.

5.9.5 Pareto Front Exploration

For our next set of experimental results, we focus on solving Problem 8. We inquire whether exploring a complete Pareto optimal front leads to significantly higher clustering accuracy compared to retrieving an arbitrary Pareto optimal solution. To provide a fair comparison and avoid putting the solutions of Problem 6 at a disadvantage, we consider a global timeout limit (30 minutes) for generating the Pareto front. A global timeout limit spans over multiple calls to the solver. Specifically, it limits every call to the solver to the remaining time, i.e., the global limit minus the overall time spent. Note that Algorithm 2 requires every optimization step to be optimal. Thus, we stop the algorithm whenever the solver times out and produces a potentially non-optimal solution or no solution at all. An incomplete front would then consist of all of the optimal solutions produced up to that point and the potentially non-optimal last one. We use lower values for κ since a highly constrained instance tends to converge and have very few solutions in its complete Pareto front.

In Table 5.11, we report the size, maximum ARI score, and maximum NMI score of the Pareto optimal fronts averaged across seeds. Furthermore, we highlight the number of cases where a complete Pareto front is successfully generated (Comp.), where an incomplete front is generated (Inc.), and when no feasible solution is found (Inf.). Lastly, we report average

ARI and NMI scores from Table 5.2 again to compare against maximum scores across each front. We observe that finding a complete Pareto optimal front and selecting the best solution leads to comparable quality when a moderate amount of constraints are present. In non-constrained ($\kappa = 0$) cases however, the best solution of the front is able to achieve significantly higher scores in 5 datasets. The similar impact of Pareto front generation and constraints is expected since increasing the constraints forces the front to converge on a few high-quality solutions. We observe the number of solutions in the Pareto front to almost monotonically decrease as more constraints are added, with a few minor increases from $\kappa = 0.1$ to $\kappa = 0.25$ due to high-quality solutions becoming infeasible and being replaced with more than one dominated ones. Lastly, we see feasible solutions produced in as many instances as the singular Pareto optimal solution approach (Table 5.2), with the only minor improvement occurring for Ionosphere at $\kappa = 0.25$.

5.9.6 Ablation

In our last experiment, we study the impact of the smart pairs pruning algorithm (Section 5.5) and objective approximation via distance classes (Section 5.3) on the performance of our approach. We solve Problem 6 in three settings of pruning and non-zero ϵ, non-zero ϵ with no pruning, and no pruning with zero ϵ. Note that $\epsilon = 0$ corresponds to not utilizing distance classes and approximation. We omit presenting ablation results for soft constraints (Section 5.6) since: 1) their final objective optimization is not as challenging due to maximal constraint satisfaction confining the search space; 2) there cannot be a direct comparison in number of clauses and runtime in a multi-stage approach, as the result of each stage can affect the following ones. In addition to validation criteria and runtime, we report the average number of clauses alongside how many seeds lead to: 1) feasible solutions (Feas.); 2) solver declaring that no solution exists (Inf.); 3) solver terminating without finding a solution or claiming infeasibility (Unk.); and 4) an out-of-memory error (Mem.).

The results in Table 5.12 show that employing pruning and approximation allows us to significantly reduce the runtime and the number of clauses. The improvement in performance translates to finding higher quality solutions in all cases except for Iris and Wine, easy instances that enjoy a slight benefit in accuracy when approximation of the objective is not in effect ($\epsilon = 0$). For Libras, Spam, and Lsun, the improved performance can help us find more feasible solutions and prove infeasibility for more cases. Note that unlike ϵ-approximation, the smart pairs algorithm is guaranteed not to change the set of feasible solutions and the set of optimal solutions. However, since we optimize $\lambda^- - \lambda^+$ and there can be multiple optimal solutions, the different encoding may still lead to a different solution with slightly different ARI and NMI scores, even if the timeout limit is not reached.

5.10 Related Works

The problem of constrained clustering without decision tree restriction has been solved in Dao, Duong, and Vrain [66] as combinatorial optimization problem. Their approach supports pairwise links as well as cluster-level constraints including bounds on the size and local density of clusters. Their focus is on the bi-criteria objective of maximum diameter and minimum split and the authors demonstrate that their approach can be embedded in a complete Pareto front generation algorithm. They use Constraint Programming (CP) and global constraints to outperform the two well-known Branch-and-Bound and Graph Coloring baselines, with the added benefit of CP expressiveness enabling the encoding of more complex constraints.

Constrained clustering can also be solved by local search approaches usually based on k-means. The CVQE algorithm [70] extends k-means with constraint support by adding penalties for violated constraints to the error function. The MPCK-means algorithm [42] is also based on k-means and integrates pairwise constraints with unsupervised data through learning distance metrics. Other clustering approaches such as hierarchical [69] and spectral [287] have also been extended with support for user-specified constraints.

Requiring a clustering solution to conform to a decision tree adds interpretability by revealing how each point is assigned to a cluster and which features impacted the decision making. However, the restriction on the solution space also comes with unavoidable cost to the clustering objective. Recent literature has proven lower bounds for the increase in cost and proposed increasingly tight upper bounds via algorithms with guaranteed approximation factors. Moshkovitz et al. [203] shows that local search approaches can have arbitrarily large cost and proves a $\Omega(\log k)$ lower bound factor for k-means and k-medians objectives. They further present algorithms for $O(k)$-approximation of k-medians and $O(k^2)$-approximation of k-means. Gamlath et al. [91] later improved the upper bounds to $O(\log^2 k)$ and $O(k \log^2 k)$ for the k-medians and k-means objectives respectively and proved new lower bound of $\Omega(k)$ for k-means. Relaxed variations of decision trees such as oblique ones are not subject to these limitations and can be learned to represent partitions founds by non-tree algorithms such as k-means [90]. Relaxing the decision tree structure could also be beneficial in enabling more efficient learning algorithms, as is the case with soft decision trees and continuous optimization [60].

The DL8.5 algorithm presented in Aglin, Nijssen, and Schaus [6] is a decision tree learning approach based on Branch-and-Bound search and Dynamic Programming. DL8.5 can optimize an additive cost function for leaves by recursively exploring all possible splits within a decision tree. In addition to bounding and pruning the search space, caching is used to improve the search performance through exploiting the fact that the same subset of data might appear in multiple subproblems. DL8.5 was originally introduced for the problem of classification but was later shown to also work with an additive cost for clustering [7]. The cost being optimized is the sum of Euclidean distance for each point and the centroid of its cluster. DL8.5 for clustering does not support a cluster to be spread across multiple leaves

and neither can it naturally support constraints due to its search-based framework.

A combinatorial optimization approach to the decision tree clustering problem without constraints is presented in Bertsimas, Orfanoudaki, and Wiberg [38]. Building upon the authors' previous decision tree classification work [37], the problem is formulated as a Mixed Integer Optimization (MIO) instance that optimizes the Silhouette Metric [236]. However, the authors highlight the lack of scalability of their formulation and instead focus on an heuristic-based local search algorithm. Their iterative coordinate-descent approach is limited to one leaf per cluster and approximates a solution in the Silhouette Metric [236] or the Dunn index [78] validation criteria without any approximation guarantees. The authors' line of work in repurposing a model initially conceived for classification to a clustering context reflects that of ours in Chapter 3 and this chapter. However, the extension of our approach to clustering differs in the sense that it also adds support for user-provided clustering constraints.

5.11 Conclusion and Future Work

We present a MaxSAT encoding of decision tree clustering with pairwise constraints as the first combinatorial optimization approach to interpretable constrained clustering. Our approach reuses the direct non-binary branchings first presented in Chapter 3 and equips it with a clustering objective and support for constraints. Specifically, it optimizes the well-known bi-criteria objective of maximum diameter and minimum split with guaranteed ϵ-approximation of a Pareto optimal solution. We also detail how our encoding can be employed in an iterative algorithm that provides approximations of all, rather than one, of Pareto optimal solutions. The complete approximation of the Pareto front allows the user to choose a viable solution based on domain-specific expertise or heuristics.

Our solutions are interpretable as the decision making for placing a data point into a cluster is transparent. They are also constrained since must-links and cannot-links can represent domain-specific knowledge that should be adhered to, ranging from instance-level constraints to higher level constraints regarding clusters that can be translated to pairwise ones. Using the formulation approach with the declarative combinatorial optimization problem of MaxSAT allows us to have guarantees on solution quality, flexibility in specifying constraints, and the ability to spread a cluster amongst multiple leaf nodes. All areas where most of the current body of work on decision tree clustering, which relies on local search and heuristics, fail.

We integrate must-links and cannot-links with our objective into the Smart Pairs algorithm to provide a pruning framework that detects redundant and infeasible clauses in the time of encoding. As a result, not every clause that we describe in the paper will end up in the instances that we solve, even though there is an equivalency in feasible solutions and the objective value. Smart pairs demonstrates that formulation in a combinatorial optimization problem can be pruned in an automated way to improve performance, as long as equivalency with the original formulation is proven.

We further add support for soft constraints in 1-stage, 2-stage, and 3-stage approaches to address the potential issue of infeasibility and pave the way for utilizing a larger amount of semi-supervision through pairwise constraints. The soft approach to constrained ML lacks guaranteed enforcement, but is still clear in how constraint satisfaction is optimized and which constraints are enforced in the end. Our approach can be easily modified to support hard and soft pairwise constraints simultaneously, with the hard ones representing strict user-provided guidelines and the soft ones mostly utilized for accuracy. Moreover, our chained approach to link satisfaction provides means for the user to specify their priority and influence how constraint satisfaction is optimized.

We experimentally show that we can find interpretable solutions of high quality in short runtimes, being the first exact approach to do so. We further show that decision trees, the bi-criteria objective, and pairwise constraints can improve solution quality on their own and through complementing each other. This is a very important result demonstrating that restriction to interpretable solutions, surprisingly, improves accuracy, expanding upon one of the central claims in the literature of interpretable ML [237] and extending it to the problem of clustering. The improvement can hypothetically be explained by observing that the natural structure and simulatability in interpretable solutions aptly complements the clustering objective in aspects that it does not encompass. Moreover, the results show that clustering constraints improve the solution accuracy in the context of decision tree solutions, similar to how they do for non-tree ones [283].

Our experiments demonstrate the trade-off between solution quality and feasibility. We show how the support for soft constraints in our proposed schemes allows us to consider a larger set of constraints without infeasibility and further improve accuracy. Moreover, we run the complete Pareto front generation algorithm and show that solutions of higher quality can be found when there is an absence of constraints. Picking a solution from the front can be seen as another form of user involvement that can replace the utilization of constraints in improving accuracy. Lastly, we perform ablation studies and highlight the impact of ϵ-approximation and Smart Pairs on the size of the produced encodings and, as a result, performance.

Our work can be extended in multiple interesting directions in the future. Multiple solutions of varying accuracy can have the same objective values of MD and MS. A secondary criterion, e.g., within cluster sum of squares [51], can be considered and optimized as a secondary stage to distinguish such solutions. Additionally, experiments can be performed to study the simultaneous optimization of the bi-criteria objective and soft pairwise constraint satisfaction. Examples include weighing one objective against the other or producing all Pareto optimal solutions. In any case where multiple objectives are being optimized, alternative strategies and tools, e.g., multi-objective solvers [143], can be investigated and empirically evaluated for exploring the space of all Pareto optimal solutions more efficiently. Another direction to enhance performance is more thorough pruning. The Smart Pairs algorithm can be equipped with more sophisticated algorithms from graph theory to detect more infeasible and redundant cases. For example, the current version permits the infea-

sible case in which there are two clusters and three data points with cannot-links between them. It also does not detect redundancy when there are two clusters and a must-link is being added between two points that both are connected to a third point with cannot-links. Lastly, the decision tree encoding can be extended to new objectives and new constraints with potential integration into the Smart Pairs algorithm in interesting ways.

Data	κ	Tree [MD, MS]				Tree MD			CC [MD, MS]		CC MD	
		ARI	NMI	Time(s)	Feas.	ARI	NMI	Feas.	ARI	NMI	ARI	NMI
Iris	0	0.60	0.67	0.85	20	0.60	0.67	20	0.60	0.67	**0.70**	**0.72**
	0.1	**0.83**	**0.82**	0.76	20	0.70	0.71	20	**0.82**	0.82	0.57	0.61
	0.25	**0.85**	**0.84**	0.68	20	0.77	0.77	20	0.79	0.79	0.60	0.62
	0.5	**0.91**	**0.89**	0.68	20	0.87	0.85	20	0.80	0.79	0.58	0.61
	1	**0.95**	**0.93**	0.70	13	0.94	0.92	13	0.91	0.89	0.68	0.68
Wine	0	0.00	0.02	1.79	20	0.14	**0.20**	20	0.00	0.02	**0.17**	**0.20**
	0.1	**0.71**	**0.70**	1.23	20	0.39	0.42	20	0.51	0.50	0.37	0.38
	0.25	**0.80**	**0.78**	2.19	20	0.60	0.60	20	0.52	0.51	0.22	0.25
	0.5	**0.84**	**0.81**	4.10	20	0.75	0.71	20	0.54	0.53	0.18	0.20
	1	**0.94**	**0.92**	4.00	20	0.90	0.87	20	0.71	0.67	0.20	0.20
Glass	0	0.23	0.38	7.91	20	0.22	0.34	20	0.23	**0.39**	**0.24**	0.37
	0.1	**0.22**	**0.33**	10.63	20	0.19	0.30	20	0.21	0.32	**0.22**	0.32
	0.25	**0.22**	**0.31**	75.82	20	0.18	0.29	20	0.16	0.26	0.16	0.24
	0.5	**0.28**	**0.35**	1098.45	20	0.26	**0.35**	20	0.15	0.23	0.13	0.21
	1	-	-	301.54	0	-	-	0	**0.14**	**0.20**	0.08	0.15
Ionosphere	0	0.01	0.02	11.65	20	0.06	**0.06**	20	0.01	0.02	**0.09**	**0.06**
	0.1	**0.31**	**0.25**	32.77	20	0.11	0.07	20	0.14	0.12	0.04	0.03
	0.25	**0.51**	**0.40**	664.44	13	0.49	0.38	14	0.21	0.16	0.07	0.04
	0.5	-	-	57.53	0	-	-	0	0.39	**0.32**	0.10	0.06
	1	-	-	8.38	0	-	-	0	**0.78**	**0.70**	0.70	0.62
Seeds	0	**0.71**	0.67	1.47	20	0.59	0.58	20	0.68	0.65	**0.71**	**0.69**
	0.1	**0.68**	**0.66**	0.98	20	0.64	0.62	20	0.63	0.62	0.53	0.56
	0.25	**0.73**	**0.71**	1.34	20	0.72	0.69	20	0.61	0.61	0.51	0.55
	0.5	**0.79**	**0.76**	1.89	18	0.77	0.74	18	0.62	0.61	0.55	0.58
	1	-	-	1.18	0	-	-	0	0.78	0.76	0.71	0.70
Libras	0	0.18	0.44	1811.31	20	0.18	0.44	20	**0.19**	**0.45**	0.14	0.38
	0.1	**0.18**	**0.44**	1404.89	20	0.15	0.41	20	0.17	0.43	0.14	0.37
	0.25	**0.17**	**0.42**	1139.53	20	0.14	0.38	20	0.14	0.38	0.11	0.34
	0.5	**0.16**	**0.41**	787.08	20	0.14	0.37	20	0.12	0.34	0.09	0.30
	1	**0.16**	0.40	1673.42	13	0.14	0.37	12	0.11	0.32	0.08	0.27
Spam	0	0.00	0.00	309.24	20	-0.01	0.01	20	0.00	0.00	**0.13**	**0.09**
	0.1	-	-	292.93	0	-	-	0	**0.06**	**0.04**	0.04	0.03
	0.25	-	-	194.23	0	-	-	0	0.04	0.03	0.05	**0.04**
	0.5	-	-	136.92	0	-	-	0	**0.08**	**0.05**	0.06	0.03
	1	-	-	105.56	0	-	-	0	0.61	0.55	**0.62**	**0.56**
Lsun	0	**0.40**	**0.51**	2.04	20	**0.40**	0.50	20	**0.40**	**0.51**	0.38	0.49
	0.1	**0.90**	**0.91**	1.82	20	0.68	0.69	20	0.74	0.77	0.31	0.33
	0.25	**1.00**	**1.00**	1.49	20	0.88	0.86	20	0.90	0.90	0.32	0.31
	0.5	**1.00**	**1.00**	1.45	20	0.95	0.93	20	**1.00**	**1.00**	0.31	0.29
	1	**1.00**	**1.00**	1.22	20	0.98	0.97	20	**1.00**	**1.00**	0.49	0.39
Chainlink	0	0.24	0.19	6.70	20	0.00	0.00	20	**1.00**	**1.00**	0.06	0.05
	0.1	0.87	0.80	6.20	19	0.83	0.75	19	**1.00**	**1.00**	0.06	0.06
	0.25	0.93	0.87	3.84	1	0.90	0.83	1	**1.00**	**1.00**	0.03	0.03
	0.5	-	-	3.33	0	-	-	0	**1.00**	**1.00**	0.03	0.03
	1	-	-	3.01	0	-	-	0	**1.00**	**1.00**	0.63	0.56
Target	0	0.33	0.39	14.16	20	0.31	0.34	20	0.06	0.22	**0.38**	**0.42**
	0.1	**1.00**	**1.00**	6.39	20	0.64	0.60	20	**1.00**	**1.00**	0.57	0.54
	0.25	**1.00**	**1.00**	6.79	20	0.88	0.81	20	**1.00**	**1.00**	0.40	0.37
	0.5	**1.00**	**1.00**	8.99	20	0.95	0.91	20	**1.00**	**1.00**	0.23	0.24
	1	**1.00**	**1.00**	7.46	20	0.98	0.96	20	**1.00**	**1.00**	0.44	0.38
Wingnut	0	**1.00**	**1.00**	6.50	20	**1.00**	**1.00**	20	**1.00**	**1.00**	0.16	0.16
	0.1	**1.00**	**1.00**	3.54	20	**1.00**	**1.00**	20	**1.00**	**1.00**	0.21	0.18
	0.25	**1.00**	**1.00**	3.39	20	**1.00**	**1.00**	20	**1.00**	**1.00**	0.29	0.25
	0.5	**1.00**	**1.00**	3.37	20	**1.00**	**1.00**	20	**1.00**	**1.00**	0.49	0.42
	1	**1.00**	**1.00**	3.29	20	**1.00**	**1.00**	20	**1.00**	**1.00**	0.82	0.75

Table 5.2: Interpretable tree clustering with constraints for $\epsilon = 0.1$ averaged over 20 runs and compared against three baselines.

Dataset	κ	ϵ	Feas.	ARI	NMI	Time (s)
Iris	0.5	0.01	20	0.90	0.88	5.96
		0.1	20	0.91	0.89	0.68
		1	20	0.90	0.88	1.00
		10	20	0.86	0.84	0.94
Wine	1	0.01	20	0.93	0.91	18.37
		0.1	20	0.94	0.92	4.00
		1	20	0.93	0.91	6.71
		10	20	0.91	0.88	3.27
Glass	0.5	0.01	20	0.27	0.33	1238.01
		0.1	20	0.28	0.35	1098.45
		1	20	0.27	0.34	834.19
		10	19	0.24	0.31	366.12
Ionosphere	0.1	0.01	20	0.31	0.24	59.71
		0.1	20	0.31	0.25	32.77
		1	20	0.28	0.21	37.20
		10	20	0.28	0.21	15.25
Seeds	0.25	0.01	20	0.73	0.70	14.72
		0.1	20	0.73	0.71	1.34
		1	20	0.72	0.70	1.32
		10	20	0.75	0.71	1.13
Libras	0.5	0.01	20	0.18	0.42	887.06
		0.1	20	0.16	0.41	787.08
		1	20	0.18	0.42	974.55
		10	20	0.15	0.39	1031.73
Spam	0	0.01	20	0.00	0.00	86.22
		0.1	20	0.00	0.00	309.24
		1	20	0.00	0.00	35.04
		10	20	0.00	0.00	34.01
Lsun	1	0.01	20	1.00	1.00	16.38
		0.1	20	1.00	1.00	1.22
		1	20	1.00	1.00	1.15
		10	20	1.00	1.00	1.12
Chainlink	0	0.01	20	0.10	0.08	22.97
		0.1	20	0.24	0.19	6.70
		1	20	0.07	0.05	3.21
		10	20	0.51	0.47	2.80
Target	1	0.01	20	1.00	1.00	12.92
		0.1	20	1.00	1.00	7.46
		1	20	1.00	1.00	2.86
		10	20	1.00	1.00	2.89
Wingnut	1	0.01	20	1.00	1.00	25.04
		0.1	20	1.00	1.00	3.29
		1	20	1.00	1.00	1.84
		10	20	1.00	0.99	1.85

Table 5.3: Tree clustering with the [MD, MS] objective for different ϵ values averaged over 20 runs.

Dataset	Depth	ARI	NMI	Feas.	Time (s)
Iris	2	-	-	0	0.55
	3	0.90	0.88	20	0.69
	4	0.86	0.85	20	0.93
Wine	2	0.90	0.87	1	0.71
	3	0.85	0.82	20	4.42
	4	0.77	0.75	20	5.25
Glass	3	-	-	0	4.61
	4	0.29	0.36	20	1166.98
	5	0.25	0.31	20	46.48
Ionosphere	2	-	-	0	1.83
	3	-	-	0	59.87
	4	0.67	0.55	13	1250.52
Seeds	2	-	-	0	0.74
	3	0.81	0.77	18	1.91
	4	0.76	0.73	20	2.20
Libras	4	0.17	0.40	2	1805.05
	5	0.16	0.41	20	814.88
	6	0.15	0.39	20	496.58
Spam	2	-	-	0	73.70
	3	-	-	0	140.83
	4	-	-	0	629.25
Lsun	2	1.00	1.00	20	1.40
	3	1.00	1.00	20	1.49
	4	1.00	1.00	20	1.68
Chainlink	2	-	-	0	2.78
	3	-	-	0	3.32
	4	1.00	1.00	20	5.04
Target	3	-	-	0	7.19
	4	1.00	1.00	20	8.84
	5	1.00	1.00	20	9.60
Wingnut	2	1.00	1.00	20	2.91
	3	1.00	1.00	20	3.35
	4	1.00	1.00	20	4.03

Table 5.4: Tree clustering with the [MD, MS] objective for different tree depths ($\kappa = 0.5$, $\epsilon = 0.1$) averaged over 20 runs.

Dataset	κ	ARI	NMI	ML%	CL%	Solver Time (s) S1	Feas.
Iris	0.1	0.83	0.83	100.00	100.00	0.72	20
	0.25	0.87	0.85	100.00	100.00	1.27	20
	0.5	0.90	0.88	100.00	100.00	1.30	20
	1	0.95	0.93	99.58	99.85	12.59	20
	1.5	0.97	0.96	99.44	99.77	42.14	20
	2	0.97	0.96	99.47	99.67	159.57	20
Wine	0.1	0.69	0.68	100.00	100.00	1.54	20
	0.25	0.78	0.76	100.00	100.00	4.03	20
	0.5	0.85	0.82	100.00	100.00	8.84	20
	1	0.94	0.91	100.00	100.00	7.87	20
	1.5	0.97	0.95	100.00	100.00	4.30	20
	2	0.98	0.97	100.00	100.00	3.39	20
Glass	0.1	0.22	0.34	100.00	100.00	19.77	20
	0.25	0.22	0.31	100.00	100.00	179.09	20
	0.5	0.27	0.34	100.00	100.00	1323.76	20
	1	0.29	0.34	85.15	97.23	1800.03	20
	1.5	0.31	0.37	73.04	95.09	1800.03	20
	2	0.33	0.39	68.17	94.34	1800.03	20
Ionosphere	0.1	0.29	0.24	100.00	100.00	53.73	20
	0.25	0.49	0.37	99.54	98.96	1221.32	20
	0.5	0.66	0.54	96.40	92.22	1800.03	20
	1	0.68	0.57	93.69	86.18	1800.03	20
	1.5	0.71	0.61	92.87	86.56	1800.03	20
	2	0.71	0.60	91.50	84.84	1800.04	20
Seeds	0.1	0.68	0.66	100.00	100.00	1.42	20
	0.25	0.73	0.70	100.00	100.00	2.11	20
	0.5	0.81	0.78	99.89	99.93	13.23	20
	1	0.91	0.88	98.70	99.12	1246.82	20
	1.5	0.91	0.87	98.07	98.88	1316.63	13
	2	0.94	0.90	97.59	98.55	1317.04	12
Libras	0.1	0.17	0.43	100.00	100.00	1605.63	20
	0.25	0.16	0.41	100.00	100.00	1250.90	20
	0.5	0.15	0.39	100.00	100.00	998.27	20
	1	0.15	0.38	100.00	100.00	1584.48	20
	1.5	0.16	0.39	98.81	99.99	1774.70	20
	2	0.15	0.38	89.63	99.87	1800.12	20
Spam	0.1	0.41	0.32	81.95	72.61	1800.55	20
	0.25	0.41	0.32	79.79	66.21	1800.57	20
	0.5	0.35	0.27	78.26	57.80	1800.58	20
	1	0.18	0.14	84.83	34.79	1800.63	20
	1.5	0.23	0.18	74.44	48.66	1800.73	20
	2	0.20	0.17	84.22	35.58	1800.89	20
Lsun	0.1	0.90	0.91	100.00	100.00	2.09	20
	0.25	1.00	1.00	100.00	100.00	2.13	20
	0.5	1.00	1.00	100.00	100.00	2.66	20
	1	1.00	1.00	100.00	100.00	2.76	20
	1.5	1.00	1.00	100.00	100.00	2.11	20
	2	1.00	1.00	100.00	100.00	1.78	20
Chainlink	0.1	0.87	0.80	1.00	1.00	15.59	20
	0.25	0.92	0.86	0.99	0.99	641.31	20
	0.5	0.93	0.88	0.98	0.99	1246.58	10
	1	0.95	0.90	0.98	0.98	1796.83	19
	1.5	0.84	0.80	0.92	0.68	1800.06	20
	2	0.74	0.71	0.89	0.56	1800.07	20
Target	0.1	1.00	1.00	100.00	100.00	11.35	20
	0.25	1.00	1.00	100.00	100.00	13.88	20
	0.5	1.00	1.00	100.00	100.00	12.06	20
	1	1.00	1.00	100.00	100.00	13.61	20
	1.5	1.00	1.00	100.00	100.00	16.18	20
	2	1.00	1.00	100.00	100.00	10.60	20
Wingnut	0.1	1.00	1.00	100.00	100.00	2.18	20
	0.25	1.00	1.00	100.00	100.00	2.16	20
	0.5	1.00	1.00	100.00	100.00	2.15	20
	1	1.00	1.00	100.00	100.00	2.96	20
	1.5	1.00	1.00	100.00	100.00	2.80	20
	2	1.00	1.00	100.00	100.00	2.58	20

Table 5.5: Tree clustering ($[\text{MD}, \text{MS}]$ objective with $\epsilon = 0.1$) using non-chained 1-stage soft constraints method, averaged over 20 runs.

Dataset	κ	ARI	NMI	ML%	CL%	Solver Time (s) S1	Feas.
Iris	0.1	0.81	0.81	100.00	100.00	0.63	20
	0.25	0.85	0.84	100.00	100.00	0.74	20
	0.5	0.90	0.88	100.00	100.00	0.74	20
	1	0.95	0.93	99.34	99.23	10.09	20
	1.5	0.96	0.95	95.88	96.38	236.13	20
	2	0.95	0.94	92.49	93.40	376.12	18
Wine	0.1	0.73	0.72	100.00	100.00	1.34	20
	0.25	0.77	0.75	100.00	100.00	3.36	20
	0.5	0.86	0.83	100.00	100.00	8.01	20
	1	0.95	0.92	100.00	100.00	4.56	20
	1.5	0.97	0.95	100.00	100.00	2.28	20
	2	0.98	0.98	100.00	100.00	1.67	20
Glass	0.1	0.20	0.32	100.00	100.00	16.14	20
	0.25	0.23	0.31	100.00	100.00	109.17	20
	0.5	0.29	0.36	100.00	100.00	1249.39	20
	1	0.37	0.39	73.93	99.08	1800.05	20
	1.5	0.30	0.38	40.55	99.56	1800.24	20
	2	0.28	0.36	25.00	99.69	1800.14	20
Ionosphere	0.1	0.31	0.26	100.00	100.00	32.35	20
	0.25	0.48	0.37	99.57	98.22	756.63	20
	0.5	0.46	0.38	93.90	49.05	1593.15	14
	1	0.10	0.10	98.54	6.59	1727.26	16
	1.5	0.05	0.07	99.87	1.69	1199.43	9
	2	0.01	0.02	100.00	0.46	1414.61	12
Seeds	0.1	0.68	0.65	100.00	100.00	1.18	20
	0.25	0.73	0.70	100.00	100.00	1.88	20
	0.5	0.81	0.77	100.00	99.72	6.20	20
	1	0.87	0.83	90.22	96.17	597.80	20
	1.5	0.88	0.84	84.50	91.91	1155.85	15
	2	0.90	0.87	83.95	92.86	414.86	3
Libras	0.1	0.17	0.43	100.00	100.00	1618.48	20
	0.25	0.17	0.41	100.00	100.00	1266.77	20
	0.5	0.17	0.41	100.00	100.00	948.22	20
	1	0.16	0.39	100.00	100.00	1764.46	20
	1.5	0.13	0.36	98.18	100.00	1800.15	20
	2	0.15	0.38	86.23	100.00	1800.12	20
Spam	0.1	0.01	0.01	1.00	0.03	1800.50	20
	0.25	0.00	0.00	1.00	0.01	1800.49	20
	0.5	0.00	0.00	1.00	0.01	1800.51	20
	1	0.00	0.00	1.00	0.00	1800.54	20
	1.5	0.00	0.00	1.00	0.00	1770.74	17
Lsun	0.1	0.89	0.91	100.00	100.00	1.80	20
	0.25	1.00	1.00	100.00	100.00	2.70	20
	0.5	1.00	1.00	100.00	100.00	2.51	20
	1	1.00	1.00	100.00	100.00	2.23	20
	1.5	1.00	1.00	100.00	100.00	2.22	20
	2	1.00	1.00	100.00	100.00	2.25	20
Chainlink	0.1	0.86	0.79	100.00	99.87	8.98	20
	0.25	0.91	0.85	86.52	85.84	413.25	19
	0.5	0.93	0.88	68.45	76.32	760.56	6
	1	-	-	-	-	801.38	0
	1.5	-	-	-	-	699.45	0
	2	-	-	-	-	658.53	0
Target	0.1	1.00	1.00	100.00	100.00	13.39	20
	0.25	1.00	1.00	100.00	100.00	16.38	20
	0.5	1.00	1.00	100.00	100.00	14.50	20
	1	1.00	1.00	100.00	100.00	11.41	20
	1.5	1.00	1.00	100.00	100.00	8.12	20
	2	1.00	1.00	100.00	100.00	10.39	20
Wingnut	0.1	1.00	1.00	100.00	100.00	2.60	20
	0.25	1.00	1.00	100.00	100.00	2.04	20
	0.5	1.00	1.00	100.00	100.00	3.28	20
	1	1.00	1.00	100.00	100.00	3.53	20
	1.5	1.00	1.00	100.00	100.00	3.74	20
	2	1.00	1.00	100.00	100.00	2.85	20

Table 5.6: Tree clustering ([MD, MS] objective with $\epsilon = 0.1$) using chained 1-stage soft constraints method, averaged over 20 runs.

Dataset	κ	ARI	NMI	ML%	CL%	Solver Time (s) S1	Solver Time (s) S2	Feas.
Iris	0.1	0.85	0.84	100.00	100.00	0.24	0.71	20
	0.25	0.85	0.84	100.00	100.00	0.23	0.65	20
	0.5	0.91	0.89	100.00	100.00	0.27	0.65	20
	1	0.95	0.93	99.56	99.85	0.39	0.68	20
	1.5	0.97	0.96	99.40	99.80	0.54	0.57	20
	2	0.98	0.97	99.43	99.70	1.18	0.54	20
Wine	0.1	0.71	0.70	100.00	100.00	0.36	1.16	20
	0.25	0.80	0.78	100.00	100.00	0.53	2.12	20
	0.5	0.84	0.81	100.00	100.00	0.91	4.05	20
	1	0.94	0.92	100.00	100.00	1.08	3.91	20
	1.5	0.97	0.95	100.00	100.00	0.71	1.18	20
	2	0.98	0.97	100.00	100.00	0.68	1.02	20
Glass	0.1	0.22	0.33	100.00	100.00	1.74	10.67	20
	0.25	0.23	0.32	100.00	100.00	2.24	64.42	20
	0.5	0.28	0.36	100.00	100.00	36.77	1060.35	20
	1	0.29	0.36	83.52	98.65	1800.61	1559.42	19
	1.5	0.36	0.42	74.14	96.41	1800.61	490.84	20
	2	0.40	0.43	70.03	95.24	1800.61	206.34	20
Ionosphere	0.1	0.29	0.23	100.00	100.00	5.73	39.72	20
	0.25	0.51	0.40	99.91	98.89	575.71	468.46	20
	0.5	0.63	0.52	95.83	91.32	1800.72	85.88	20
	1	0.70	0.58	92.70	88.36	1800.72	24.26	20
	1.5	0.72	0.61	92.25	87.81	1800.72	13.85	20
	2	0.73	0.62	91.45	86.42	1800.72	10.67	20
Seeds	0.1	0.68	0.66	100.00	100.00	0.30	0.90	20
	0.25	0.73	0.71	100.00	100.00	0.36	1.27	20
	0.5	0.80	0.77	99.89	99.93	1.79	1.84	20
	1	0.91	0.87	98.65	99.16	32.85	1.32	20
	1.5	0.92	0.89	97.92	99.03	135.83	0.95	20
	2	0.94	0.90	97.61	98.77	356.80	0.79	20
Libras	0.1	0.18	0.43	100.00	100.00	37.05	1482.43	20
	0.25	0.18	0.43	100.00	100.00	45.52	1189.50	20
	0.5	0.15	0.40	100.00	100.00	75.18	762.91	20
	1	0.16	0.40	100.00	100.00	170.06	1733.30	12
	1.5	0.28	0.51	97.61	99.96	1126.86	1806.85	1
	2	-	-	-	-	1662.52	1806.84	0
Spam	0.1	0.46	0.36	84.02	75.70	1842.85	753.56	20
	0.25	0.46	0.36	81.06	70.75	1842.61	487.23	20
	0.5	0.41	0.33	79.90	62.88	1843.10	357.49	20
	1	0.33	0.27	76.73	58.29	1843.21	200.68	20
	1.5	0.23	0.19	79.70	44.01	1843.12	210.11	20
	2	0.30	0.24	76.89	52.49	1842.89	175.21	20
Lsun	0.1	0.90	0.91	100.00	100.00	1.38	1.68	20
	0.25	1.00	1.00	100.00	100.00	1.38	1.34	20
	0.5	1.00	1.00	100.00	100.00	0.58	1.29	20
	1	1.00	1.00	100.00	100.00	0.62	1.07	20
	1.5	1.00	1.00	100.00	100.00	0.64	0.89	20
	2	1.00	1.00	100.00	100.00	0.72	0.88	20
Chainlink	0.1	0.87	0.79	100.00	99.87	2.52	5.16	20
	0.25	0.91	0.85	99.13	99.19	15.67	3.60	20
	0.5	0.94	0.89	98.73	98.70	68.41	2.56	20
	1	0.95	0.91	98.53	98.32	347.40	1.95	20
	1.5	0.95	0.91	98.29	98.33	1036.76	1.88	20
	2	0.96	0.91	98.23	98.23	1440.14	1.85	20
Target	0.1	1.00	1.00	100.00	100.00	6.55	5.30	20
	0.25	1.00	1.00	100.00	100.00	7.55	5.90	20
	0.5	1.00	1.00	100.00	100.00	5.90	7.88	20
	1	1.00	1.00	100.00	100.00	5.05	6.26	20
	1.5	1.00	1.00	100.00	100.00	4.39	5.61	20
	2	1.00	1.00	100.00	100.00	4.74	6.39	20
Wingnut	0.1	1.00	1.00	100.00	100.00	2.04	1.98	20
	0.25	1.00	1.00	100.00	100.00	2.01	1.80	20
	0.5	1.00	1.00	100.00	100.00	1.99	1.80	20
	1	1.00	1.00	100.00	100.00	1.98	1.70	20
	1.5	1.00	1.00	100.00	100.00	1.99	1.69	20
	2	1.00	1.00	100.00	100.00	1.99	1.66	20

Table 5.7: Tree clustering ($[\text{MD}, \text{MS}]$ objective with $\epsilon = 0.1$) using non-chained 2-stage soft constraints method, averaged over 20 runs.

Dataset	κ	ARI	NMI	ML%	CL%	Solver Time (s) S1	Solver Time (s) S2	Feas.
Iris	0.1	0.85	0.84	100.00	100.00	0.23	0.73	20
	0.25	0.85	0.84	100.00	100.00	0.24	0.65	20
	0.5	0.91	0.89	100.00	100.00	0.26	0.66	20
	1	0.95	0.93	99.34	99.23	0.40	0.67	20
	1.5	0.96	0.94	95.88	96.38	0.55	0.58	20
	2	0.96	0.94	89.52	93.99	0.45	0.54	20
Wine	0.1	0.71	0.70	100.00	100.00	0.40	1.20	20
	0.25	0.80	0.78	100.00	100.00	0.53	2.13	20
	0.5	0.84	0.81	100.00	100.00	0.83	4.20	20
	1	0.94	0.92	100.00	100.00	0.93	3.92	20
	1.5	0.97	0.95	100.00	100.00	0.81	1.24	20
	2	0.98	0.97	100.00	100.00	0.84	1.04	20
Glass	0.1	0.22	0.33	100.00	100.00	1.64	10.62	20
	0.25	0.23	0.32	100.00	100.00	2.27	61.98	20
	0.5	0.28	0.36	100.00	100.00	30.88	1064.52	20
	1	0.38	0.40	70.88	99.56	1739.01	1724.04	17
	1.5	0.35	0.41	40.41	99.83	1560.61	1396.34	19
	2	0.30	0.36	25.80	99.89	1626.20	1138.19	20
Ionosphere	0.1	0.29	0.23	100.00	100.00	8.70	37.52	20
	0.25	0.50	0.38	98.97	98.83	645.55	514.49	20
	0.5	0.53	0.44	89.18	59.02	1261.87	258.77	20
	1	0.23	0.22	96.60	10.30	1005.53	60.27	20
	1.5	0.04	0.06	99.75	1.65	175.72	9.86	20
	2	0.01	0.02	100.00	0.40	89.96	6.71	20
Seeds	0.1	0.68	0.66	100.00	100.00	0.32	0.90	20
	0.25	0.73	0.71	100.00	100.00	0.41	1.27	20
	0.5	0.80	0.77	100.00	99.72	0.90	1.89	20
	1	0.87	0.83	90.22	96.17	3.96	1.82	20
	1.5	0.88	0.84	83.48	90.47	3.56	1.24	20
	2	0.88	0.85	77.07	86.96	2.64	1.03	20
Libras	0.1	0.18	0.44	100.00	100.00	37.38	1482.33	20
	0.25	0.17	0.43	100.00	100.00	46.90	1189.38	20
	0.5	0.15	0.39	100.00	100.00	77.92	772.54	20
	1	0.16	0.40	100.00	100.00	117.58	1735.75	12
	1.5	0.28	0.51	99.88	100.00	661.08	1806.90	1
	2	-	-	-	-	1566.54	1807.23	0
Spam	0.1	0.01	0.01	99.94	3.41	1843.51	1042.16	19
	0.25	0.00	0.00	99.91	1.38	1742.05	851.32	20
	0.5	0.00	0.00	99.83	0.64	1792.59	522.03	20
	1	0.00	0.00	99.87	0.24	1836.19	208.77	20
	1.5	0.00	0.00	95.98	0.07	1649.84	106.30	20
	2	0.00	0.00	97.94	0.03	1453.85	61.09	20
Lsun	0.1	0.90	0.91	100.00	100.00	1.10	1.66	20
	0.25	1.00	1.00	100.00	100.00	0.62	1.32	20
	0.5	1.00	1.00	100.00	100.00	0.70	1.28	20
	1	1.00	1.00	100.00	100.00	0.73	1.06	20
	1.5	1.00	1.00	100.00	100.00	0.72	0.88	20
	2	1.00	1.00	100.00	100.00	0.74	0.87	20
Chainlink	0.1	0.87	0.79	100.00	99.87	2.18	5.17	20
	0.25	0.90	0.84	86.30	84.89	6.88	4.77	20
	0.5	0.92	0.87	53.78	79.01	9.91	4.84	20
	1	0.42	0.40	67.92	34.84	15.50	3.47	20
	1.5	0.05	0.05	96.64	3.68	13.57	2.05	20
	2	0.00	0.00	100.00	0.08	15.01	1.95	20
Target	0.1	1.00	1.00	100.00	100.00	9.27	5.55	20
	0.25	1.00	1.00	100.00	100.00	5.86	5.90	20
	0.5	1.00	1.00	100.00	100.00	7.34	7.97	20
	1	1.00	1.00	100.00	100.00	5.28	6.32	20
	1.5	1.00	1.00	100.00	100.00	4.57	5.46	20
	2	1.00	1.00	100.00	100.00	4.54	6.52	20
Wingnut	0.1	1.00	1.00	100.00	100.00	2.06	1.95	20
	0.25	1.00	1.00	100.00	100.00	2.06	1.82	20
	0.5	1.00	1.00	100.00	100.00	2.07	1.78	20
	1	1.00	1.00	100.00	100.00	2.14	1.77	20
	1.5	1.00	1.00	100.00	100.00	2.14	1.71	20
	2	1.00	1.00	100.00	100.00	2.21	1.65	20

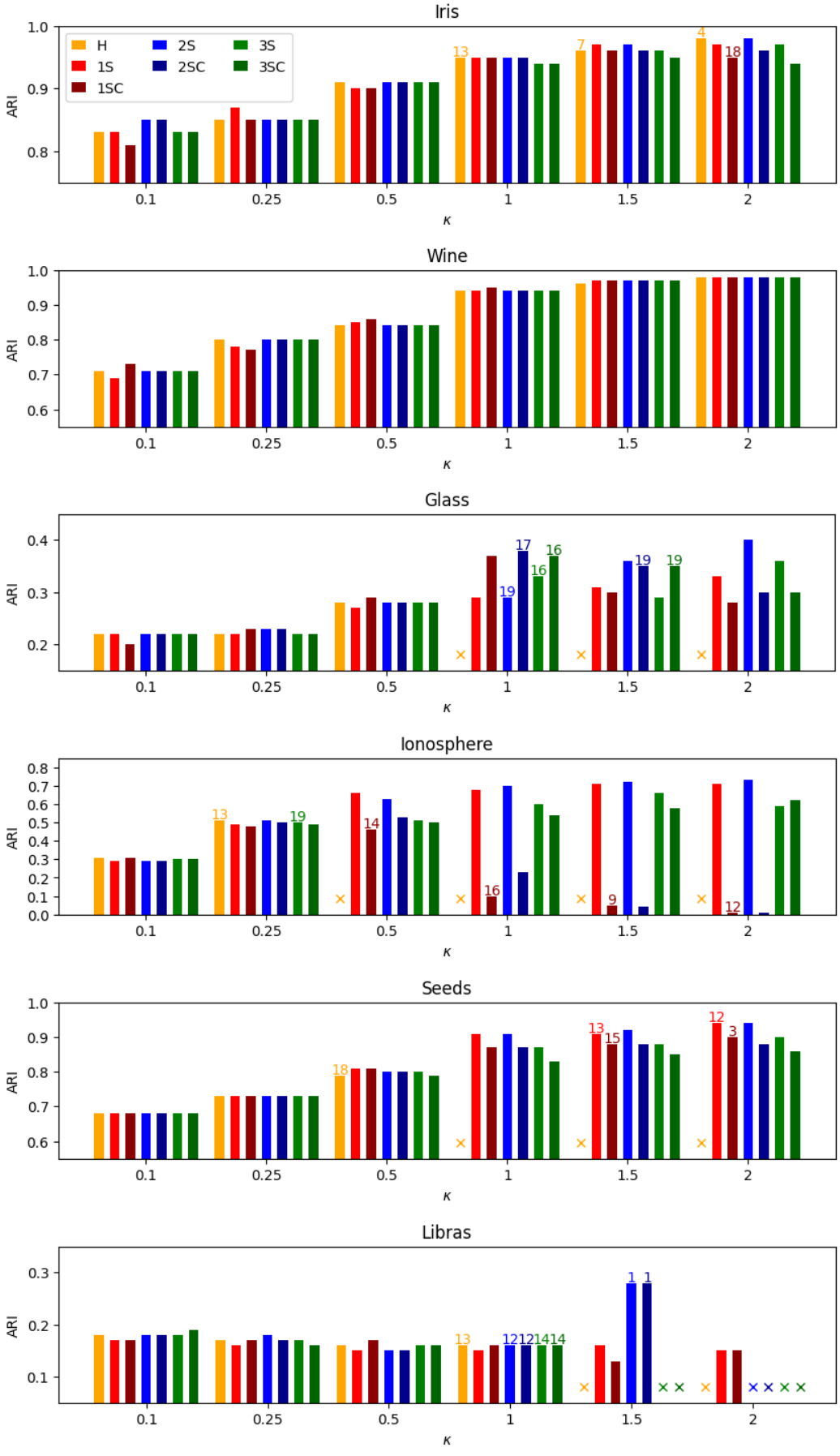
Table 5.8: Tree clustering ([MD, MS] objective with $\epsilon = 0.1$) using chained 2-stage soft constraints method, averaged over 20 runs.

Dataset	κ	ARI	NMI	ML%	CL%	Solver Time (s) S1	S2	S3	Feas.
Iris	0.1	0.83	0.82	100.00	100.00	0.25	0.17	0.73	20
	0.25	0.85	0.84	100.00	100.00	0.24	0.17	0.65	20
	0.5	0.91	0.89	100.00	100.00	0.25	0.20	0.67	20
	1	0.94	0.93	98.98	100.00	0.32	0.36	0.72	20
	1.5	0.96	0.95	98.75	99.94	0.42	0.39	0.62	20
	2	0.97	0.95	98.48	99.77	0.91	0.42	0.57	20
Wine	0.1	0.71	0.70	100.00	100.00	0.40	0.30	1.20	20
	0.25	0.80	0.78	100.00	100.00	0.39	0.47	2.14	20
	0.5	0.84	0.81	100.00	100.00	0.67	0.96	4.08	20
	1	0.94	0.92	100.00	100.00	1.46	1.67	3.95	20
	1.5	0.97	0.95	100.00	100.00	2.05	1.49	1.22	20
	2	0.98	0.97	100.00	100.00	2.48	1.56	1.04	20
Glass	0.1	0.22	0.33	100.00	100.00	1.86	1.86	10.51	20
	0.25	0.22	0.31	100.00	100.00	1.66	2.18	76.23	20
	0.5	0.28	0.35	100.00	100.00	1.62	39.16	1097.62	20
	1	0.33	0.38	79.40	100.00	3.05	1800.45	1490.18	16
	1.5	0.29	0.36	65.00	100.00	8.91	1800.47	498.11	20
	2	0.36	0.41	58.28	100.00	34.43	1800.47	207.79	20
Ionosphere	0.1	0.30	0.24	100.00	100.00	4.59	7.69	32.02	20
	0.25	0.50	0.38	99.13	100.00	8.56	617.02	676.74	19
	0.5	0.51	0.40	82.42	98.50	794.13	1306.95	133.52	20
	1	0.60	0.47	82.62	91.53	1800.72	186.86	18.14	20
	1.5	0.66	0.54	84.67	90.32	1800.72	30.15	13.31	20
	2	0.59	0.47	80.42	85.58	1800.72	24.68	15.61	20
Seeds	0.1	0.68	0.66	100.00	100.00	0.33	0.25	0.93	20
	0.25	0.73	0.71	100.00	100.00	0.34	0.32	1.30	20
	0.5	0.80	0.76	99.54	100.00	0.49	1.45	1.97	20
	1	0.87	0.83	94.88	99.87	4.59	6.10	1.69	20
	1.5	0.88	0.85	93.66	99.42	152.91	5.91	1.11	20
	2	0.90	0.86	93.65	99.26	318.36	4.05	0.95	20
Libras	0.1	0.18	0.44	100.00	100.00	36.51	33.87	1402.16	20
	0.25	0.17	0.42	100.00	100.00	40.08	45.87	1138.11	20
	0.5	0.16	0.41	100.00	100.00	54.42	85.63	782.67	20
	1	0.16	0.40	100.00	100.00	70.06	232.78	1670.91	14
	1.5	-	-	-	-	76.25	1002.12	1806.71	0
	2	-	-	-	-	86.58	1748.72	1806.74	0
Spam	0.1	0.36	0.27	69.25	82.52	1843.08	1053.66	698.94	20
	0.25	0.46	0.36	73.64	78.17	1843.09	594.76	440.64	20
	0.5	0.35	0.60	68.57	70.03	1843.13	542.63	376.67	20
	1	0.35	0.26	68.05	67.99	1843.11	374.44	172.61	20
	1.5	0.36	0.28	68.50	68.50	1842.94	283.34	157.66	20
	2	0.34	0.26	67.96	66.78	1843.19	324.20	147.31	20
Lsun	0.1	0.90	0.91	100.00	100.00	1.31	1.20	1.70	20
	0.25	1.00	1.00	100.00	100.00	0.77	1.04	1.34	20
	0.5	1.00	1.00	100.00	100.00	0.67	0.55	1.31	20
	1	1.00	1.00	100.00	100.00	0.62	0.72	1.09	20
	1.5	1.00	1.00	100.00	100.00	0.62	0.55	0.92	20
	2	1.00	1.00	100.00	100.00	0.62	0.58	0.91	20
Chainlink	0.1	0.86	0.79	99.92	100.00	1.82	1.05	5.17	20
	0.25	0.91	0.84	97.77	99.78	3.80	2.78	3.63	20
	0.5	0.93	0.87	96.97	99.26	11.12	1.80	2.51	20
	1	0.94	0.90	97.21	98.76	42.97	1.21	1.94	20
	1.5	0.95	0.91	97.71	98.52	131.85	0.81	1.90	20
	2	0.95	0.91	97.69	98.41	262.17	0.75	1.88	20
Target	0.1	1.00	1.00	100.00	100.00	6.49	6.66	5.42	20
	0.25	1.00	1.00	100.00	100.00	8.55	5.55	5.58	20
	0.5	1.00	1.00	100.00	100.00	7.47	5.78	7.81	20
	1	1.00	1.00	100.00	100.00	5.78	4.21	6.26	20
	1.5	1.00	1.00	100.00	100.00	4.82	3.20	5.53	20
	2	1.00	1.00	100.00	100.00	4.67	3.14	6.37	20
Wingnut	0.1	1.00	1.00	100.00	100.00	2.02	0.46	1.96	20
	0.25	1.00	1.00	100.00	100.00	2.03	0.46	1.82	20
	0.5	1.00	1.00	100.00	100.00	2.02	0.47	1.81	20
	1	1.00	1.00	100.00	100.00	2.01	0.46	1.72	20
	1.5	1.00	1.00	100.00	100.00	2.00	0.46	1.70	20
	2	1.00	1.00	100.00	100.00	1.99	0.47	1.66	20

Table 5.9: Tree clustering ($[\mathrm{MD}, \mathrm{MS}]$ objective with $\epsilon = 0.1$) using non-chained 3-stage soft constraints approach, averaged over 20 runs.

Dataset	κ	ARI	NMI	ML%	CL%	Solver Time (s) S1	S2	S3	Feas.
Iris	0.1	0.83	0.82	100.00	100.00	0.24	0.17	0.73	20
	0.25	0.85	0.84	100.00	100.00	0.23	0.17	0.67	20
	0.5	0.91	0.89	100.00	100.00	0.24	0.22	0.67	20
	1	0.94	0.92	93.30	100.00	0.28	0.39	0.74	20
	1.5	0.95	0.93	72.11	99.94	0.37	0.59	0.84	20
	2	0.94	0.93	65.41	98.60	0.86	0.59	0.85	20
Wine	0.1	0.71	0.70	100.00	100.00	0.39	0.30	1.19	20
	0.25	0.80	0.78	100.00	100.00	0.39	0.46	2.13	20
	0.5	0.84	0.81	100.00	100.00	0.54	0.84	4.09	20
	1	0.94	0.92	100.00	100.00	1.35	1.29	3.96	20
	1.5	0.97	0.95	100.00	100.00	1.41	1.54	1.23	20
	2	0.98	0.97	100.00	100.00	2.00	1.83	1.04	20
Glass	0.1	0.22	0.33	100.00	100.00	1.46	1.74	10.49	20
	0.25	0.22	0.31	100.00	100.00	1.62	2.30	76.26	20
	0.5	0.28	0.35	100.00	100.00	1.83	32.55	1098.78	20
	1	0.37	0.39	71.62	100.00	2.90	1517.88	1702.93	16
	1.5	0.35	0.40	42.40	100.00	9.62	981.22	1292.64	19
	2	0.30	0.36	25.99	100.00	31.25	579.55	980.35	20
Ionosphere	0.1	0.30	0.24	100.00	100.00	6.28	6.33	31.93	20
	0.25	0.49	0.38	95.23	100.00	6.49	596.21	601.24	20
	0.5	0.50	0.38	17.98	97.27	541.35	1100.41	705.40	20
	1	0.54	0.42	10.47	59.36	578.95	697.71	463.26	20
	1.5	0.58	0.46	17.56	44.25	411.81	324.54	172.11	20
	2	0.62	0.51	17.16	36.89	200.74	156.91	107.95	20
Seeds	0.1	0.68	0.66	100.00	100.00	0.32	0.25	0.93	20
	0.25	0.73	0.71	100.00	100.00	0.34	0.34	1.30	20
	0.5	0.79	0.76	98.28	100.00	0.45	1.11	2.26	20
	1	0.83	0.80	54.27	99.63	4.07	7.78	5.85	20
	1.5	0.85	0.81	34.88	97.56	8.23	7.23	2.93	20
	2	0.86	0.82	35.06	93.94	11.09	4.70	3.39	20
Libras	0.1	0.19	0.45	100.00	100.00	36.33	31.71	1402.26	20
	0.25	0.16	0.41	100.00	100.00	39.86	43.97	1137.15	20
	0.5	0.16	0.41	100.00	100.00	47.15	85.99	781.24	20
	1	0.16	0.40	100.00	100.00	75.95	213.93	1671.21	14
	1.5	-	-	-	-	80.64	910.66	1806.72	0
	2	-	-	-	-	90.70	1759.69	1807.01	0
Spam	0.1	0.27	0.22	1.92	31.48	1170.52	1382.18	1704.55	18
	0.25	0.18	0.16	0.78	11.58	852.62	934.08	1266.59	19
	0.5	0.16	0.15	0.47	5.94	621.62	832.46	1049.19	20
	1	0.10	0.11	0.39	2.95	688.62	725.38	871.57	20
	1.5	0.14	0.15	0.26	1.96	690.26	696.10	700.57	20
	2	0.11	0.14	0.23	1.58	661.59	794.83	644.36	20
Lsun	0.1	0.90	0.91	100.00	100.00	1.11	0.73	1.67	20
	0.25	1.00	1.00	100.00	100.00	0.59	0.58	1.34	20
	0.5	1.00	1.00	100.00	100.00	0.60	0.62	1.30	20
	1	1.00	1.00	100.00	100.00	0.70	0.77	1.07	20
	1.5	1.00	1.00	100.00	100.00	0.59	0.70	0.91	20
	2	1.00	1.00	100.00	100.00	0.61	0.59	0.90	20
Chainlink	0.1	0.86	0.79	98.79	100.00	1.80	1.09	5.17	20
	0.25	0.89	0.81	38.33	99.04	2.70	3.91	7.76	20
	0.5	0.92	0.86	20.89	91.93	4.37	3.32	8.46	20
	1	0.91	0.86	12.27	78.53	4.28	2.35	7.31	20
	1.5	0.93	0.88	11.11	73.90	3.13	1.78	5.00	20
	2	0.93	0.88	9.20	73.04	2.77	1.77	5.32	20
Target	0.1	1.00	1.00	100.00	100.00	7.10	7.85	5.39	20
	0.25	1.00	1.00	100.00	100.00	7.37	5.47	5.68	20
	0.5	1.00	1.00	100.00	100.00	6.24	5.88	7.88	20
	1	1.00	1.00	100.00	100.00	5.29	4.20	6.28	20
	1.5	1.00	1.00	100.00	100.00	4.58	3.80	5.41	20
	2	1.00	1.00	100.00	100.00	4.51	3.67	6.36	20
Wingnut	0.1	1.00	1.00	100.00	100.00	2.03	0.47	1.96	20
	0.25	1.00	1.00	100.00	100.00	2.03	0.49	1.81	20
	0.5	1.00	1.00	100.00	100.00	2.02	0.52	1.78	20
	1	1.00	1.00	100.00	100.00	2.01	0.57	1.72	20
	1.5	1.00	1.00	100.00	100.00	2.00	0.61	1.68	20
	2	1.00	1.00	100.00	100.00	2.00	0.67	1.66	20

Table 5.10: Tree clustering ($[\mathrm{MD}, \mathrm{MS}]$ objective with $\epsilon = 0.1$) using chained 3-stage soft constraints approach, averaged over 20 runs.


Iris
Wine
Glass
Ionosphere
Seeds
Libras
H
1S
1SC
2S
2SC
3S
3SC
ARI
κ
0.1
0.25
0.5
1
1.5
2

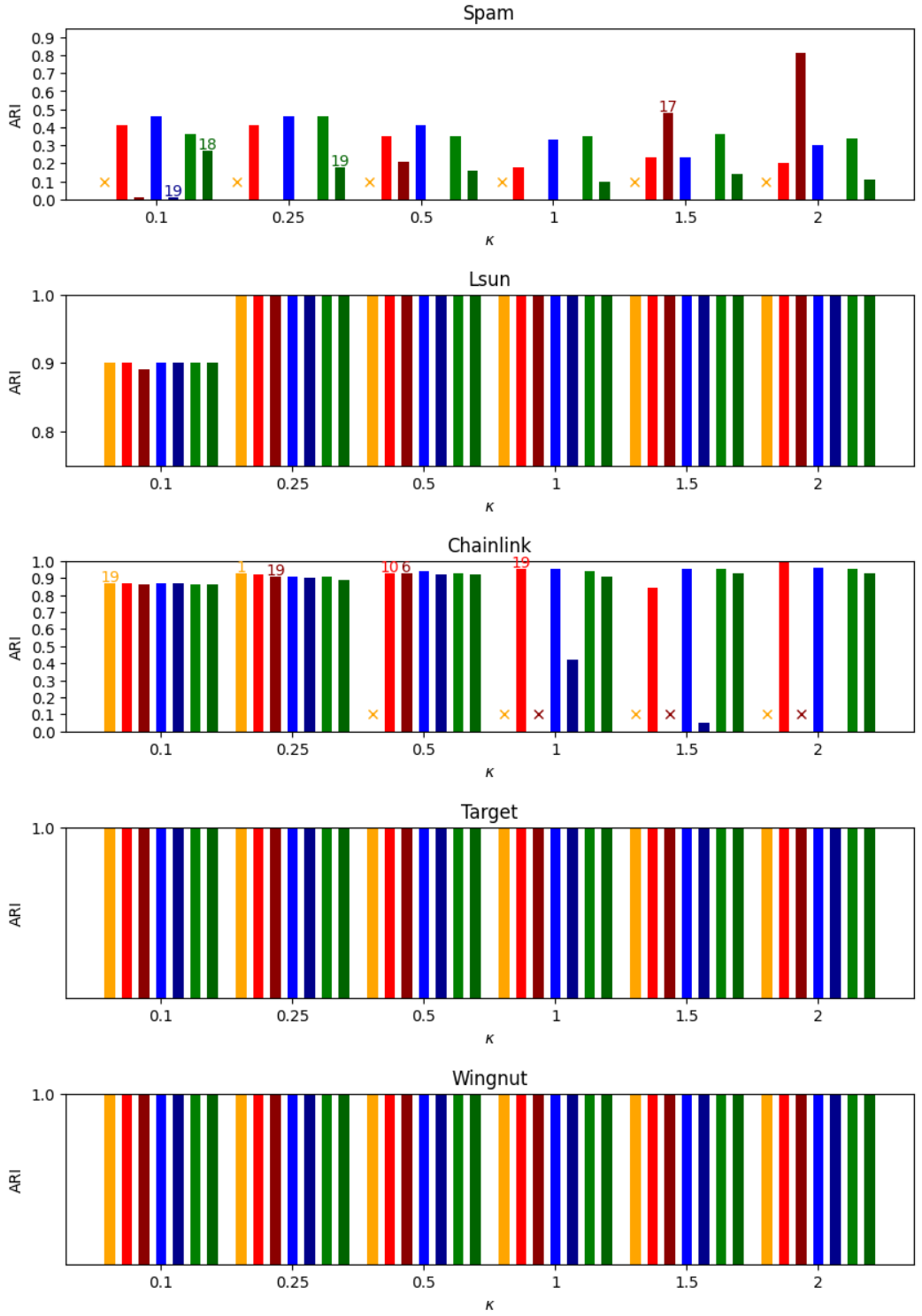


Figure 5.4: Tree clustering ([MD, MS] objective with $\epsilon = 0.1$) using hard constraints (H) and 1-stage (1S), 1-stage chained (1SC), 2-stage (2S), 2-stage chained (2SC), 3-stage (3S), and 3-stage chained (3SC) soft constraint approaches. In-figure numbers indicate the number of feasible solutions (with 20 invisible and 0 as ×).

Dataset	κ	ARI (F)	NMI (F)	Size	Time (s)	Comp./Inc./Inf.	ARI (S)	NMI (S)
Iris	0	0.89	0.87	7	7.71	20 / 0 / 0	0.6	0.67
	0.1	0.87	0.85	1.75	1.98	20 / 0 / 0	0.83	0.82
	0.25	0.87	0.86	2.1	2.41	20 / 0 / 0	0.85	0.84
Wine	0	0.43	0.55	3	4.85	20 / 0 / 0	0	0.02
	0.1	0.71	0.70	1.4	3.14	20 / 0 / 0	0.71	0.7
	0.25	0.77	0.74	1.8	11.22	20 / 0 / 0	0.8	0.78
Glass	0	0.23	0.37	11	39.43	20 / 0 / 0	0.23	0.38
	0.1	0.23	0.35	3.05	47.11	20 / 0 / 0	0.22	0.33
	0.25	0.27	0.35	3.65	533.59	18 / 2 / 0	0.22	0.31
Ionosphere	0	0.10	0.14	3	63.52	20 / 0 / 0	0.01	0.02
	0.1	0.38	0.29	4.3	292.01	20 / 0 / 0	0.31	0.25
	0.25	0.51	0.40	0.7	1356.61	5 / 10 / 5	0.51	0.4
Seeds	0	0.69	0.66	5	8.31	20 / 0 / 0	0.71	0.67
	0.1	0.71	0.69	2.65	4.44	20 / 0 / 0	0.68	0.66
	0.25	0.77	0.74	2.9	7.81	20 / 0 / 0	0.73	0.71
Libras	0	0.17	0.42	0	1808.08	0 / 20 / 0	0.18	0.44
	0.1	0.17	0.43	0.5	1777.20	2 / 18 / 0	0.18	0.44
	0.25	0.16	0.41	0.6	1663.99	4 / 16 / 0	0.17	0.42
Spam	0	0.00	0.01	6	719.16	20 / 0 / 0	0	0
	0.1	-	-	0	234.59	0 / 0 / 20	-	-
	0.25	-	-	0	168.26	0 / 0 / 20	-	-
Lsun	0	1.00	1.00	3	7.52	20 / 0 / 0	0.4	0.51
	0.1	1.00	1.00	4.95	10.42	20 / 0 / 0	0.9	0.91
	0.25	1.00	1.00	3.8	7.89	20 / 0 / 0	1	1
Chainlink	0	0.23	0.18	9	38.37	20 / 0 / 0	0.24	0.19
	0.1	0.88	0.81	1.3	11.65	19 / 0 / 1	0.87	0.8
	0.25	0.93	0.87	0.05	2.45	1 / 0 / 19	0.93	0.87
Target	0	1.00	1.00	9	125.56	20 / 0 / 0	0.33	0.39
	0.1	1.00	1.00	7.65	98.13	20 / 0 / 0	1	1
	0.25	1.00	1.00	4.1	50.14	20 / 0 / 0	1	1
Wingnut	0	1.00	1.00	1	9.02	20 / 0 / 0	1	1
	0.1	1.00	1.00	1	3.65	20 / 0 / 0	1	1
	0.25	1.00	1.00	1	3.47	20 / 0 / 0	1	1

Table 5.11: Tree clustering with the [MD, MS] objective ($\epsilon = 0.1$), best of Pareto front (F) compared against single Pareto optimal solution (S), averaged over 20 runs.

Dataset	ϵ	Pruning	ARI	NMI	Time (s)	# Clauses	Feas./Inf./Unk./Mem.
Iris	0.1	SP	0.90	0.88	0.70	3.49E+04	20 / 0 / 0 / 0
	0.1	-	0.90	0.88	2.13	1.02E+05	20 / 0 / 0 / 0
	0	-	0.91	0.89	199.24	1.52E+05	20 / 0 / 0 / 0
Wine	0.1	SP	0.85	0.82	4.42	4.81E+04	20 / 0 / 0 / 0
	0.1	-	0.85	0.82	6.27	1.53E+05	20 / 0 / 0 / 0
	0	-	0.86	0.83	149.16	2.25E+05	20 / 0 / 0 / 0
Glass	0.1	SP	0.29	0.36	1166.21	1.25E+05	20 / 0 / 0 / 0
	0.1	-	0.26	0.32	1290.41	5.45E+05	20 / 0 / 0 / 0
	0	-	0.24	0.32	1414.21	6.52E+05	20 / 0 / 0 / 0
Ionosphere	0.1	SP	-	-	59.19	1.51E+05	0 / 20 / 0 / 0
	0.1	-	-	-	63.59	3.95E+05	0 / 20 / 0 / 0
	0	-	-	-	312.83	6.84E+05	0 / 20 / 0 / 0
Seeds	0.1	SP	0.81	0.77	1.91	5.13E+04	18 / 2 / 0 / 0
	0.1	-	0.80	0.77	3.86	1.88E+05	18 / 2 / 0 / 0
	0	-	0.80	0.77	300.76	2.89E+05	18 / 2 / 0 / 0
Libras	0.1	SP	0.16	0.41	825.30	2.09E+06	20 / 0 / 0 / 0
	0.1	-	0.15	0.39	945.47	4.77E+06	20 / 0 / 0 / 0
	0	-	0.10	0.32	1460.35	5.07E+06	9 / 0 / 0 / 11
Spam	0.1	SP	-	-	143.34	3.83E+06	0 / 20 / 0 / 0
	0.1	-	-	-	343.13	4.61E+07	0 / 20 / 0 / 0
	0	-	-	-	2698.46	9.90E+07	0 / 5 / 15 / 0
Lsun	0.1	SP	1.00	1.00	1.48	5.08E+04	20 / 0 / 0 / 0
	0.1	-	1.00	1.00	6.13	5.96E+05	20 / 0 / 0 / 0
	0	-	0.96	0.93	1686.21	9.90E+05	14 / 0 / 0 / 6
Chainlink	0.1	SP	-	-	3.26	8.50E+04	0 / 20 / 0 / 0
	0.1	-	-	-	17.13	2.08E+06	0 / 20 / 0 / 0
	0	-	-	-	92.84	4.57E+06	0 / 20 / 0 / 0
Target	0.1	SP	1.00	1.00	8.86	2.61E+05	20 / 0 / 0 / 0
	0.1	-	1.00	1.00	61.87	4.96E+06	20 / 0 / 0 / 0
	0	-	0.96	0.93	1843.70	6.43E+06	20 / 0 / 0 / 0
Wingnut	0.1	SP	1.00	1.00	3.32	9.41E+04	20 / 0 / 0 / 0
	0.1	-	1.00	1.00	18.84	2.14E+06	20 / 0 / 0 / 0
	0	-	1.00	1.00	1679.13	4.71E+06	20 / 0 / 0 / 0

Table 5.12: Ablation study ($[\mathrm{MD}, \mathrm{MS}]$ objective with $\kappa = 0.5$) averaged over 20 runs.

Chapter 6

Neural Sequence Generation with Requirements

6.1 Introduction

In this chapter, we present a framework to solve combinatorial optimization problems by combining pre-trained neural sequence models with CSPs formulating the hard constraint aspects of the problem. Our approach enables constrained ML by guaranteeing satisfaction of constraints for the solutions that are generated by the neural model. Unlike previous chapters, the declarative combinatorial optimization problem is not responsible for formulating the problem of learning a model as a whole. Instead, it is used to represent the CSP aspect of a problem that is primarily solved by a pre-trained model with no support for user-specified constraints. Specifically, the training duties are delegated to deep learning, while the combinatorial component is used to control the output of the deep model.

Neural sequence models, including recurrent neural networks [198] and transformers [278], have been successful in solving combinatorial optimization problems [29, 162]. While typically applied to natural language tasks, neural sequence models can be trained to generate structured solutions to combinatorial problems either via supervised learning [282] or more recently via reinforcement learning [170, 162]. Such models are necessarily deep and complex to be able to make challenging combinatorial decisions. However, their output cannot be reliably expected to conform to specifications, i.e., satisfy hard constraints. This causes issues for when operational applicability and involvement of human expertise is required. These models can also produce suboptimal solutions, which is of concern in applications with a significant emphasis on any slight degradation of the objective value.

A neural sequence model is commonly decoded through iterative next token prediction, generating the solution one token at a time [264, 23]. A beam search procedure can further expand iterative production to generate a set of solutions rather than one [87, 61]. Beam search works well with most sequence generation tasks. Combinatorial tasks, however, of-

ten involve strict requirements that are not guaranteed to be satisfied by approaches that purely optimize predictive scores such as beam search [63]. A requirement on the neural sequential generation output can often be formulated as a set of declarative constraints in a CSP [49]. Depending on the application, requirements may reflect safety or liveness constraints, guardrails to ensure appropriate behavior, customizing constraints that reflect laws in different jurisdictions, or they may include physical or temporal constraints that capture properties of the domain. In the context of solving combinatorial optimization problems, the requirement is simply the collective hard constraints of the problem. Constraining ML solutions to those that adhere to requirements increases their applicability in cases where rigorous reasoning is demanded to prune unwanted behavior, as the user can formulate and enforce their expectation of the output. Satisfying such requirements usually needs a global perspective and meticulous reasoning and thus does not suit a statistical iterative approach to generation, even if done in large quantities.

We present a modular framework to combine neural sequence generation with a CSP requirement. Our approach can guarantee requirement satisfaction when applied to any pre-trained model with no user-specified constraint support. Although a new model can be trained or fine-tuned towards satisfying requirements while optimizing the objective, such approach is time-consuming, requires a large amount of specific data for each singular or combination of requirements, and is not guaranteed to satisfy the requirement for each generated solution. By utilizing the formulation approach to solving combinatorial problems, our approach encodes requirements that can be reused, tuned, and combined with one another without re-training the model. For the purposes of this chapter, we focus on Vehicle Routing Problems (VRP) as a case study for our approach. We chose VRPs as they are important combinatorial optimization problems where a wide variety of constraints are required in their numerous variants. Neural sequence models have been successful in solving VRPs [162] but only integrate myopic constraints that do not require advanced reasoning. We solve new VRP variants by combining the prediction model of a simpler variant with requirements encoded as CSPs.

The contributions of this chapter are as follows:

1. We introduce *beam search with cuts* (BSC) to solve neural sequence generation when solutions must adhere to a requirement. Cuts are decided for each partial solution in the search and prevent the expansion of those that are incapable of satisfying the requirement. BSC is guaranteed to generate a solution satisfying the requirement, if one exists.

2. We encode requirements as constraint satisfaction problems. We introduce bin packing and regular language acceptance as two types of requirements that find application in multiple settings. We further show how they can be implemented as cuts by solving the feasibility problem with adherence to a given partial solution using IP or SAT.

3. We combine an existing pre-trained VRP neural model with our two requirement types. The model's support for two simpler VRP variants combined with requirements allows

us to solve three VRP variants that are more complex.

4. We experimentally show that beam search with cuts enables satisfaction of a requirement without significant cost to quality, in a short runtime, and without requiring large quantities of solutions. Moreover, we show that cuts allow us to significantly tighten requirements while maintaining solution quality. The iterative tightening depicts a trade-off until solution quality becomes unstable and, ultimately, infeasibility is reached. Furthermore, we show that beam search with cuts scales exponentially better when increasing the solution length or the requirement strength. Namely, standard beam search exhibits an exponential increase in the number of solutions that it should generate to satisfy the requirement, while BSC's performance remains stable. Lastly, we show that an incremental solving of the CSP instances significantly lowers the time spent by the solver, improving the overall runtime of BSC.

While our contributions are conveyed and evaluated in the context of VRPs (an important application area), our methodology for imposing requirements on solutions is general and can be applied to a variety of neural models and requirements to solve problems in different settings.

Our beam search with cuts work first appeared in the proceedings of the Seventeenth International Symposium on Combinatorial Search [249]. The content of this chapter mirrors our paper, with slight modifications, further elaboration, and a deeper analysis on some of the discussion points and results.

6.2 Preliminaries

In this section, we provide preliminary information on the problem of neural sequence generation and how it is typically solved. A neural sequence model generates solutions using a given alphabet referred to as the tokens, denoted by Σ. The set Σ^* contains all finite sequences of tokens and the set $\mathcal{P}(\Sigma)$ contains all probability distributions over tokens. In Section 6.2.1, we define the next token prediction function. In Section 6.2.2, we describe the beam search procedure as a decoder for generating sequences. Finally, in Section 6.2.3, we describe sequence generation with requirements and define constrained decoders.

6.2.1 Next Token Prediction

In this section, we define the next token prediction function and explain how it gives rise to a predictive score for a sequence. The task of generating a solution sequentially is often carried out by a local endeavor that selects and appends the next token to an existing sequence called the partial solution [118]. The local expansion is guided by a function that predicts the appearance of the next token.

Definition 25 (Next Token Prediction Function)**.** *Given a set of tokens Σ and $\mathcal{P}(\Sigma)$ representing the set of all probability distributions over Σ, a next token prediction function*

$p : \Sigma^* \to \mathcal{P}(\Sigma)$ *assigns probabilities to tokens for occurring next after observing a sequence of existing tokens as the partial solution.*

Next token prediction is often accompanied by a filtering function that maintains solution validity by determining the subset of tokens that can appear next at each step. Only solutions that respect the filtering function are considered throughout this chapter. We refer to a solution x as *complete* when a specified termination condition is met.

Definition 26 (Sequence Score)**.** *Given a set of tokens* Σ *and a next token prediction function* p*, the score* $\theta_p(.)$ *of a partial solution* $x = x_1, x_2, ..., x_k \in \Sigma^*$ *is defined as the joint probability of its tokens according to* p*, i.e.,* $\theta_p(x) = \prod_{i=0}^{k-1} p(x_1, ..., x_i)[x_{i+1}]$ *where* $p(x)[x_{i+1}]$ *represents the probability assigned to* x_{i+1} *in the distribution specified by* $p(x)$*.*

6.2.2 Beam Search Decoder

In this section, we describe how the next token prediction function (Definition 25) can be used to decode a neural model and find a sequence as a solution. We further define beam search as a sequential decoder.

Definition 27 (Sequential Decoder)**.** *Given a set of tokens* Σ *and a next token prediction function* p *modelled by a neural network, a sequential decoder* ϕ *aims to find a complete solution* $x \in \Sigma^*$ *with maximum score by iteratively extending partial solutions through next token prediction.*

We hereafter refer to sequential decoders as simply decoders. A decoder ϕ can be implemented by using p to randomly sample or select the maximum probability token at each step to iteratively generate a sequence until a complete solution is found. However, greedily constructing a single solution is highly prone to getting stuck in local minima.

A more comprehensive approach to exploring the solution space involves considering a set of alternatives simultaneously. The beam search method accomplishes this by maintaining a set of partial solutions at each step and considering their highest scoring expansions for the next step [61].

Definition 28 (Beam Search)**.** *Given a set of tokens* Σ *and a next token prediction function* p *modelled by a neural network, a beam search decoder* ϕ^{bs} *of width* w *generates sets of partial solutions* S_i *at iteration* i *with* $i \leq k$*.* S_k *only contains complete solutions,* S_0 *is the singular set containing the empty string* ϵ*, and each* S_i *with* $i > 0$ *contains at most* w *solutions of length* i *and is recursively constructed using the equation* $S_i = argmax_{1:w}(\{\theta_p(x.a) \,|\, x \in S_{i-1}, a \in \Sigma\})$ *where* $argmax_{1:w}$ *selects the members corresponding to the* w *highest scores.*

The pseudo-code for beam search is presented in Algorithm 3. It is important to note that the solution length k is not necessarily predetermined and beam search is terminated when all generated solutions are deemed complete. Once the procedure is finished, the highest-scoring solution from the set S_k is chosen as the output. The sequence score can serve as the primary objective of the problem or as a proxy for another objective it aims to

approximate. In the latter case, the final solution is selected based on the main objective instead.

Algorithm 3 Beam Search

Input: Set of tokens Σ, Next token prediction function p modelled by a neural network
Parameter: Width w
Output: Solution $x \in \Sigma^*$

$S_0 = \{\epsilon\}$ {Add the empty string as the initial solution.}
$i = 0$
while S_i contains non-complete solutions **do**
  $S'_i = \{x.a | x \in S_i, a \in \Sigma\}$ {Consider all allowed expansions after filtering.}
  Sort S'_i based on descending values of θ_p.
  $S_{i+1} = S'_i[1 : w]$ {Keep the w highest scoring solutions for next iteration.}
end while
$S_k = S_i$
return $argmax(\{\theta_p(x) | x \in S_k\})$

6.2.3 Solution Requirements

In this section, we define the notion of solution requirements and constrained decoders. A requirement imposes a constraint on the solution space, and a complete solution that satisfies the requirement is referred to as feasible. A constrained decoder is required to find a solution that is feasible in addition to maximizing the prediction score [13, 63].

Definition 29 (Constrained Sequential Decoder)**.** *Given a set of tokens Σ, a next token prediction function p modelled by a neural network, and a requirement $R \subseteq \Sigma^*$, a constrained sequential decoder ϕ_c aims to find a complete feasible solution $x \in R$ with maximum score by iteratively extending partial solutions through next token prediction.*

A naive approach to building constrained decoders is to utilize beam search as before and exclude any complete solutions that do not satisfy the requirement at the end. However, relying solely on beam search provides no guarantees to find any feasible solutions.

6.3 Vehicle Routing Problems

In this section, we discuss Vehicle Routing Problems (VRP) and how neural sequence models have been used to solve them in previous work. VRPs are a family of optimization problems aimed at navigating one or multiple vehicles through a set of nodes (locations) while potentially interacting with said nodes along the way through pickups, deliveries, or time-sensitive tasks [269]. The goal of a VRP is usually to optimize the cost in terms of different aspects such as distance, duration, or the number of vehicles while respecting capacity, time, or vehicular constraints. We focus our attention on an established variant of the Constrained Vehicle Routing Problem (CVRP) alongside two variants based on the Travelling Salesman Problem (TSP) that we introduce.

Problem 9 (Constrained Vehicle Routing Problem with Max Tours (CVRPM))**.** *Given a finite set of demand nodes as 2D coordinates* $N = \{n_i \,|\, n_i \in \mathbb{R} \times \mathbb{R}\}$*, a demand function* $D : N \to \mathbb{N}$*, a depot node* $n_d \in \mathbb{R} \times \mathbb{R}$*, a capacity* $c \in \mathbb{N}$*, and a maximum number of tours* $m \in \mathbb{N}$*, find a series of tours* T *partitioning the nodes with* $|T| \leq m$ *and* $\forall x \in T : \sum D(x_i) \leq c$ *to minimize the total distance covered starting from the depot* $\sum_{x \in T} dis(n_d.x)$ *where* x_i *is the* i*-th node in* x *and* $dis(x)$ *represents the total distance for traversing* x *from start to finish and back to start.*

Note that our CVRPM definition is commonly referred to as CVRP in the combinatorial optimization literature [271, 18]. However, we refer to CVRP as Problem 9 without a maximum number of tours m. We do so to remain consistent with our baseline and make the distinction between the problem that we solve and theirs [162].

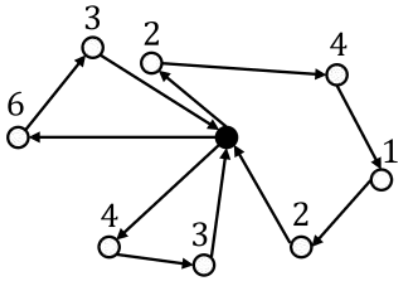
Next, we introduce two new TSP variants. First, we extend TSP with the notion of demands from Problem 9. In this variant, there is no depot but we have multiple supply nodes playing the role of the single depot node from CVRPM. More specifically, they are providers and the vehicle is resupplied upon each visit with items that should be delivered to other nodes. Note that we have multiple supply nodes instead of one depot node and supply nodes are to be visited once but the depot is the starting and ending point of each tour. Visiting all of the nodes once results in a single Hamiltonian cycle as we have in TSP. For the second variant, we extend TSP with a regular language specification. Namely, we remove the sequences of nodes that do not belong to the regular language from consideration. Regular languages are expressive and flexible paradigms that can be used to specify different constraints that are commonly added to TSP such as precedence relations [167].

Problem 10 (TSP with Demands (TSPD))**.** *Given a finite set of nodes as 2D coordinates* $N = \{n_i \,|\, n_i \in \mathbb{R} \times \mathbb{R}\}$*, a subset of supply nodes* $N_d \subseteq N$*, a starting node* $n_0 \in N_d$*, demands for non-supply nodes* $D : N \setminus N_d \to \mathbb{N}$*, and capacity* $c \in \mathbb{N}$*, find a complete permutation of nodes* x *with minimum total distance covered* $dis(x)$ *such that* x *starts with* n_0 *and the sum of demands in any continuous sequence of non-supply nodes does not exceed the capacity.*

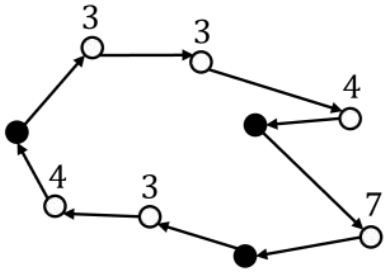
Note that having a starting node n_0 is not crucial to the problem since a TSP tour can start and end in any node. However, due to practicality of sequential generation in our baseline and our own approach, we assume the solution to start with a supply node referred to as the start.

Problem 11 (TSP with Regular Specification (TSPR))**.** *Given a finite set of nodes as 2D coordinates* $N = \{n_i \,|\, n_i \in \mathbb{R} \times \mathbb{R}\}$*, a starting node* $n_0 \in N$*, an alphabet mapping* $\sigma : N \to \Sigma_{\mathcal{A}}$*, and a regular language* $\mathcal{A}$ *of alphabet* $\Sigma_{\mathcal{A}}$*, find a complete permutation of nodes* x *with minimum total distance covered* $dis(x)$ *such that* x *starts with* n_0 *and* $\sigma(x)$ *is accepted by* $\mathcal{A}$*, where* $\sigma(x)$ *is shorthand for applying* σ *to each* x_i *and concatenating the results.*

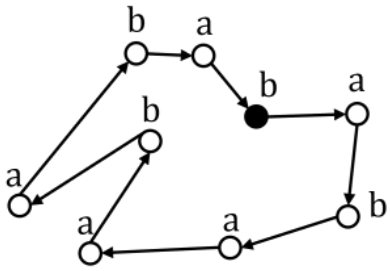
Similar to Problem 10, it is not necessary to consider a starting node n_0 and we do so for practical reasons. Note that if a starting node is not specified, it suffices to have an

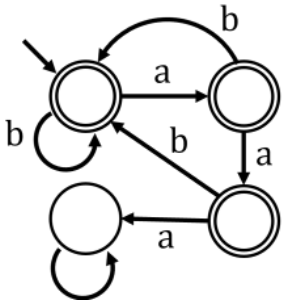


(a) CVRPM (Problem 9). The black node is the depot, the white ones are the demand nodes with their demands specified above them, the capacity is 10, and the maximum number of tours is 3.



(b) TSPD (Problem 10). The black nodes are the supply nodes, the white ones are the demand nodes with their demands specified above them, and the capacity is 10.





(c) TSPR (Problem 11). The black node is the starting node, the alphabet mapping of nodes is specified above them, and the regular language is presented as a DFA. The language is the set of all strings where no 3 consecutive a's appear.

Figure 6.1: Examples of VRP variants with valid solutions.

accepting sequence when starting from any node along the tour. Furthermore, note that a many-to-one alphabet mapping can indicate a division of nodes based on properties such as containing region, type of interaction, or corresponding party. Such mappings can be used to formulate specifications concerning safety, fairness, or other regulations as regular languages. The regular language is assumed to be given as a deterministic finite automata (DFA) [257].

In Figure 6.1, we present visualizations for the three VRP variants on which we focus in this chapter. In these examples, we can see that solutions satisfying the additional constraints, e.g., capacity and regular language ones, are not necessarily optimal in cost.

6.3.1 Solving VRPs with Neural Sequence Models

While traditionally solved by heuristic or exact methods, modern deep learning has also enabled us to solve VRPs [285] through methods including iterative improvement [296, 190] and sequence generation [162, 210]. We use the attention-based model presented in Kool,

Hoof, and Welling [162] both as a non-constrained baseline and as a basis for implementing our constrained sequential decoder.

Kool, Hoof, and Welling utilizes a deep learning model based on attention layers trained using REINFORCE [290]. Their model uses nodes of a VRP instance as tokens and is trained to minimize the total distance covered through next token prediction. A neural-based approach is commonly not competitive against exact combinatorial ones in solution quality. In our experiments, the model in Kool, Hoof, and Welling is able to find solutions whose objective values are typically within 10% to 20% of the best known upper bounds on optimal values. However, neural-based approaches have the added capability of optimizing complicated, data-driven objectives that do not have compact mathematical formulations. Such objectives include those based on historical data, user preference, or prompting.

The algorithm in Kool, Hoof, and Welling guarantees solution validity only with regard to *myopic* constraints. In myopic constraints, any partial solution that does not immediately violate the constraints can be extended to a complete feasible one, i.e., satisfaction of such constraints does not require global reasoning. More specifically, they use a simple function to filter out any tokens that cannot be appended to a partial solution due to violating the constraints.

Kool, Hoof, and Welling demonstrates support for multiple VRP variants including TSP and CVRP. TSP is equivalent to Problem 10 with no demand and capacity or Problem 11 with no alphabet mapping and regular language specification. CVRP is equivalent to Problem 9 with unlimited number of tours. Note that solving these two VRP variants with sequence generation requires serialization of solutions. Namely, a CVRP solution is seen as a concatenation of tours divided by the token corresponding to the depot and a TSP solution always starts from a specific node even though it is cyclical. The filtering function used for TSP guarantees the validity of the solution by excluding the nodes that are already visited. The filtering function used for CVRP guarantees the validity of the solution by excluding the non-depot nodes that are already visited and the nodes whose addition would violate the capacity limit.

6.4 Beam Search with Cuts

In this section, we introduce a variation of the beam search procedure called Beam Search with Cuts (BSC). Unlike ordinary beam search, BSC allows us to implement a constrained decoder (Definition 29) that is not agnostic of the demanded requirement. Instead, BSC performs explicit checks at each step and cuts any partial solutions that cannot be extended to a complete one satisfying the requirement. As a result, a constrained decoder utilizing BSC is guaranteed to produce feasible solutions if they exist. We formally define BSC as a constrained decoder.

Definition 30 (Beam Search with Cuts)**.** *Given a set of tokens Σ, a next token prediction function p modelled by a neural network, and a requirement R, a beam search with cuts*

Algorithm 4 Beam Search with Cuts

Input: Set of tokens Σ, Next token prediction function p modelled by a neural network, Requirement R
Parameter: Width w
Output: Solution $x \in \Sigma^*$ or $\varnothing$

$S_0 = \{\epsilon\}$ {Add the empty string as the initial solution.}
$i = 0$
while S_i contains non-complete solutions **do**
  $S'_i = \{x.a | x \in S_i, a \in \Sigma\}$ {Consider all allowed expansions after filtering.}
  Sort S'_i based on descending values of θ_p.
  $S_{i+1} = \emptyset$
  for all $x \in S'_i$ **do**
    $feas = \exists x' : x.x' \in R \wedge x.x'$is complete {Check feasibility via solving the CSP.}
    if $feas$ **then**
      $S_{i+1} = S_{i+1} \cup \{x\}$
    end if
    if $|S_{i+1}| == w$ **then**
      Break
    end if
  end for
end while
$S_k = S_i$
if $|S_k| == 0$ **then**
  return $\varnothing$
else
  return $argmax(\{\theta_p(x) | x \in S_k\})$
end if

decoder ϕ^{bsc} of width w is a constrained decoder that generates sets of partial solutions S_i at iteration i with $i \leq k$. S_k only contains complete solutions, S_0 is the singular set containing the empty string ϵ and each S_i with $i > 0$ contains at most w solutions of length i and is recursively constructed using the following equation:

$$S_i = argmax_{1:w}(\{\theta_p(x.a) \,|\, x \in S_{i-1}, a \in \Sigma, \\ \exists x' : x.a.x' \in R \wedge [x.a.x' \text{is complete}]\})$$

where $argmax_{1:w}$ selects the members corresponding to the w highest scores. An expansion $x.a$ is said to be cut *if it is removed from contention due to not satisfying $\exists x' : x.a.x' \in R \wedge [x.a.x' \text{is complete}]$.*

In the combinatorial optimization literature, cuts commonly refer to (linear) constraints that are added to the instance to remove infeasible or undesirable solutions from consideration. For example, Benders cuts [101] remove solutions that make the subsequent stages infeasible and Gomory cuts [103] remove non-integer solutions to relaxations of integer problems. Canonical cuts [25], also commonly referred to as no-good cuts, are most similar to the cuts employed in BSC. They enforce a positive distance to a specific solution to remove

it from contention. Despite the similarity, the cuts in BSC are not no-good cuts as they do not add constraints to subsequent attempts at finding solutions.

Algorithm 4 presents the pseudo-code for beam search with cuts. Underlined parts are additions that do not exist in the pseudo-code for ordinary beam search (Definition 28). Note that cuts are decided at line 8. If, at any iteration, all partial solutions are cut, the condition at line 18 is triggered and no solution ($\varnothing$) is returned. In the following, we elaborate on and analyze the algorithm in multiple aspects:

- **CSP cuts:** Since we use CSPs to specify solution requirements, we can implement a BSC decoder by extending beam search with calls to a CSP solver acting as cuts. Specifically, at each step we check partial solutions in decreasing order of score (line 5), only add them to the next S_i (line 10) if they can be extended to a feasible solution (i.e., they are not cut) (line 8), and stop when w solutions are added (line 12). We check whether a partial solution needs to be cut by solving the CSP instance with the additional constraint that the assignment must match our current partial solution up to the point that it has been decoded. Note that the CSP instance needs to validate that the extended solution is also complete (i.e., $x.x'$is complete) in addition to belonging to R. Thus, it needs to encode the termination condition and the filtering function (representing myopic constraints) as well as the requirement.

- **Incremental CSP Solving:** Given a solution length of k and width of w, at least kw cut decisions are made in a successful run of beam search with cuts. Additionally, all of the calls to the CSP solver share much in common as they are solving the same requirement conditioned on different partial solutions. The multitude and repetitive nature of the CSP calls makes our approach amenable to employing incremental solving, a family of techniques aimed at improving the performance through communication between successive calls. Incremental solving enables storing learned information that will be useful in detection of both feasible and infeasible instances in the future. Previous work has utilized incremental CSP solving for pruning infeasible partial solutions in problems such as preferential subset selection [48, 43]. In Section 6.6, we demonstrate how incremental solving can be applied to our SAT-based requirements.

- **Timeouts:** Deciding a cut via solving a CSP instance is inherently difficult and can potentially take a long time. Thus, we use a timeout limit for line 8 to disrupt the solver if it does not manage to find a feasible solution in time. A partial solution leading to a timeout is still cut ($feas = false$) as its feasibility is unclear and it has not been shown to be extendable to a complete feasible solution.

- **Completeness:** A BSC decoder employing timeouts is not guaranteed to produce a feasible solution, even if one exists. This is because cuts due to timeouts might remove all of the partial solutions that can be extended to complete feasible ones. Thus, if

there is a timeout in the decoding process, lack of feasible solutions (returning $\varnothing$) cannot be interpreted as infeasibility.

Contrarily, if no timeouts occur or timeouts are not employed at all, a BSC decoder is guaranteed to produce feasible solutions if any exists. Equivalently, we can state that given enough time, the BSC decoder will find solutions or correctly declare infeasibility. These statements are equivalent since timeouts are principally avoidable by dedicating more time to each CSP call.

Proposition 8 (BSC Completeness)**.** *For any width w and CSP requirement R, a beam search with cuts decoder ϕ^{bsc} that is armed with the CSP solver for R given enough time, is guaranteed to find a complete and feasible solution $x \in R$ if one exists, or to declare infeasibility if not.*

Note that Algorithm 4 can be implemented with any CSP encoding a requirement on a sequential solution. In the next two sections, we introduce two types of requirements and implement them as CSPs to be inserted into the BSC algorithm.

6.5 Bin Packing Requirement

In this section, we propose bin packing as our first type of requirement. The aim of the bin packing problem is to partition items into a limited number of subsets while satisfying a capacity constraint.

Requirement 1 (Bin Packing)**.** *Given a set of items I, their weights $W : I \to \mathbb{N}$, a bin capacity $c \in \mathbb{N}$, and maximum number of bins $m \in \mathbb{N}$, a bin packing requirement R_{bin} can be satisfied by a partition $B = \{B_1, B_2, ..., B_m\}$ of items such that each bin respects the capacity constraint $\forall B_i : \sum_{i \in B_i} W(i) \leq c$.*

Bin packing-based requirements can be utilized to solve sequence generation tasks that involve partitioning elements into subgroups. Such tasks include multiple VRP variants, scheduling problems, or spatial reasoning challenges.

6.5.1 Application as Cuts

To define feasibility and implement cuts, we need to provide a way in which a sequential solution is interpreted with regard to a requirement. For a sequence to be interpreted as a bin packing solution, certain tokens act as *dividers* with others representing items. A bin is then every maximal continuous subsequence of items.

We solve CVRPM (Problem 9) and TSPD (Problem 10) using beam search with cuts employing the bin packing requirement R_{bin}. Note that the requirement is not specifically designed for these two problems, nor for VRP problems in general, and can be applied in a variety of other settings. To solve CVRPM (respectively, TSPD), we utilize the pre-trained CVRP (respectively, TSP) model from the baseline using next token prediction

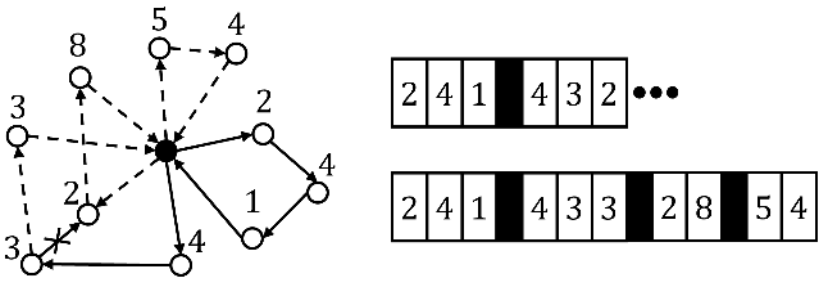


Figure 6.2: An instance of the CVRPM problem (left) with an infeasible partial solution (top right) and a complete feasible solution (bottom right). Dashed lines appear only in the complete solution, the crossed out line appears only in the partial solution, and solid lines belong to both. Tokens in the solution are marked with their corresponding demand and black tokens represent dividers.

function p_{cvrp} (p_{tsp}). To solve CVRPM, the depot node is the divider token, effectively equating nodes to items, tours to bins, and demands to weights, with capacities playing the same role. To solve TSPD, the supply nodes are the divider tokens, effectively equating non-supply nodes to items, non-supply segments to bins, and demands to weights, with capacities playing the same role again. Note that a valid solution to CVRP (respectively, TSP) that satisfies the bin packing requirement is a valid solution to CVRPM (respectively, TSPD). Moreover, in both cases, the objective is optimizing the total distance which remains the same between the constrained and non-constrained variants. Thus, BSC is effectively solving the CVRPM and TSPD problems by combining the bin packing requirement with CVRP and TSP support from the baseline.

Figure 6.2 depicts an example of solving the CVRPM problem alongside a partial infeasible solution and a complete feasible solution. The partial solution is cut since there is no feasible partition of items in the bin packing requirement conditioned on the existing assignments. Note that the first segment (bin) in the solution cannot accept any more nodes (items) since it is sealed off with a divider. The complete solution matches the partial solution except for its last token, showing that slightly steering the generation process using cuts can address infeasibility.

6.5.2 IP-based Encoding

To implement a bin packing requirement R_{bin} as a cut in BSC, we need to solve the feasibility problem for the bin packing requirement given a partial solution. A partial solution, in CVRPM or TSPD, dictates fixed assignment of a subset of items to bins and a subset of bins to be closed off from accepting more items. When the partial solution has already generated t^F divider tokens (tours in CVRPM or visits to supply nodes in TSPD), there are fixed assignments for bins $[1, t^F + 1]$ and bins $[1, t^F]$ have been closed off. We denote the bins with fixed assignments as $B^F_1, B^F_2, ..., B^F_{t^F+1}$. Note that these bins can potentially be empty. In particular, $B^F_{t^F+1}$ can be empty because its content is yet to be generated.

We present the following IP encoding to decide feasibility of the bin packing requirement with the added constraint of conforming to a partial solution. We use binary variables $\{a_{i,j} \mid i \in I, 1 \leq j \leq m\}$ to represent that item i is assigned to bin j. Constraints in Eq. (6.1) enforce that each item is assigned to exactly one bin. Constraints in Eq. (6.2) enforce that the sum of weights for items assigned to a bin is less than the capacity. Constraints in Eq. (6.3) enforce the bin packing to match the given fixed assignments based on the partial solution. Lastly, constraints in Eq. (6.4) enforce that the items yet to appear in the partial solution, are not assigned to the bins that have been closed off.

$$\forall i \in I : \quad \sum_j a_{i,j} = 1 \tag{6.1}$$

$$\forall j \in [1, m] : \quad \sum_i a_{i,j} W(i) \leq c \tag{6.2}$$

$$\forall j \leq t^F + 1, i \in B_j^F : \quad a_{i,j} = 1 \tag{6.3}$$

$$\forall j \leq t^F, i \notin \bigcup_{j \leq t^F + 1} B_t^F : \quad a_{i,j} = 0 \tag{6.4}$$

6.6 Regular Language Requirement

In this section, we propose regular languages as our second type of requirement. Regular languages are a family of simple formal languages where finite memory suffices for their detection [257]. Despite their simplicity, they have crucial applications in multiple fields such as formal verification, synthesis, and planning. They enable the support of a wide range of specifications including safety, reachability, and grammatical ones. Moreover, modal logic paradigms such as finite linear temporal logic (LTL-f) can be translated to regular languages [73]. In Section 6.8.2, Section 6.8.3, and Section 6.8.4, we use several concrete regular language requirements to solve TSPR (Problem 11).

Any regular language can be represented by a Deterministic Finite Automata (DFA) and vice-versa. Thus, a regular language requirement is equivalent to requiring the solution to be accepted by the corresponding DFA. We use the symbol $\mathcal{A}$ for both a regular language and the DFA representing that language, interchangeably.

Requirement 2 (Regular Language)**.** *Let $\mathcal{A}$ be a DFA with alphabet $\Sigma_\mathcal{A}$, finite set of states $Q_\mathcal{A}$, initial state $q_0 \in Q_\mathcal{A}$, accept states $Q_\mathcal{A}^F \subseteq Q_\mathcal{A}$, and transition function $\delta_\mathcal{A} : Q_\mathcal{A} \times \Sigma_\mathcal{A} \to Q_\mathcal{A}$, let also $W_\mathcal{A} \subseteq \Sigma_\mathcal{A}^*$ be a set of strings corresponding to complete solutions. A regular language requirement $R_\mathcal{A}$ can be satisfied by $w \in W_\mathcal{A}$ such that $\Delta(q_0, w) \in Q_\mathcal{A}^F$ where $\Delta(.,.)$ is recursively defined as follows:*

$$\Delta(q, w) = \begin{cases} \Delta(\delta(q, a), w') & a \in \Sigma_\mathcal{A}, w = a.w' \\ q & w = \epsilon \end{cases}$$

Note that, as discussed in Section 6.4, a requirement should encode the termination condition and the filtering function of the sequence generation model as well. For Requirement 1, this is implicit in the formulation because all of the items should partake in the bin packing. In Requirement 2, however, we explicitly include this by limiting the set of candidate solutions to $W_{\mathcal{A}}$, representing the set of all complete solutions. Employing CSPs allows us to encode a wide array of termination conditions and filtering functions as the $w \in W_{\mathcal{A}}$ constraint.

6.6.1 Application as Cuts

Similar to the interpretation that we provided in Section 6.5, we describe how a sequential solution is interpreted in the context of Requirement 2. The alphabet mapping function σ is applied to every token in the solution and the resulting string is used as input to be accepted or not by the DFA.

We solve a TSPR (Problem 11) instance with regular language $\mathcal{A}$ using beam search with cuts employing the regular language requirement $R_{\mathcal{A}}$. We use alphabet mapping σ to construct the set of candidate inputs $W_{\mathcal{A}}$ in the requirement $R_{\mathcal{A}}$. Specifically, $W_{\mathcal{A}}$ contains all $\sigma(x)$ sequences where x is a permutation of all nodes in the problem.

Similar to bin packing, Requirement 2 is not particular to this problem and can be utilized to solve a variety of VRP and non-VRP problems. From the baseline, we use the model for solving TSP and the next token prediction function p_{tsp}. Note that a valid solution to TSP that satisfies the regular language requirement (the corresponding string is accepted by the DFA) is a valid solution to TSPR, and the two variants share an optimization objective. Thus, BSC is effectively solving TSPR by combining the regular language requirement with the TSP support from the baseline.

In Figure 6.3, we present a TSPR instance with an infeasible partial solution and a complete feasible solution. The partial solution is cut because it cannot be extended to a complete solution corresponding to a string accepted by the DFA. Note that an arbitrary extension can satisfy the requirement. But since the set of strings are limited to $W_{\mathcal{A}}$, the partial solution can only be extended by *aaa*, resulting in a non-accepting string. Similar to our CVRPM example (Figure 6.2), we see how cutting an infeasible extension of a partial solution can guide it towards a complete feasible one.

6.6.2 Encoding

To implement a regular language requirement $R_{\mathcal{A}}$ as a cut in BSC, we need to solve the DFA acceptance problem for $\mathcal{A}$ conditioned on a partial solution. Adherence to the partial solution dictates that the input to $\mathcal{A}$ should start with a sequence w^F of length l^F that corresponds to the output of applying the alphabet mapping to the partial solution. We use the number of nodes $|N|$ as the length of the input as a complete TSP solution contains all nodes exactly once. To guarantee that the input belongs to $W_{\mathcal{A}}$, for each member of the input alphabet $a \in \Sigma_{\mathcal{A}}$ we count the number of nodes n such that $\sigma(n) = a$ and denote

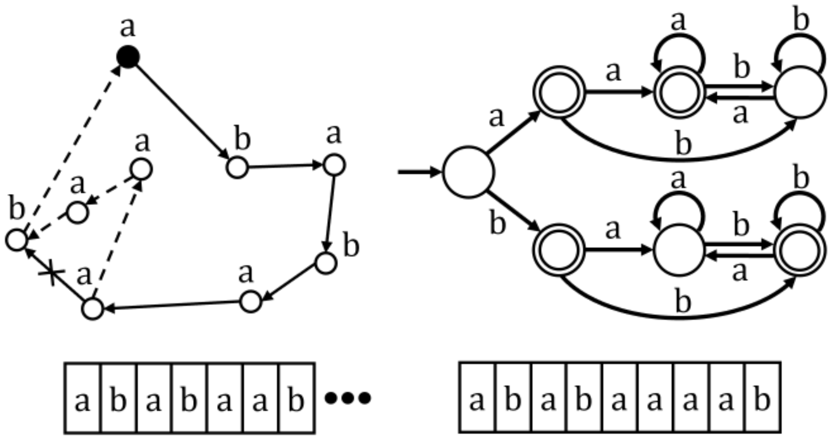


Figure 6.3: An instance of the TSPR problem (top left) with its DFA specification (top right), an infeasible partial solution (bottom left), and a complete feasible solution (bottom right). Dashed lines appear only in the complete solution, the crossed out line appears only in the partial solution, and solid lines belong to both. Tokens in the solution are marked with their corresponding member of the alphabet. The regular language represented by the DFA consists of all strings where the second token (after the starting node) and the last token are the same in the member of the alphabet to which they are mapped.

the number as W_a. Note that it suffices to enforce the number of appearances to make sure that the input corresponds to a permutation of all nodes.

We present the following SAT encoding to solve the problem of DFA acceptance while adhering to a partial solution. We use variables $\{d_{i,a} \mid 1 \leq 1 \leq |N|, a \in \Sigma_{\mathcal{A}}\}$ to represent that the i-th position in the string is a, i.e., $w_i = a$. Note that we only model the string that is to be accepted by the DFA and not the solution itself. The correspondence between the two is guaranteed by enforcing the W_a values. Moreover, we use variables $\{s_{i,q} \mid 1 \leq 1 \leq |N|, q \in Q_{\mathcal{A}}\}$ to represent that the DFA is transitioned to state q after observing $w_1, w_2, ..., w_i$. Note that encoding the initial state is implicit as it is pre-determined.

Next, we add disjunctive clauses (in parentheses), cardinality clauses (using the inequality symbol $\leq$), and assumptions (in brackets) to model how the DFA operates on its input and whether it accepts it. Clauses in Eq. (6.5) and Eq. (6.6) guarantee that exactly one member of the alphabet is chosen at each step. Clauses in Eq. (6.7) and Eq. (6.8) guarantee that exactly W_a appearances exist for each $a \in \Sigma_{\mathcal{A}}$. Clauses in Eq. (6.9) and Eq. (6.10) guarantee that exactly one state is visited at each step. Clauses in Eq. (6.11) guarantee that the first state visited after the initial one is according to the transition function, and clauses in Eq. (6.12) enforce the same validity of transition for all of the next steps according to their previous states. Clauses in Eq. (6.13) guarantee that the last state visited is an accept state. Lastly, the assumptions in Eq. (6.14) guarantee that the beginning of the solution match w^F and consequently the given partial solution.

$$\forall i : \quad (\bigvee_{a \in \Sigma_{\mathcal{A}}} d_{i,a}) \tag{6.5}$$

$$\forall i : \quad \sum_{a \in \Sigma_{\mathcal{A}}} d_{i,a} \leq 1 \tag{6.6}$$

$$\forall a \in \Sigma_{\mathcal{A}} : \quad \sum_{i} d_{i,a} \leq W_a \tag{6.7}$$

$$\forall a \in \Sigma_{\mathcal{A}} : \quad \sum_{i} \neg d_{i,a} \leq |N| - W_a \tag{6.8}$$

$$\forall i : \quad (\bigvee_{q \in Q_{\mathcal{A}}} s_{i,q}) \tag{6.9}$$

$$\forall i : \quad \sum_{q \in Q_{\mathcal{A}}} s_{i,q} \leq 1 \tag{6.10}$$

$$\forall a \in \Sigma_{\mathcal{A}} : \quad (\neg d_{1,a} \vee s_{1,\delta(q_0,a)}) \tag{6.11}$$

$$\forall 1 < i, q \in Q_{\mathcal{A}}, a \in \Sigma_{\mathcal{A}} : \quad (\neg s_{i-1,q} \vee \neg d_{i,a} \vee s_{i,\delta(q,a)}) \tag{6.12}$$

$$(\bigvee_{q \in Q^F_{\mathcal{A}}} s_{|N|,q}) \tag{6.13}$$

$$\forall i \leq l^F : \quad [d_{i,w^F_i}] \tag{6.14}$$

Note that adherence to the partial solution needs to be added as assumptions rather than clauses, since it changes for each call to the solver. Employing assumptions allows incremental SAT solving because the remaining of the encoding is unchanged for all of the successive calls. Incremental learning carries over learned clauses, that are independent of each partial solution, to future calls to the solver. These clauses are related to the inherent properties of the DFA acceptance problem and can effectively keep track of feasible and infeasible partial solutions, potentially improving the performance of successive calls to a considerable extent.

6.7 Experimental Setup

In this section, we go over the setup for producing our experimental results to evaluate beam search with cuts. We describe the problems and baseline in our setting, the tools used in our implementation, and the datasets that we test our approach on.

6.7.1 Setting

Through our experiments, we investigate whether beam search with cuts can effectively equip existing neural models with the ability to satisfy hard constraints. We focus our attention on VRPs, a well-known family of combinatorial problems that involve a variety of hard constraints. VRPs have been heavily studied in neural approaches to solve combinatorial problems. We use one of the existing approaches, namely, the attention-based model in Kool, Hoof, and Welling [162], as our baseline to solve CVRPM (Problem 9), TSPD (Problem 10),

and TSPR (Problem 11) and compare the results against our approach. The objective function to minimize in all of the problems is the total distance.

Note that the model in Kool, Hoof, and Welling does not directly support solving these problems and instead solves the corresponding simplified variants without the hard constraints, i.e., CVRP and TSP, respectively. We decode the baseline model using standard beam search with a large value for width. Doing so allows us to generate a large number of solutions for the non-constrained variant, in the hope that some of them satisfy the hard constraint as well. Note that this approach is not guaranteed to produce feasible solutions. Moreover, all of the infeasible generated solutions are disregarded after the search is terminated and do not affect the reported values.

6.7.2 Implementation

We also use the model in Kool, Hoof, and Welling as the pre-trained model in our beam search with cuts framework. We combine their model with bin packing requirements encoded in IP and regular language requirements encoded in SAT. We solve IP instances using the Gurobi solver [112] version 10, and SAT instances using PySAT [135] with Gluecard 4 solver. Gluecard 4 is a solver based on Glucose 4, extended with direct support for cardinality clauses. Gluecose 4 itself is based on Minisat 2.2 [80]. Both solvers are allocated a time limit of 10 seconds for each call. Note that a low timeout limit is chosen because, as discussed in Section 6.4, potentially numerous calls to the solver is made in each run of the BSC algorithm. Our results are produced on a server with two 12-core Intel E5-2697v2 and 128G of RAM.

6.7.3 Datasets

To cover a wide range of challenges, we use an established dataset from the literature but also synthesis our own instances. We use the instances proposed in Uchoa et al. [271] for CVRPM. Due to the difficulty in solving the corresponding IP encodings, we omit instances with a maximum number of tours larger than 20. To synthesize our own instances, we follow the procedure described in the baseline [162]. We synthesize instances for TSPR and TSPD. All of our synthesized instances are randomized using 10 different seeds and the results are averaged over seeds unless noted otherwise.

6.8 Experimental Results

In this section, we experimentally evaluate beam search with cuts in its ability to produce high quality sequential solutions that satisfy the given requirement. Our goal is to understand whether the additional effort in solving CSP requirements and making cuts is worth it, or the baseline approach of standard beam search with large quantity is already good enough to satisfy requirements. Furthermore, we aim to understand the trade-off between feasibility in requirements and the solution quality, i.e., whether enforcing satisfaction comes

with significant cost to quality. Lastly, we explore ways to improve the performance of the beam search with cuts procedure.

6.8.1 Sequence Generation with Requirements

For our first experiment, we solve Problem 9 (CVRPM) for instances in Uchoa et al. For our approach, we use a bin packing requirement that enforces a maximum number of tours. As discussed, we omit instances with a maximum number of tours larger than 20. We run the standard beam search decoder with a comparatively large width (8096) and our beam search with cuts approach with significantly lower width (4).

We observed that in 9 out of 27 instances, no generated solutions using the standard beam search were able to satisfy the maximum number of tours requirement. Contrarily, beam search with cuts, with a lower width value, was able to find feasible solutions in all cases. This disparity indicates that requirement satisfaction in BSC cannot simply be compensated for by using a higher width value in BS. Table 6.1 contains the BS and BSC results for the 9 unsatisfied cases. We report the required maximum number of tours m alongside the best value that each approach was able to achieve. Additionally, we include the best total distance amongst the feasible solutions generated by BSC in relative comparison against the best from BS (Δdis. (%)) with a negative value indicating an improvement in BSC over BS. Our results show that despite the lower width and the added benefit of satisfying the requirement, BSC was able to achieve solutions of comparable total distance, with even significant improvement in some cases. Furthermore, we observe that ordinary beam search shows higher runtimes in the majority of cases, with BSC taking longer only if a significant number of cuts are needed to prune infeasible solutions. While the runtimes can be further improved with parallelized or GPU-based methods, the results still indicate the intractability of simply increasing the width to satisfy requirements. Table 6.2 contains the BS and BSC results for the 18 satisfied cases. The results match our expectation that the high width used in standard beam search leads to better quality solutions in all but one case. Aligned with our observation in the infeasible cases, BSC achieves better runtime in all instances because it does not need large width to satisfy the requirement.

Width Parameter

To better understand the effects of the width parameter, we use beam search with cuts in three configurations to solve the same 9 instances where standard BS failed to produce a feasible solution. The configurations consist of: a beam width of 4, a larger beam width of 64, and a hybrid approach referred to as *sub-width*. The sub-width approach is a variation of our BSC algorithm that only guarantees a fraction of the partial solutions to be extendable to complete feasible ones. Specifically, we consider a sub-width value (= 4) in addition to width (= 64) and we solve the CSP problem starting from the highest score as many times as needed in order to fill the sub-width portion with solutions that can lead to feasibility. The partial solutions shown to be infeasible will then be disregarded and the remaining

Instance	Beam Search		Beam Search with Cuts			
($\|N\|$,m)	Time (s)	m	Time (s)	m	Δdis.	Cuts
(134,13)	26.7	14	45.1	13	0.8%	168
(157,13)	35.7	14	14.9	13	-12.2%	17
(190,8)	49.3	11	14.4	8	-23.9%	52
(209,16)	60.7	17	28.3	16	2.0%	19
(214,11)	61.7	12	27.0	11	5.7%	92
(233,16)	73.4	17	384.6	16	20.1%	219
(256,16)	89.3	17	630.4	16	9.0%	408
(367,17)	185.6	18	84.4	17	-0.6%	6
(411,19)	236.7	22	116.7	19	-14.6%	44

Table 6.1: Beam search (width=8096) against beam search with cuts (width=4) on select instances from Uchoa et al. for which BS fails to generate a feasible solution.

Instance	Beam Search		Beam Search with Cuts			
($\|N\|$,m)	Time (s)	m	Time (s)	m	Δdis.	Cuts
(106,14)	26.93	14	9.34	14	0.75%	2
(110,13)	19.92	13	8.60	13	1.53%	0
(115,10)	20.88	10	8.11	10	5.69%	23
(120,6)	21.67	6	6.33	6	0.48%	0
(129,18)	25.46	18	13.13	18	1.36%	3
(139,10)	28.10	10	9.71	10	1.99%	4
(143,7)	28.69	7	8.50	7	6.20%	0
(162,11)	35.89	11	13.21	11	1.66%	0
(167,10)	38.43	10	13.98	10	3.24%	0
(186,15)	46.43	15	24.42	15	3.17%	0
(204,19)	64.86	19	36.60	19	5.25%	0
(237,14)	80.15	14	34.60	14	0.19%	0
(261,13)	86.71	13	38.87	13	7.37%	0
(280,17)	99.40	17	57.76	17	0.75%	0
(284,15)	104.05	15	49.75	15	1.37%	0
(308,13)	118.88	13	53.45	13	-3.54%	0
(327,20)	140.90	20	82.62	20	3.00%	0
(331,15)	146.51	15	61.70	15	0.80%	0

Table 6.2: Beam search (width=8096) against beam search with cuts (width=4) on select instances from Uchoa et al. for which BS manages to generate at least one feasible solution.

width will be filled with untested ones. To extend the BSC pseudo-code (Algorithm 4) with sub-width capability, line 8 is circumvented and $feas$ is set to true on the condition that $|S_{i+1}| \geq \hat{w}$, with $\hat{w}$ being the sub-width value.

The results for our experiment on the three width parameter configurations is presented in Table 6.3. We report the same values that we did in Table 6.1. Note that we omit reporting the number of tours achieved as all approaches are able to satisfy the given specification in all instances. As expected, the approach with a width of 64 and the hybrid approach are both able to produce solutions with better distance compared to a width of 4 in all but two

instances. Interestingly, we observe that the hybrid approach is closer in runtime to the lower width as opposed to the higher one, despite employing a significantly more number of cuts compared to the lower width approach. In two instances, employing sub-width leads to even more cuts compared to larger width. The high number of cuts is hypothetically due to the high number of infeasible but high quality partial solutions that are introduced from the section of the search that is not checked for feasibility. Due to being left unchecked for potentially multiple iterations, the infeasibility check for these solutions can be straightforward, explaining the low runtime to number of cuts ratio.

Instance		Beam Search with Cuts		
($\|N\|$,m)	Width	Time (s)	Δdis.	Cuts
(134,13)	64	918.5	-1.6%	7609
	64-4	90.1	-0.6%	2397
	4	45.1	0.8%	168
(157,13)	64	180.6	-12.7%	178
	64-4	24.0	-12.7%	147
	4	14.9	-12.2%	17
(190,8)	64	189.3	-24.1%	1136
	64-4	56.4	-23.6%	2186
	4	14.4	-23.9%	52
(209,16)	64	372.7	2.4%	192
	64-4	40.2	0.2%	175
	4	28.3	2.0%	19
(214,11)	64	405.2	0.8%	1126
	64-4	40.5	0.5%	282
	4	27.0	5.7%	92
(233,16)	64	4869.6	18.2%	4172
	64-4	441.5	17.7%	3528
	4	384.6	20.1%	219
(256,16)	64	8652.7	5.7%	6541
	64-4	735.2	6.1%	1814
	4	630.4	9.0%	408
(367,17)	64	1258.4	-1.5%	505
	64-4	123.5	-1.5%	277
	4	84.4	-0.6%	6
(411,19)	64	1700.4	-15.6%	466
	64-4	251.6	-16.8%	1610
	4	116.7	-14.6%	44

Table 6.3: BSC with different width configurations (sub-width indicated by -) on select instances from Uchoa et al. for which BS fails to generate a feasible solution.

6.8.2 Tightening Requirements

In our second experiment, the goal is to understand the limits of feasibility and the trade-off between challenging requirements and solution quality. Thus, we avoid fixing the requirement as part of the input and instead use BSC to satisfy incrementally tighter requirements until no feasible solution exists. We use this approach to solve Problem 10 (TSPD) for synthesized instances where the demand for each non-supply node is randomly selected from a uniform distribution. We use a bin packing requirement and tighten it by decreasing the capacity c. Moreover, we solve Problem 11 for synthesized instances where an equal number of nodes are randomly mapped to each member of the alphabet. We use a regular language requirement represented by the DFA $\mathcal{A}_{s_i}$. The requirement enforces that in the TSPR solution x, there cannot be i successive visits to nodes that all map to the same member of the alphabet by σ. This requirement can be tightened by decreasing the maximum successive visits i.

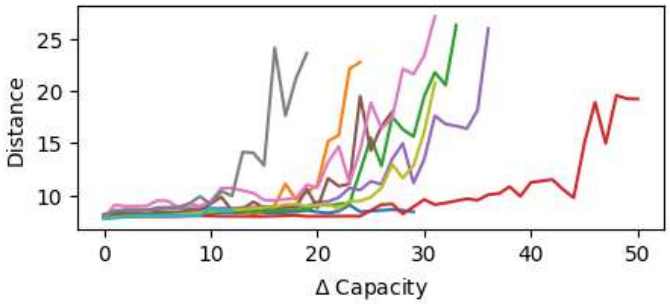
We first run a standard beam search and observe the lowest c value for TSPD or the lowest i value for TSPR that any of the generated solutions satisfy. This value represents the tightest requirement that BS can satisfy and acts as a starting point. We iteratively tighten the requirement (by decreasing c or i) and use BSC to solve the tighter cases until we fail to generate a feasible solution.

The TSPD results are presented in Figure 6.4 (top) and the TSPR results in Figure 6.4 (bottom). The horizontal axes indicate how much tighter the requirement is, compared to the starting point. Namely, a value of zero for Δ indicates using BS (width=8096) and positive values for Δ indicates using BSC (width=4). Moreover, larger Δ values equate to tighter requirements and lower c or i values. The vertical axes show the total distance (objective value) of the solutions. Each line (color) represents an instance with a different seed.

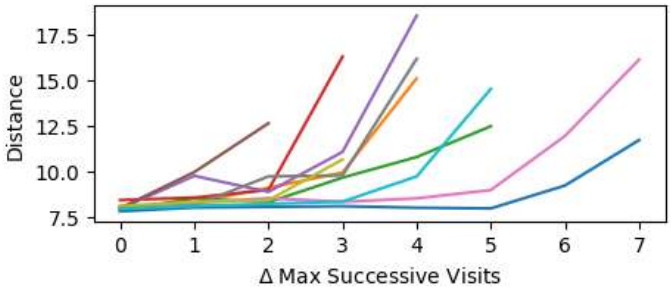
The results show that BSC, even with lower width, is able to reduce capacity by a margin of 10 to 30 across instances, and reduce the maximum number of successive visits by up to 5 on some instances, with relatively minor impact on total distance. In both cases, we see that tightening the requirement too much will result in a sudden jump in total distance before reaching infeasibility. In all instances, the infeasible runs contain no timeouts, guaranteeing that the declaration of infeasibility is valid and there cannot exist any feasible solutions. In TSPD, we see that highly challenging requirements also exhibit instability in total distance. This observation, alongside the decline of quality, can be indicative of the fact that the search is too constrained to explore solutions of high predictive scores.

6.8.3 Scaling Problem Size

In our next experiment, we study the impact of solution length and requirement strength on the ability of BS and BSC to produce feasible solutions. We solve Problem 11 (TSPR) for synthesized instances where an equal number of nodes are randomly mapped to each member of an alphabet of size 6. We use a regular language requirement represented by the DFA $\mathcal{A}_i^P$



(a) TSPD.



(b) TSPR.

Figure 6.4: Iterative tightening of the requirement against solution quality, from beam search ($\Delta = 0$, width=8096) to beam search with cuts ($\Delta > 0$, width=4) until infeasibility. Different colors represent randomly-generated synthetic instances using different seeds.

that is the combination of DFAs $\mathcal{A}_1^p$ to $\mathcal{A}_i^p$. The language of $\mathcal{A}_i^p$ consists of solutions x, where there are not two successive nodes x_j and x_{j+1} such that $\sigma(x_j) = a_i$ and $\sigma(x_{j+1}) = a_{i+1}$ with a_i and a_{i+1} being the i-th and $i+1$-th members of the alphabet. Combining multiple DFAs into one allows us to quantify and control the requirement strength. Note that we can combine DFAs and evaluate their joint acceptance by simply considering the DFA product.

First, we use standard beam search to solve instances of size 24 (i.e., solution length) with $\mathcal{A}_1^P$ to $\mathcal{A}_5^P$ as requirements. We start from a width of 1 and double the width every time BS fails to generate a feasible solution. Second, we solve instances of size 12, 24, and 36 with $\mathcal{A}_3^P$ as requirement and double the width in case of infeasibility as before. For all cases, we report the average logarithm of the lowest width that was able to satisfy the requirement. For comparison, we solve all cases again using BSC with a width of 4. Note that doubling the width is not required for BSC as it is complete with regard to satisfying requirements at any width.

The results, provided in Table 6.4, show that with a linear increase in the number of combined DFAs in the requirement, there is an almost linear increase in the logarithm of the lowest width required for BS. When the requirement is fixed, a similar linear relation exists between the length of the solution (number of nodes) and the logarithm of the lowest width required. The trends show that the beam search decoder needs to produce an expo-

nentially larger number of solutions in order to satisfy a stronger requirement or satisfy the same requirement for a longer solution. Beam search with cuts, however, is able to satisfy combined requirements and produce longer solutions with an almost linear increase in cuts and negligible cost to runtime.

$\|N\|$	R	Beam Search $log(w)$	Beam Search with Cuts Cuts	Time (s)
24	P_1	2.7	2.8	1.81
	P_2	4.4	6.4	1.79
	P_3	6.8	8.4	1.80
	P_4	8.9	12.4	1.78
	P_5	10.5	15.4	1.87
12	P_3	1.2	3.5	1.73
24		6.4	8.4	1.80
36		11.1	16.7	1.95

Table 6.4: Minimum width required in log-scale for BS against the number of cuts and time used by BSC (width=4) for iteratively stronger requirements and longer solutions.

6.8.4 Incremental CSP Solving

In our last experiment, we investigate how incremental SAT solving can enhance the performance of beam search with cuts. We solve Problem 11 (TSPR) for synthesized instances where an equal number of nodes are randomly mapped to each member of an alphabet ($\Sigma_{\mathcal{A}}$) of size 4. We use a regular language requirement represented by the DFA $\mathcal{A}^W$. Acceptance by $\mathcal{A}^W$ states that within any window of length 8 in the solution, there should be at least one appearance for every member of the alphabet through the mapping σ.

We run the BSC algorithm for width values of 4 and 16 and different instance sizes. For the incremental runs, we allow the solver to carry information (i.e., learned clauses) forward to successive calls. In the non-incremental runs, we wipe the SAT solver's slate clean after each call. We report the total time that is purely spent on solving SAT instances for comparison. Note that we only report the number of cuts once as the two approaches behave exactly the same with regard to cuts since no timeouts are involved. Moreover, we omit the information on solution quality as the two approaches produce the exact same solutions for the same reason.

The results presented in Table 6.5 show that the incremental approach is able to improve solve time in all cases. Furthermore, the relative difference in solve time grows monotonically as we increase the size of the instance or the width of BSC, closely correlating to the number of cuts in both cases. Note that increasing the width or the solution length results in making more calls to the solver, which generally amplifies the improvement when employing incremental solving.

$\|N\|$	Width	Cuts	Solve Time (s) Non-Inc.	Inc.
12	4	1.2	1.59	0.45
	16	4	6.00	1.56
24	4	14.5	13.95	2.45
	16	64.5	57.00	7.62
36	4	40.9	39.58	5.59
	16	143.7	147.59	14.94
48	4	66	89.65	10.94
	16	233.4	345.05	32.09
60	4	94.9	150.17	15.28
	16	332.1	565.47	33.92

Table 6.5: Solve time of incremental and non-incremental SAT solving for beam search with cuts.

6.9 Related Works

A recent example in combining neural sequence models with combinatorially specified constraints is presented in Lafleur, Chandar, and Pesant [170]. The authors base their work on the algorithm in Jaques et al. [148] which tunes sequence models by using reinforcement learning to optimize an added reward function as well as the predictive scores. The algorithm in Lafleur, Chandar, and Pesant replaces hard-coded penalties with specifications in constraint programming (CP), claiming that it better suits the problem of music generation which both papers focus on. Specifically, belief propagation is used, which is a heuristic estimating the marginal probability of value-variable pairs that participate in feasible solutions. The marginal probability and the number of violations are then treated as signals that, when optimized, guide the sequential generation towards solutions that are feasible in music theory constraints. All the while, simultaneously optimizing the predictive scores from the original model guarantees that the generated melodic lines also reflect the style of a given corpus. The shift from hard-coded penalties to CP constraints enjoys the same benefits that we argue for when using the formulation approach to solving combinatorial optimization problems. Namely, the declarative nature allows flexibility and modularity in specification. Some of the potential issues with the work in Lafleur, Chandar, and Pesant include: (1) the lack of guarantee in finding feasible solutions, because the final product still relies on deep learning models and has feasibility as only one of its optimized objectives; (2) the non-trivial task of finding a neural model capable of satisfying a given CP specification, and training such a model which is resource- and time-consuming; (3) the potential cost to solution quality due to manipulation of predictive scores with marginal feasibility probabilities; (4) the lack of accuracy and reliance on heuristics that comes with using belief propagation; and (5) the limitation to belief propagation which is only supported in the CP problems, excluding a variety of other standard combinatorial problems that can be used to specify CSPs. The work in Yin, Cappart, and Pesant [299] expanded upon and improved

this work in the aspects of CP specification, evaluation of violated constraints, and reward signals.

A more limited but common set of requirements imposed on neural sequence models are lexical constraints. Lexical constraints require a set of phrases to appear in the output and are significantly less expressive than requirements encoded as CSPs. The work in Hokamp and Liu [122] achieves constrained beam search with lexical constraints by dividing the partial solutions in each iteration based on their number of satisfied constraints. Their algorithm, referred to as grid beam search, is shown to significantly improve tasks such as translation in interactive and non-interactive scenarios. The approach in Post and Vilar [227] also divides the search based on satisfied constraints count, but utilizes dynamic beam allocation to improve the performance by reducing the complexity of the search from linear to constant in the number of the constraints. A more sophisticated approach, which categorizes partial solutions based on the specific set of satisfied constraints rather than merely the count, is presented in Anderson et al. [13]. A finite-state machine encodes the constraints as a whole by representing each possible satisfaction status as a state and each appearance of a word as a transition. The size of the machine, and the complexity of the search as a result, can be exponential in the number of constraints. The authors use this approach to caption images with the enforced inclusion of certain keywords. Lexical constraints can also be supported by augmenting the predictive scores in the beam search to lean towards the target words, as presented in Pascual et al. [221]. Similar to our work, all of the aforementioned beam search-based approaches to satisfying lexical constraints do not require re-training the model. Instead, they structure the decoding procedure towards constraint satisfaction, facilitating modularity and application to different neural models and constraints. Alternative approaches to satisfying lexical constraints include modifying the output through stochastic sampling [200] or gradient-guided [247] edits.

Lastly, standard beam search can also be used to agnostically solve neural sequence generation with requirements. This approach, referred to as complete any-time beam search (CAB), consists of multiple runs of standard beam search. The width of the search, and the number of solutions as a result, is doubled at each iteration until a feasible solution is found. The main benefit of CAP is its simplicity and flexibility in application to any neural model and any requirement represented in a variety of paradigms such as CSPs, formal languages, and even hardcoding. The major downside in utilizing CAB is the lack of feasibility guarantee and any optimization or guidance towards satisfying the requirement. Recent work in Cohen and Beck [63] has shown CAB to exhibit fat- and heavy-tailed behavior, similar to other complete search algorithms (e.g., [105, 62, 83]). The authors propose randomized restarts by adding noise to the input of the neural model, mitigating the heavy-tailed behavior and improving performance across four tasks.

6.10 Conclusion and Future Work

In this chapter, we present an approach to use combinatorial problems for constrained ML. Namely, we propose the beam search with cuts framework that allows us to combine pre-trained neural sequence models with requirements. Requirements are formulated as CSPs into declarative combinatorial problems that do not have an optimization aspect. Our algorithm enables constrained decoding without relying on re-training the neural models for this purpose, even guaranteeing requirement satisfaction. The plug-and-play nature of our approach is evident in our proposed requirements that are applicable in multiple settings. We introduce and implement the bin packing requirement in IP and the regular language requirement in SAT. Utilizing the two combinatorial problems demonstrates the variety of CSPs in different specifications that our algorithm can support. We use a pre-trained model from recent work and our two implemented requirements to solve three VRP variants as a case study. Note that neither our framework nor our requirements are particularly tailored for VRP. We showcase beam search with cuts using VRP but also pave the way for application to other combinatorial, or generally sequential, problems that can be solved using constrained generation with neural models. When solving combinatorial optimization problems, the division of responsibility between neural sequence generation and CSP requirements can be roughly viewed as decomposing the problem into its optimization and hard constraints components. While the objective is optimized using deep learning, the feasibility aspect is in turn handled by a formulation approach to combinatorial reasoning.

Through our experiments, we show that cuts in beam search enable satisfaction of requirements where standard beam search fails. Even in cases where beam search is able to satisfy requirements, we show that it requires an exponentially larger width as the length of the solution or the strength of the requirement increases. Contrarily, BSC's performance remains stable as the instances get more challenging, highlighting the importance of explicitly solving feasibility problems. Furthermore, we show that satisfaction of requirements does not necessarily come with a significant impact on quality, and can even improve the objective value in some cases. This demonstrates that constraining ML solutions is not compromising their quality, and is aligned with the same argument that we made for interpretable ML in Chapter 3, Chapter 4, and Chapter 5. We also experimentally depict the trade-off between feasibility and solution quality, and show that BSC can satisfy considerably tighter requirements with negligible cost to quality, while also correctly declaring infeasibility in severe cases. Lastly, we investigate ways to improve the performance by: (1) checking feasibility for only a portion of partial solutions; and (2) employ incremental SAT solving to transfer information between calls. Both approaches are shown to be successful in improving runtime, though the former can slightly impact solution quality both positively and negatively.

Our approach can be developed in multiple interesting directions. We can maintain records of equivalency between partial solutions as the model is being decoded, e.g., two partial TSPD solutions that have visited the same nodes in different orders are equivalent.

Doing so would allow us to avoid explicit checks for deciding the feasibility of a partial solution, when the result for an equivalent one is cached and can be queried. This technique improves the performance in a similar vein that incremental solving does, but is independent of the solver's capability for solving the instances incrementally. The more explicit approach to incremental solving can also empower the expert user with a greater level of control to to customize the equivalency relations, potentially with precise formulations. Equivalency checks can also cut solutions that are of lower quality, e.g., higher total distance, in order to improve the search or increase the diversity of the results. Compared to the standard infeasibility cuts, the ones based on equivalency more closely resemble no-good cuts [25] from the literature, since they add a constraint and prevent equivalent solutions to be considered in future.

In this chapter, we decided to cut partial solutions causing timeouts to maintain completeness. However, there also exists arguments for keeping these solutions as they might represent interesting edge cases that are barely feasible but high in quality. A BSC algorithm that keeps timed out partial solutions is incomplete. However, completeness can be achieved using a hybrid approach similar to sub-width (presented in Section 6.8.1). In such an approach, a fraction of search only keeps solutions that are guaranteed to be feasible, while the other part also keeps timed out solutions for exploratory purposes. Another tweak that leads to losing completeness is to use incomplete cuts, i.e., those that detect a subset of infeasible partial solutions. Constraint propagation in constraint programming is an example method that can be used to implement incomplete cuts, significantly improving performance compared to solving CSP instances to completion. Another example is combinatorial local-search methods that try to find feasible solutions but are not guaranteed to do so even if one exists. Local search is also considerably faster than fully solving a CSP instance. Similar to before, a hybrid approach can be utilized to have a complete algorithm even when incomplete cuts are employed.

Furthermore, our approach can be extended by implementing new types of requirements such as those based on scheduling, set cover, or coloring constraints. It can also be extended in application to other neural sequence models in areas such as planning or program synthesis where neural sequence models have been successful and could significantly benefit from support for guaranteed satisfaction of hard constraints. Lastly, cuts can be added to the beam-stack search procedure [310], a variant of beam search where backtracks are also utilized. Using backtracks helps mitigate the impact of leading the search towards a suboptimal space due to employing cuts.

Chapter 7

Conclusion

7.1 Summary

Common ML models such as deep neural networks and boosted trees can be challenging to interpret by humans and/or lack the ability to specify and enforce constraints. Such models do not meet the human-compatibility and transparency demand in applications where mistakes are costly and expert auditing is necessary. In contrast, in this dissertation we proposed using combinatorial optimization problems to achieve interpretable and constrained ML by formulating interpretable models, different classification and clustering objectives, and constraints representing user knowledge, requirements, and/or training considerations. The combinatorial approach is rigorous and has theoretical guarantees on quality and satisfaction of specifications. The wide range of problems that we solved using different declarative combinatorial optimization problems demonstrates that they can, and should, play a crucial role in the future of AI. Our collective approaches provide a framework that extends the current ML landscape with complex reasoning and enhances trust in ML solutions to give confidence to developers and end users that the models are acting as intended.

One aspect of the work presented in this dissertation is training ML models by formulating the training as a combinatorial optimization problem. We focused on using the declarative combinatorial optimization problem of MaxSAT, and its decision variant SAT, to learn inherently interpretable models in Chapter 3, Chapter 4, and Chapter 5. We significantly improved the existing state-of-the-art encodings and solved some problems for the first time, demonstrating the enhancement potential in existing formulations of ML problems. The improvements are far from trivial and are mostly due to shifts in perspective such as: (1) designing encodings for settings that better suit ML solutions rather than those that are closely associated with SAT, e.g., directly supporting numerical features rather than binarizing them; (2) alternative approaches to optimizing objectives, e.g., modeling how much a diagram can be reduced rather than directly modeling its size; and (3) employing preprocessing or multi-stage schemes to alleviate the computational burden on a single call to the solver, examples include pruning or finding a feasible subset of constraints.

Using declarative combinatorial optimization problems provided the theoretical guarantee of finding optimal solutions for clustering and solutions that are optimal with regard to the training data for classification. Note that optimality is only reached and proved given sufficient resources such as time and memory. However, most solvers for declarative combinatorial optimization problems provide provable upper and/or lower bounds on the optimal objective value in case of a timeout. Using declarative combinatorial optimization problems was also shown to allow repurposing components and encoding different constraints and objectives to solve a wide variety of problems. Included in those are objectives further optimizing the interpretability of our models, e.g., by size minimization. The formulation approach also enables the possibility of further improving the performance of our approaches by using more efficient solvers in the future. A shared conclusion throughout these chapters is that our approaches are able to produce highly accurate interpretable solutions in short runtimes. Additionally, the interpretability does not significantly compromise solution quality. In fact, it often enhances it in cases where a solution could benefit from limitation to a structure. This observation is consistent with the literature, but is extended to new settings in our work.

The other aspect of our work is using combinatorial problems to formulate and satisfy ML constraints. When we learned a model as a combinatorial optimization problem, we specified the constraints within the same problem so that a solution is forced to adhere to the constraints. We employed this approach in Chapter 3 to enforce constraints that prune overfitting models and in Chapter 5 to enforce constraints based on domain expertise. Constraints can also be formulated independently as a CSP and accompany models that are pre-trained by other approaches such as deep learning. We employed this approach in Chapter 6 to guarantee inference-time constraint satisfaction and extend rigorous reasoning capabilities of a neural sequence model. Even though we focused on a specific case study, we proposed a novel modular framework that treats pre-trained models and CSP requirements as interchangeable components, providing a new perspective in combining them to solve complex problems in multiple settings. Using combinatorial models to formulate constraints provides the theoretical guarantee of satisfaction, whether the formulation includes the model itself or not. Overall, we observed how constraints can be utilized to serve different purposes in ML such as enhancing training, enabling expert audit, or elevating models in terms of combinatorial reasoning. In each of these cases, we concluded that constraint satisfaction does not significantly compromise solution quality despite limiting the solution space. Contrarily, it can improve solution accuracy, and even objective value in non-exact approaches.

This dissertation as a whole has demonstrated the capability of employing combinatorial optimization problems towards realizing interpretable and constrained ML, with each chapter acting in support of this objective by proposing an algorithm to solve a key problem.

> **Thesis Statement.** Formulating machine learning tasks as declarative combinatorial optimization problems enables learning a wide variety of interpretable

and constrained machine learning models, providing theoretical guarantees and often improving solution quality.

7.2 Contributions

In this section, we summarize the contributions of this dissertation, categorized by chapters.

Chapter 3: Optimal Decision Tree Classification

1. We solved the problem of interpretable classification with decision trees by using a novel SAT-based encoding with support for min-depth and max-accuracy objectives. Our encoding directly supports branching for numerical and categorical features, avoiding a quadratic increase in size as a result of binarizing non-binary features.

2. Our approach was shown to significantly outperform the state of the art on datasets with mostly numerical features and provide anytime solutions of higher quality. It was also shown to consume significantly less memory, particularly compared against the approaches based on caching branch-and-bound search.

3. We showed that direct encoding of categorical features enables power set branching, leading to more expressive solutions. Such solutions were shown to be of higher quality, leading to higher training and testing accuracies.

4. We extended our objective to support different costs for different cases of misclassification. The support for cost-sensitive learning was showed to significantly improve balanced accuracy when training on imbalanced datasets.

5. We added support for constrained ML by encoding minimum support and minimum margin pruning constraints. The pruning was shown to improve generalization to unseen data despite limiting the solution space.

Chapter 4: Compact Binary Decision Diagrams for Classification

1. We solved the problem of interpretable classification with binary decision diagrams with a novel SAT-based encoding. We reused the basis of our decision tree encoding with direct support for non-binary features and focused on maximum accuracy as the objective. Our approach initially models a non-reduced BDD, which can subsequently be reduced by assigning labels to empty terminals and merging equivalent nodes.

2. Through our experiments, we showed that our BDD encoding improves the state of the art in runtime and, in case of a timeout, in solution accuracy.

3. To improve compactness and, as a result, interpretability, we extended our encoding by modelling the final size to which the BDD can be reduced. We used the size as a secondary objective to optimize simultaneously, or in a post-processing stage by only

affecting empty terminals. We observed that size optimization in both cases results in more compact solutions while maintaining accuracy.

4. We extended our encoding to directly learn diagrams rather than learning non-reduced BDDs first. We focused on multi-dimensional BDDs, a family of diagrams that employ multi-dimensional branching and are more expressive than BDDs using the same number of branchings. We experimentally showed that multi-dimensional BDDs scale better than BDDs due to their expressiveness and ability to utilize more features towards higher-quality solutions in compact sizes.

Chapter 5: Constrained Clustering via Decision Trees

1. We solved the problem of interpretable clustering with decision trees to optimality for the first time in the literature. We used a novel SAT-based encoding that extends the basis of our decision tree classification approach with ϵ-approximation of a Pareto-optimal solution in maximum diameter (MD) and minimum split (MS) criteria. We also added support for constrained clustering by encoding pairwise user-provided constraints.

2. Our experimental results showed that we can find interpretable solutions of high quality in short runtimes. It was further shown that tree clustering improves the non-tree counterpart in evaluation metrics, despite limiting the solution space by considering interpretability. The pairwise constraints and the bi-criteria objective were also shown to improve quality, with any combination of the three factors further magnifying the improvement.

3. We joined our bi-criteria objective and pairwise constraint encodings in smart pairs, a novel pruning framework that detects and removes redundant or infeasible clauses. Smart pairs, along with the approximation of the objective, is shown to significantly boost performance leading to higher quality solutions.

4. We found a trade-off between solution accuracy and infeasibility in the presence of user-provided constraints. Thus, we further extended our encoding by support for soft constraints, that are encouraged rather than required to be satisfied. We proposed 1-stage, 2-stage, and 3-stage schemes that consider different orders of prioritization for soft constraint satisfaction and the clustering objective. We experimentally showed that soft constraints can improve accuracy by enabling a higher level of semi-supervision without the risk of infeasibility.

5. We used our encoding in an iterative algorithm to find approximations to all of the Pareto-optimal solutions in the bi-criteria objective, rather than only one. Our experiments showed that computing the Pareto front leads to finding higher-quality solutions in instances with less pairwise constraints.

Chapter 6: Neural Sequence Generation with Requirements

1. We introduced the beam search with cuts procedure to solve the problem of constrained neural sequence generation. Our approach can combine any pre-trained neural sequence model with a requirement encoded as a CSP, resulting in a decoder that guarantees the satisfaction of the given requirement. BSC works by employing explicit feasibility checks for partial solutions, referred to as cuts.

2. We proposed the versatile and universal requirements of bin packing and regular language acceptance, and implemented them as cuts in IP and SAT, respectively.

3. We focused on three VRP variants as the case study for our approach. We solved these variants by using BSC to combine the support for simpler variants in a pre-trained neural model and our two implemented requirements.

4. We experimentally showed that BSC can satisfy requirements in a short runtime and without significant cost to solution quality. We also showed that BSC is capable of satisfying increasingly tight requirements until they become infeasible.

5. We compared BSC against standard beam search as the length of the solution or the strength of the requirement is increased. The results showed that BSC scales exponentially better than its standard counterpart.

6. We investigated improving the performance of BSC by employing incremental solving when satisfying a requirement encoded in SAT. The results showed a significant improvement in the solver runtime, which was further amplified for longer solutions.

7.3 Future Work

In the main body of the dissertation we showed how combinatorial optimization problems can be used to learn inherently interpretable models with the capacity to enforce constraints but also be used in conjunction with deep learning models by formulating constraints only. In the same vein, we group interesting directions for future work into multiple paradigms, including the two aforementioned ones. For each paradigm, we highlight some of the high-level benefits and challenges and include some example ideas. Note that the following discusses a more general view of the next steps regarding the topic of this dissertation, and more concrete and incremental future work was already discussed in each individual chapter.

7.3.1 Combinatorial Optimization for Learning Inherently Interpretable Models

This approach and its strengths and weaknesses were comprehensively discussed in the first three chapters. However, an exciting next step is to merge the three encodings into a unified framework for using SAT to learn inherently interpretable models. An elegant merge

is possible because the approaches share and reuse the fundamental aspects of encoding branchings and tree-like decision structures. The unified framework can easily be modular in interchanging solution formats, constraints, and objectives. Example problems that were not supported in our original approaches but can be solved using this framework include but are not limited to: (1) decision tree clustering with direct support for categorical features and power set branching; (2) clustering using oblivious decision trees or BDDs; (3) BDD classification with minimum support or margin constraints and cost-sensitive learning objectives; and (4) multi-dimensional branchings in decision trees. A unified framework also paves the way for implementing new components such as: (1) clustering objectives, e.g., sum of (squared) distances; (2) solution formats, e.g., decision lists or sets; and (3) constraints, e.g., fairness specifications. The added components can readily interact with the existing ones, enabling applicability to multiple settings yet to be imagined.

The work on implementing the toolset for the unified framework of using SAT to learn decision trees and diagrams has already begun. Our goal is to redesign our implementations into a coherent architecture and provide a user-friendly interface that can communicate with the existing common libraries and be easily integrated into a user's project. The well-defined architecture fosters continuous development of new features and capabilities.

Another interesting direction is to formulate more complex inherently interpretable models into combinatorial optimization problems. A straight-forward extension of decision trees are random decision forests [121], where an ensemble of trees classify a given data point with the majority of predictions dictating the final label. Our SAT-based encoding can extend its support to optimal random decision forests, an area that has seen interest in recent work [5]. An encoding for an entire ensemble can be constructed by concatenating multiple tree encodings and merging the assigned labels for each point. However, more elaborate structures can be investigated where multiple trees share some branchings in predetermined connections.

Decision trees can also be expanded upon by employing more expressive branchings. An IP-based formulation can easily be extended to support multivariate branchings where a linear combination of features is employed rather than one [37]. The extension is not easy for SAT-based methods due to their lack of support for numerical variables. Multivariate branchings can be further increased in expressiveness by employing a more comprehensive model of computation. Namely, it can be replaced with an arbitrary expression of mathematical operators, represented by a syntactic tree. This approach would be, in spirit, similar to multi-dimensional branching as they both employ multiple features and assign the branching values based on an inner tree.

7.3.2 Combinatorial Optimization for Post-hoc Interpretability

Combinatorial optimization problems can also be employed to learn interpretable models in a post-hoc manner, focused on understanding existing models or using black-box models as an intermediate to training data due to their superior scaling. Recent literature has focused

on post-hoc interpretability for explaining a model as a whole, or its particular behavior in the neighborhood of a given data point [268, 233, 226, 109]. Our SAT-based approach can trivially support post-hoc interpretability through sufficient sampling of (local or global) data points and labels assigned by the model. A more interesting yet ambitious idea is to encode an objective into our approach that aims to replicate a neural model's behavior directly rather than via sampling. Such an optimization is likely to be computationally costly, but can still be applied to replicate and replace a part of the model in a neural-symbolic manner. For an example, post-hoc interpretability can be utilized to replace the very last layer in a deep network, explaining decisions based on features that have been learned by the previous layers combined. Conceptually, achieving interpretability in this approach could, for example, explain classification as a cat drawing by the existence of a circle with two triangles on top.

Post-hoc interpretability has also been investigated in the area of deep reinforcement learning. A common approach is again to sample points from the Q function to fit a decision tree mapping states to preferred actions [10]. However, an arbitrarily accurate decision tree in this aspect can still perform disastrously, as one wrong action can derail the agent from acquiring future rewards. Other approaches in the literature construct the decision tree policy by direct communication with the environment or the Q function [235, 274, 294]. An approach based on combinatorial optimization can provide guarantees on replicating a Q function while also addressing the issue of scalability by focusing on an abstraction of the state space. More specifically, the problem of finding a decision tree policy on the abstraction and a lower and upper bound on the reward value objective can be formulated into a combinatorial optimization problem. The lower bound is what can be achieved by the decision tree policy alone within the abstraction, and the upper bound is what can be achieved under the assumption that the policy performs as well as the original Q function in cases where the abstraction no longer holds. An iterative approach can refine and expand the abstraction until a termination condition, e.g., lower and upper bounds being sufficiently close, is met.

7.3.3 Combinatorial Constraint Formulation for Pre-trained Models

In addition to explaining them in a post-hoc manner, combinatorial components can also help constrain deep learning models into desired behaviors or specifications. In Chapter 6, we presented a search procedure that constrained the output of a pre-trained neural sequence model by repeatedly solving the feasibility problem for constraints as the solutions are being generated. However, the literature on neural sequence models is not limited to one-time iterative generation, there also exists approaches that edit a generated or existing solution to denoise [232], debug [128], or refine [113] it and even to enforce lexical constraints [200]. Editing a sequence facilitates a more global view on the solution as a whole, but usually requires more complex models. Inspired by this perspective, our modular approach to

combine pre-trained black-box models with CSP requirements can be reconceived. Specifically, we can have an approach in which solutions are generated first and then edited to satisfy the requirements. An obvious candidate for the objective function in the editing stage is a criterion akin to edit distance. However, slightly editing the original solution is not guaranteed to maintain high predictive probability. For example, replacing one of the starting tokens can affect the multiplicative score throughout the rest of the sequence. A more elaborate approach is to allow a back-and-forth in which the neural model tries to generate a solution that resembles the current one, e.g., matches a maximal prefix, after each edit. The search can then be terminated once a predefined condition is met, e.g., the edits become too minimal. If the back-and-forth approach is employed, the objective function for the editing stage can be more complicated and based on the history of all generated solutions. It can use similarity to all previously generated segments or optimize lower and upper bound estimates on the sequence score.

A more structured variation of the back-and-forth approach is to use the CSP solver to determine a point to which the solution generation needs to backtrack. A backtrack can be triggered when the generation is terminated or a violation is (belatedly) detected. Note that the backtrack point is optimized to be maximal so as not to disregard any feasible solutions unnecessarily. This approach is effectively a depth-first search to solve the feasibility problem, where the branching heuristic is dictated by the neural model. Deep learning has been utilized towards branching heuristics in the literature, but the purpose is usually to guide the search towards feasible parts of the search space and improve performance [168, 158, 259], as is the case for more traditional heuristics. In this context, however, the neural model heuristic is implicitly optimizing an objective function that is based on the neural model. We can formalize the involvement of the learning component into the search procedure even further by treating it as a constraint that guarantees a lower bound for the predictive quality of a solution. In that case, the procedure could evolve into a more complex task like a constraint programming instance. The neural sequence model could be integrated as a custom constraint, potentially with custom constraint propagation if the model supports non-iterative decoding as well.

7.3.4 Constrained Machine Learning Through Witness Generation

Another approach to constrained ML is to explicitly train a model with constraint satisfaction as an objective [170, 148]. One potential way to enrich this line of work is by using an idea similar to chain-of-thought prompting. In the context of large language models, chain-of-thought refers to a technique where the model is prompted to show its work. Chain-of-thought has been shown to improve the accuracy of reasoning as it divides the task into digestible segments for the model. A similar idea in constrained generation is to demand the neural model to simultaneously generate a witness for why the solution satisfies the given constraint. This approach can enhance the rigorous reasoning capabilities of a deep learning model, but also provide easily verifiable witnesses which help demonstrate

to humans and machines how constraints are satisfied. In the context of interpretability, this approach is reminiscent of the family of approaches that facilitate understanding the behavior by providing additional information, e.g., similar points from training data [58] or parts of an image that most contributed to a classification [245]. An example of this approach would be to satisfy the regular language requirement (presented in Section 6.6) by demanding the neural sequence model to generate the trace of DFA states as a witness. Doing so would require training the model on a set of sequences that are augmented with their corresponding traces of states.

Requiring a neural model to produce a witness as to why its output satisfies a requirement can also be applicable for the reinforcement learning (RL) problem. Constrained ML via relying on witness generation is even more suitable to RL compared to sequence generation, as RL solutions are open-ended policies and not finite sequences which external combinatorial components like cuts in BSC can navigate. Constraints in RL can sometimes be integrated into reward functions and formal specifications for rewards have been explored in recent work by using reward machines [133] that are expressive enough to represent regular languages. Reward machines can be combined with the witness generation approach to extend their support to non-deterministic machines. Namely, the neural model can output the next state that the machine should transition to at each step, resolving the non-determinism between transitions, in addition to selecting actions to perform. Support for non-deterministic machines can enable full LTL [276] specifications as represented by general Büchi automata [99] which are strictly more expressive than their deterministic counterparts. Non-determinism can also significantly reduce the size of the reward machines and the policy as a result, because a deterministic equivalent to a non-deterministic machine can be exponentially larger. Lastly, non-determinism can allow a more well-defined satisfaction of regular specifications in the context of infinite behaviors. Specifically, the model can declare when the finite behavior, that is claimed to belong to the regular language, ends. Contrarily, a deterministic reward machine can reward an agent for satisfying the regular language numerous times, even if the intended point of termination yields a non-accepting sequence of actions.

Appendix A

Revised MIP Encoding of Decision Tree Clustering Baseline

In this section, we present a revised version of the decision tree clustering MIP formulation presented in Bertsimas, Orfanoudaki, and Wiberg to be used as a baseline against our approach presented in Section 5.4. We change the original model to maintain notational consistency, fix typographical mistakes, address various logical issues that are crucial for one's ability to solve the model, and extend the model with support for pairwise constraints. The variables used to represent different aspects of the model are as follows:

- s_i: The $s(i)$ variables in the Silhouette objective.
- q_i: The $b(i)$ variables in the Silhouette objective, namely the average distance between points and their second closest cluster.
- r_i: The $a(i)$ variables in the Silhouette objective, namely the average distance between points and their cluster.
- m_i: $max(b(i), a(i))$ in the Silhouette objective.
- γ_{it} [Binary]: Cluster (leaf) t is the 2nd closest cluster to x_i.
- c_{it}: The distance between x_i and cluster (leaf) t.
- z_{it} [Binary]: x_i belongs to cluster (leaf) t.
- K_t: The size of cluster (leaf) t.
- a_{jt} [Binary]: Feature j is chosen at branching node t.
- d_t [Binary]: Branching node t is activated.

- b_t: Threshold at branching node t.
- l_t [Binary]: Leaf node t is activated.

The following set of linear and quadratic constraints model the objective (Eq. (A.1)), the relation between Silhouette values (Eq. (A.2) to Eq. (A.4)), the selection and distance to the second closest cluster (Eq. (A.5) to Eq. (A.7)), the distance to each cluster (Eq. (A.8) and Eq. (A.9)), the size of each cluster (Eq. (A.10) and Eq. (A.11)), the activation of branching nodes (Eq. (A.12) to Eq. (A.14)), the activation of leaf nodes (Eq. (A.15) and Eq. (A.16)), and the branchings leading up to appearance at leaves (Eq. (A.17)-(A.19)). Note that N_{min} specifies the minimum number of points appearing at each active leaf (minimum support). See Bertsimas, Orfanoudaki, and Wiberg [38] for a detailed discussion on the role of each constraint. The clauses in Eq. (A.20) and Eq. (A.21) are new to the model and guarantee the satisfaction of must-links and cannot-links, respectively. The range of integer and real variables are specified in Eq. (A.22) to Eq. (A.25).

$$\text{minimize} \quad -\frac{1}{|X|}\sum_{x_i \in X} s_i \tag{A.1}$$

$$\text{subject to} \quad s_i m_i = q_i - r_i, \qquad \forall x_i \in X \tag{A.2}$$

$$m_i \geq q_i, \qquad \forall x_i \in X \tag{A.3}$$

$$m_i \geq r_i, \qquad \forall x_i \in X \tag{A.4}$$

$$q_i = \sum_{t \in \mathcal{T}_L} \gamma_{it} c_{it}, \qquad \forall x_i \in X \tag{A.5}$$

$$\sum_{t \in \mathcal{T}_L} \gamma_{it} = 1, \qquad \forall x_i \in X \tag{A.6}$$

$$\gamma_{it} \leq (1 - z_{it}), \qquad \forall x_i \in X, t \in \mathcal{T}_L \tag{A.7}$$

$$r_i = \sum_{t \in \mathcal{T}_L} c_{it} z_{it}, \qquad \forall x_i \in X, t \in \mathcal{T}_L \tag{A.8}$$

$$c_{it} K_t = \sum_{x_j \in X} z_{jt} |x_i - x_j|, \qquad \forall x_i \in X, t \in \mathcal{T}_L \tag{A.9}$$

$$\gamma_{it} \leq K_t, \qquad \forall x_i \in X, t \in \mathcal{T}_L \tag{A.10}$$

$$K_t = \sum_{x_i \in X} z_{it} \qquad \forall t \in \mathcal{T}_L \tag{A.11}$$

$$\sum_{j \in F} a_{jt} = d_t, \qquad \forall t \in \mathcal{T}_B \tag{A.12}$$

$$0 \leq b_t \leq 100 d_t, \qquad \forall t \in \mathcal{T}_B \tag{A.13}$$

$$d_t \leq d_{p(t)}, \qquad \forall t \in \mathcal{T}_B - \{1\} \tag{A.14}$$

$$z_{it} \leq l_t, \qquad \forall t \in \mathcal{T}_L \tag{A.15}$$

$$\sum_{x_i \in X} z_{it} \geq N_{min} l_t, \qquad \forall t \in \mathcal{T}_L \tag{A.16}$$

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \qquad \forall x_i \in X \tag{A.17}$$

$$a_m^T x_i \geq b_m - 100(1 - z_{it}), \qquad \forall x_i \in X, t \in \mathcal{T}_L, m \in A_R(t) \tag{A.18}$$

$$a_m^T x_i + \epsilon_s \leq b_m + (100 + \epsilon_s)(1 - z_{it}), \qquad \forall x_i \in X, t \in \mathcal{T}_L, m \in A_L(t) \tag{A.19}$$

$$z_{it} = z_{jt}, \qquad \forall t \in \mathcal{T}_L, (i,j) \in ML \tag{A.20}$$

$$\sum_{t \in \mathcal{T}_L} (z_{it} - z_{jt})(z_{it} - z_{jt}) \geq 1, \qquad \forall (i,j) \in CL \tag{A.21}$$

$$a_{jt}, d_t \in \{0,1\}, \qquad \forall j \in F, t \in \mathcal{T}_B \tag{A.22}$$

$$z_{it}, l_t \in \{0,1\}, \qquad \forall x_i \in X, t \in \mathcal{T}_L \tag{A.23}$$

$$\gamma_{it} \in \{0,1\}, \qquad \forall x_i \in X, t \in \mathcal{T}_L \tag{A.24}$$

$$b_t \leq 100, \qquad \forall t \in \mathcal{T}_B \tag{A.25}$$

The model presented above is different from the original one in various ways. We have categorized our modifications into three groups based on their nature and significance:

- Changes in the notation:
  - Iterating over points: $i = 1, ..., n$ to $x_i \in X$
  - Iterating over features: $j = 1, ..., p$ to $f \in F$

  - Distance between two points: d_{ij} to $|x_i - x_j|$
  - The epsilon value used in modelling branchings (to avoid confusion with the epsilon used for approximating objective in our method): ϵ to ϵ_s

- Typographical errors:
  - The first dimension of the z_{it} in Eq. (A.23): p to n
  - Sum index in Eq. (A.8): $\forall t \in T_L$ to $t \in T_L$
  - Branching nodes in Eq. (A.18) and Eq. (A.19): T_B to T_L
  - Index of b variables in Eq. (A.18) and Eq. (A.19): b_t to b_m

- Logical issues of the model:
  - The big M constant in Eq. (A.7) serves no purpose: removed M.
  - Gurobi does not allow division to be used in Eq. (A.2) and Eq. (A.9): replaced divisions with their equivalent multiplication. Note that the divided-by-zero cases should be monitored to maintain the correct semantics.
  - Eq. (A.5) allowing q_i to be arbitrarily large will trivialize the objective: changed $\geq$ to $=$.
  - Not having Eq. (A.10) will allow empty clusters to be selected as the second closest cluster: added Eq. (A.10).
  - Since ϵ_s is multiplied by a_m^T on the left hand side of Eq. (A.19), it is possible for points to be directed to both left and right children, if a branching node is disabled: changed $a_m^T(x_i + \epsilon_s)$ to $a_m^T x_i + \epsilon_s$.
  - Not having a constant upper bound for b_t variables causes runtime issues: added Eq. (A.25).

- Modifications to match our model:
  - The original model does not support must-links and cannot-links: added Eqs. (A.20) and (A.21).
  - Constraints Eq. (A.13), Eq. (A.18), and Eq. (A.19) need to be modified for a feature normalization to $[0, 100]$ rather than $[0, 1]$: changed d_t to $100d_t$, $(1 - z_{it})$ to $100(1 - z_{it})$, and $(1 + \epsilon_s)$ to $(100 + \epsilon_s)$, respectively.

Bibliography

[1] Tobias Achterberg and Roland Wunderling. "Mixed integer programming: Analyzing 12 years of progress". In: *Facets of combinatorial optimization: Festschrift for martin grötschel*. Springer, 2013, pp. 449–481.

[2] Stefan Aeberhard, Danny Coomans, and Olivier De Vel. "Comparative analysis of statistical pattern recognition methods in high dimensional settings". In: *Pattern Recognition* 27.8 (1994), pp. 1065–1077.

[3] Constructions Aeronautiques et al. "Pddl— the planning domain definition language". In: *Technical Report, Tech. Rep.* (1998).

[4] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. "Learning optimal and fair decision trees for non-discriminative decision-making". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 1418–1426.

[5] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. "Assessing optimal forests of decision trees". In: *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2021, pp. 32–39.

[6] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. "Learning optimal decision trees using caching branch-and-bound search". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 3146–3153.

[7] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. "Pydl8. 5: a library for learning optimal decision trees". In: *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*. 2021, pp. 5222–5224.

[8] Prachi Agrawal et al. "Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019)". In: *Ieee Access* 9 (2021), pp. 26766–26791.

[9] Sheldon B. Akers. "Binary decision diagrams". In: *IEEE Transactions on computers* 27.06 (1978), pp. 509–516.

[10] Par Alizadeh Alamdari et al. "Formal methods with a touch of magic". In: *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design*. 2020.

[11] Adriana Albu. "From logical inference to decision trees in medical diagnosis". In: *2017 E-Health and Bioengineering Conference (EHB)*. IEEE. 2017, pp. 65–68.

[12] B Amarnath, S Balamurugan, and Appavu Alias. "Review on feature selection techniques and its impact for effective data classification using UCI machine learning repository dataset". In: *Journal of Engineering Science and Technology* 11.11 (2016), pp. 1639–1646.

[13] Peter Anderson et al. "Guided Open Vocabulary Image Captioning with Constrained Beam Search". In: *EMNLP*. 2017, pp. 936–945.

[14] Robert Andrews, Joachim Diederich, and Alan B. Tickle. "Survey and critique of techniques for extracting rules from trained artificial neural networks". In: *Knowl. Based Syst.* 8 (1995), pp. 373–389. URL: `https://api.semanticscholar.org/CorpusID:41520404`.

[15] Elaine Angelino et al. "Learning Certifiably Optimal Rule Lists for Categorical Data". In: *Journal of Machine Learning Research* 18 (2018), pp. 1–78.

[16] Krzysztof Apt. *Principles of constraint programming*. Cambridge university press, 2003.

[17] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information fusion* 58 (2020), pp. 82–115.

[18] Ph Augerat et al. "Computational results with a branch and cut code for the capacitated vehicle routing problem research". In: *Universite Joseph Fourier, Grenoble, France* (1995), pp. 949–M.

[19] Florent Avellaneda. "Efficient inference of optimal decision trees". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2020, pp. 3195–3202.

[20] Mohamed Azmi, George C Runger, and Abdelaziz Berrado. "Interpretable regularized class association rules algorithm for classification in a categorical data space". In: *Information Sciences* 483 (2019), pp. 313–331.

[21] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. "Constrained clustering using column generation". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2014, pp. 438–454.

[22] Sebastian Bader and Pascal Hitzler. "Dimensions of neural-symbolic integration-a structured survey". In: *arXiv preprint cs/0511042* (2005).

[23] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *ICLR*. 2015.

[24] Olivier Bailleux and Yacine Boufkhad. "Efficient CNF encoding of boolean cardinality constraints". In: *International conference on principles and practice of constraint programming*. Springer. 2003, pp. 108–122.

[25] Egon Balas and Robert Jeroslow. "Canonical cuts on the unit hypercube". In: *SIAM Journal on Applied Mathematics* 23.1 (1972), pp. 61–69.

[26] Roberto Baldacci, Paolo Toth, and Daniele Vigo. “Recent advances in vehicle routing exact algorithms”. In: *4or* 5 (2007), pp. 269–298.

[27] Clark Barrett and Cesare Tinelli. “Satisfiability modulo theories”. In: *Handbook of model checking* (2018), pp. 305–343.

[28] Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained clustering: Advances in algorithms, theory, and applications*. Chapman and Hall/CRC, 2008.

[29] Irwan Bello et al. “Neural combinatorial optimization with reinforcement learning”. In: *arXiv preprint arXiv:1611.09940* (2016).

[30] Trevor JM Bench-Capon. *Knowledge representation: An approach to artificial intelligence*. Vol. 32. Elsevier, 2014.

[31] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. “Machine learning for combinatorial optimization: a methodological tour d’horizon”. In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[32] Kristin P Bennett. “Global tree optimization: A non-greedy decision tree algorithm”. In: *Journal of Computing Science and Statistics* 26 (1994), pp. 156–160.

[33] Jeremias Berg, Fahiem Bacchus, and Alex Poole. “Abstract cores in implicit hitting set MaxSat solving”. In: *Theory and Applications of Satisfiability Testing–SAT 2020: 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings 23*. Springer. 2020, pp. 277–294.

[34] Jeremias Berg, Emir Demirović, and Peter J Stuckey. “Core-boosted linear search for incomplete MaxSAT”. In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. Springer. 2019, pp. 39–56.

[35] Jeremias Berg and Matti Järvisalo. “Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability”. In: *Artificial Intelligence* 244 (2017), pp. 110–142.

[36] Felix Berkenkamp, Andreas Krause, and Angela P Schoellig. “Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics”. In: *Machine Learning* 112.10 (2023), pp. 3713–3747.

[37] Dimitris Bertsimas and Jack Dunn. “Optimal classification trees”. In: *Machine Learning* 106.7 (2017), pp. 1039–1082.

[38] Dimitris Bertsimas, Agni Orfanoudaki, and Holly Wiberg. “Interpretable clustering: an optimization approach”. In: *Machine Learning* 110 (2021), pp. 89–138.

[39] Christian Bessiere, Emmanuel Hebrard, and Barry O’Sullivan. “Minimising decision tree size as combinatorial optimisation”. In: *International Conference on Principles and Practice of Constraint Programming (CP)*. Springer. 2009, pp. 173–187.

[40] Michael Biehl, Barbara Hammer, and Thomas Villmann. "Prototype-based models in machine learning". In: *Wiley Interdisciplinary Reviews: Cognitive Science* 7.2 (2016), pp. 92–111.

[41] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*. Vol. 185. IOS press, 2009.

[42] Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. "Integrating constraints and metric learning in semi-supervised clustering". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 11.

[43] Maxim Binshtok et al. "Generic preferences over subsets of structured objects". In: *JAIR* 34 (2009), pp. 133–164.

[44] Robert E Bixby. "A brief history of linear and mixed-integer programming computation". In: *Documenta Mathematica* 2012 (2012), pp. 107–121.

[45] Marko Bohanec and Vladislav Rajkovic. "Knowledge acquisition and explanation for multi-attribute decision making". In: *8th intl workshop on expert systems and their applications*. Avignon France. 1988, pp. 59–78.

[46] Rafael V. Borges, Artur S. d'Avila Garcez, and L. Lamb. "Learning and Representing Temporal Knowledge in Recurrent Networks". In: *IEEE Transactions on Neural Networks* 22 (2011), pp. 2409–2421. URL: `https://api.semanticscholar.org/CorpusID:8409422`.

[47] Allan Borodin, Alexander Razborov, and Roman Smolensky. "On lower bounds for read-k-times branching programs". In: *Computational Complexity* 3 (1993), pp. 1–18.

[48] Ronen I Brafman et al. "Preferences over sets". In: *AAAI*. Vol. 6. 2006, pp. 1101–1106.

[49] Sally C Brailsford, Chris N Potts, and Barbara M Smith. "Constraint satisfaction problems: Algorithms and applications". In: *EJOR* 119.3 (1999), pp. 557–581.

[50] Leo Breiman et al. *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[51] Michael J Brusco and Douglas Steinley. "A comparison of heuristic procedures for minimum within-cluster sums of squares partitioning". In: *Psychometrika* 72 (2007), pp. 583–600.

[52] Randal E Bryant. "Graph-based algorithms for boolean function manipulation". In: *Computers, IEEE Transactions on* 100.8 (1986), pp. 677–691.

[53] Gianpiero Cabodi et al. "Optimizing binary decision diagrams for interpretable machine learning classification". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1122–1125.

[54] Gianpiero Cabodi et al. "Optimizing binary decision diagrams for interpretable machine learning classification". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[55] Alberto Camacho and Sheila A McIlraith. "Learning interpretable models expressed in linear temporal logic". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 621–630.

[56] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. "Machine learning interpretability: A survey on methods and metrics". In: *Electronics* 8.8 (2019), p. 832.

[57] Félix Chalumeau et al. "Seapearl: A constraint programming solver guided by reinforcement learning". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*. Springer. 2021, pp. 392–409.

[58] Chaofan Chen et al. "This looks like that: deep learning for interpretable image recognition". In: *Advances in neural information processing systems* 32 (2019).

[59] Edmund M Clarke, Masahiro Fujita, and Xudong Zhao. "Multi-terminal binary decision diagrams and hybrid decision diagrams". In: *Representations of discrete functions* (1996), pp. 93–108.

[60] Eldan Cohen. "Interpretable Clustering via Soft Clustering Trees". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2023, pp. 281–298.

[61] Eldan Cohen and Christopher Beck. "Empirical analysis of beam search performance degradation in neural sequence models". In: *ICML*. PMLR. 2019, pp. 1290–1299.

[62] Eldan Cohen and J Christopher Beck. "Fat-and heavy-tailed behavior in satisficing planning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[63] Eldan Cohen and J Christopher Beck. "Heavy-Tails and Randomized Restarting Beam Search in Goal-Oriented Neural Sequence Decoding". In: *CPAIOR*. Springer. 2021, pp. 115–132.

[64] Eldan Cohen, Arik Senderovich, and J Christopher Beck. "An Ising framework for constrained clustering on special purpose hardware". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2020, pp. 130–147.

[65] Gal Dalal et al. "Safe exploration in continuous action spaces". In: *arXiv preprint arXiv:1801.08757* (2018).

[66] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. "Constrained clustering by constraint programming". In: *Artificial Intelligence* 244 (2017), pp. 70–94.

[67] Thi-Bich-Hanh Dao et al. "A framework for actionable clustering using constraint programming". In: *Proceedings of the Twenty-second European Conference on Artificial Intelligence*. 2016, pp. 453–461.

[68] Adnan Darwiche. "SDD: A new canonical representation of propositional knowledge bases". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[69] Ian Davidson and SS Ravi. "Agglomerative hierarchical clustering with constraints: Theoretical and empirical results". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2005, pp. 59–70.

[70] Ian Davidson and SS Ravi. "Clustering with constraints: Feasibility issues and the k-means algorithm". In: *Proceedings of the 2005 SIAM international conference on data mining*. SIAM. 2005, pp. 138–149.

[71] Ian Davidson, SS Ravi, and Leonid Shamis. "A SAT-based framework for efficient constrained clustering". In: *Proceedings of the 2010 SIAM international conference on data mining*. SIAM. 2010, pp. 94–105.

[72] Jessica Davies and Fahiem Bacchus. "Solving MAXSAT by solving a sequence of simpler SAT instances". In: *International conference on principles and practice of constraint programming*. Springer. 2011, pp. 225–239.

[73] Giuseppe De Giacomo and Marco Favorito. "Compositional approach to translate LTLf/LDLf into deterministic finite automata". In: *ICAPS*. Vol. 31. 2021, pp. 122–130.

[74] Emir Demirović et al. "Murtree: Optimal decision trees via dynamic programming and search". In: *Journal of Machine Learning Research* 23.26 (2022), pp. 1–47.

[75] Shi Dong, Ping Wang, and Khushnood Abbas. "A survey on deep learning and its applications". In: *Computer Science Review* 40 (2021), p. 100379.

[76] Rolf Drechsler, Nicole Drechsler, and Wolfgang Günther. "Fast exact minimization of BDDs". In: *Proceedings of the 35th Annual Design Automation Conference*. 1998, pp. 200–205.

[77] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[78] Joseph C Dunn. "Well-separated clusters and optimal fuzzy partitions". In: *Journal of cybernetics* 4.1 (1974), pp. 95–104.

[79] Jennifer G Dy and Carla E Brodley. "Feature selection for unsupervised learning". In: *Journal of machine learning research* 5.Aug (2004), pp. 845–889.

[80] Niklas Eén and Niklas Sörensson. "An extensible SAT-solver". In: *SAT*. Springer. 2003, pp. 502–518.

[81] Niklas Eén and Niklas Sörensson. *Minisat SAT solver*. 2003. URL: `http://minisat.se/Main.html`.

[82] Enes Eryarsoy, Gary J Koehler, and Haldun Aytug. "Using domain-specific knowledge in generalization error bounds for support vector machine learning". In: *Decision Support Systems* 46.2 (2009), pp. 481–491.

[83] Dieqiao Feng, Carla P Gomes, and Bart Selman. "Left Heavy Tails and the Effectiveness of the Policy and Value Networks in DNN-based best-first search for Sokoban Planning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36295–36307.

[84] Alexandre M Florio et al. "Optimal decision diagrams for classification". In: *arXiv preprint arXiv:2205.14500* (2022).

[85] Piotr Formanowicz and Krzysztof Tanaś. "A survey of graph coloring-its types, methods and applications". In: *Foundations of Computing and Decision Sciences* 37.3 (2012), pp. 223–238.

[86] Manoel Vitor Macedo França, Gerson Zaverucha, and Artur S. d'Avila Garcez. "Fast relational learning using bottom clause propositionalization with artificial neural networks". In: *Machine Learning* 94 (2013), pp. 81 –104. URL: https://api.semanticscholar.org/CorpusID:14682074.

[87] Markus Freitag and Yaser Al-Onaizan. "Beam Search Strategies for Neural Machine Translation". In: *WNMT*. Ed. by Thang Luong et al. 2017, pp. 56–60.

[88] Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. "ExKMC: Expanding Explainable k-Means Clustering". In: *arXiv preprint arXiv:2006.02399* (2020).

[89] Zhaohui Fu and Sharad Malik. "On solving the partial MAX-SAT problem". In: *International Conference on Theory and Applications of Satisfiability Testing (SAT)*. Springer. 2006, pp. 252–265.

[90] Magzhan Gabidolla and Miguel Á Carreira-Perpiñán. "Optimal interpretable clustering using oblique decision trees". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 400–410.

[91] Buddhima Gamlath et al. "Nearly-tight and oblivious algorithms for explainable clustering". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28929–28939.

[92] A Garcez et al. "Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning". In: *Journal of Applied Logics* 6.4 (2019), pp. 611–632.

[93] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. "Neural-symbolic learning systems - foundations and applications". In: *Perspectives in Neural Computing*. 2012. URL: https://api.semanticscholar.org/CorpusID:60656463.

[94] Artur S. d'Avila Garcez and L. Lamb. "Reasoning about Time and Knowledge in Neural Symbolic Learning Systems". In: *Neural Information Processing Systems*. 2003. URL: https://api.semanticscholar.org/CorpusID:12821013.

[95] Artur S. d'Avila Garcez and Luís C. Lamb. "A Connectionist Computational Model for Epistemic and Temporal Reasoning". In: *Neural Computation* 18 (2006), pp. 1711–1738. URL: https://api.semanticscholar.org/CorpusID:9626875.

[96] Artur S. d'Avila Garcez and Gerson Zaverucha. "The Connectionist Inductive Learning and Logic Programming System". In: *Applied Intelligence* 11 (1999), pp. 59–77. URL: https://api.semanticscholar.org/CorpusID:21039761.

[97] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979.

[98] Minos Garofalakis et al. "Building decision trees with constraints". In: *Data Mining and Knowledge Discovery* 7 (2003), pp. 187–214.

[99] Paul Gastin and Denis Oddoux. "Fast LTL to Büchi automata translation". In: *Computer Aided Verification: 13th International Conference, CAV 2001 Paris, France, July 18–22, 2001 Proceedings 13*. Springer. 2001, pp. 53–65.

[100] Héctor Geffner. "Functional STRIPS: a more flexible language for planning and problem solving". In: *Logic-based artificial intelligence* (2000), pp. 187–209.

[101] Arthur M Geoffrion. "Generalized benders decomposition". In: *Journal of optimization theory and applications* 10 (1972), pp. 237–260.

[102] Seyed Mohssen Ghafari and Christos Tjortjis. "A survey on association rules mining using heuristics". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9.4 (2019), e1307.

[103] Paul C Gilmore and Ralph E Gomory. "A linear programming approach to the cutting-stock problem". In: *Operations research* 9.6 (1961), pp. 849–859.

[104] Sean Gilpin, Siegried Nijssen, and Ian Davidson. "Formalizing hierarchical clustering as integer linear programming". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 1. 2013, pp. 372–378.

[105] Carla P Gomes et al. "Heavy-tailed phenomena in satisfiability and constraint satisfaction problems". In: *Journal of automated reasoning* 24.1 (2000), pp. 67–100.

[106] Weiwei Gong and Xu Zhou. "A survey of SAT solver". In: *AIP Conference Proceedings*. Vol. 1836. AIP Publishing. 2017.

[107] Marian B Gorzałczany and Filip Rudziński. "Interpretable and accurate medical data classification–a multi-objective genetic-fuzzy optimization approach". In: *Expert Systems with Applications* 71 (2017), pp. 26–39.

[108] Constantine Goulimis. "Optimal solutions for the cutting stock problem". In: *European Journal of Operational Research* 44.2 (1990), pp. 197–208.

[109] Riccardo Guidotti et al. "Factual and counterfactual explanations for black box decision making". In: *IEEE Intelligent Systems* 34.6 (2019), pp. 14–23.

[110] Riccardo Guidotti et al. "Local Rule-Based Explanations of Black Box Decision Systems". In: *CoRR* abs/1805.10820 (2018). arXiv: 1805.10820. URL: http://arxiv.org/abs/1805.10820.

[111] Oktay Günlük et al. "Optimal decision trees for categorical data via integer programming". In: *Journal of Global Optimization* (2021), pp. 1–28.

[112] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: `https://www.gurobi.com`.

[113] Kelvin Guu et al. "Generating Sentences by Editing Prototypes". In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 437–450.

[114] Isabelle Guyon. "Design of experiments of the NIPS 2003 variable selection benchmark". In: *NIPS 2003 workshop on feature extraction and feature selection*. Vol. 253. 2003.

[115] Isabelle Guyon et al. "Design of the 2015 chalearn automl challenge". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[116] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. "ManySAT: a parallel SAT solver". In: *Journal on Satisfiability, Boolean Modeling and Computation* 6.4 (2010), pp. 245–262.

[117] Thomas Hancock et al. "Lower bounds on learning decision lists and trees". In: *Information and Computation* 126.2 (1996), pp. 114–122.

[118] GM Harshvardhan et al. "A comprehensive survey and analysis of generative models in machine learning". In: *Computer Science Review* 38 (2020), p. 100285.

[119] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[120] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[121] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282.

[122] Chris Hokamp and Qun Liu. "Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search". In: *ACL*. Vol. 1. 2017, pp. 1535–1546.

[123] Hao Hu. "Interpretable Machine Learning Models via Maximum Boolean Satisfiability". PhD thesis. INSA de Toulouse, 2022.

[124] Hao Hu, Marie-José Huguet, and Mohamed Siala. "Optimizing Binary Decision Diagrams with MaxSAT for Classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 4. 2022, pp. 3767–3775.

[125] Hao Hu, Marie-José Huguet, and Mohamed Siala. *Optimizing Binary Decision Diagrams with MaxSAT for classification*. 2022. arXiv: `2203.11386 [cs.AI]`.

[126] Hao Hu et al. "Learning optimal decision trees with MaxSAT and its integration in AdaBoost". In: *International Joint Conference on Artificial Intelligence and Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI)*. 2020.

[127] Xia Hu et al. "Model complexity of deep learning: A survey". In: *Knowledge and Information Systems* 63 (2021), pp. 2585–2619.

[128] Yaojie Hu et al. “Fix Bugs with Transformer through a Neural-Symbolic Edit Grammar”. In: *ArXiv* abs/2204.06643 (2022). URL: https://api.semanticscholar.org/CorpusID:248177769.

[129] Hao Hua et al. “Integer programming for urban design”. In: *European Journal of Operational Research* 274.3 (2019), pp. 1125–1137.

[130] Lingying Huang et al. “Branch and bound in mixed integer linear programming problems: A survey of techniques and trends”. In: *arXiv preprint arXiv:2111.06257* (2021).

[131] Xuanxiang Huang and João Marques-Silva. “From Decision Trees to Explained Decision Sets.” In: *ECAI*. 2023, pp. 1100–1108.

[132] Lawrence Hubert and Phipps Arabie. “Comparing partitions”. In: *Journal of Classification* 2.1 (1985), pp. 193–218.

[133] Rodrigo Toro Icarte et al. “Reward machines: Exploiting reward function structure in reinforcement learning”. In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 173–208.

[134] Alexey Ignatiev and Joao Marques-Silva. “SAT-based rigorous explanations for decision lists”. In: *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*. Springer. 2021, pp. 251–269.

[135] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. “PySAT: A Python Toolkit for Prototyping with SAT Oracles”. In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.

[136] Alexey Ignatiev et al. “A SAT-based approach to learn explainable decision sets”. In: *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 9*. Springer. 2018, pp. 627–645.

[137] Alexey Ignatiev et al. “A scalable two stage approach to computing optimal decision sets”. In: *arXiv preprint arXiv:2102.01904* (2021).

[138] Alexey Ignatiev et al. “From contrastive to abductive explanations and back again”. In: *International Conference of the Italian Association for Artificial Intelligence*. Springer. 2020, pp. 335–355.

[139] Alexey Ignatiev et al. “Reasoning-Based Learning of Interpretable ML Models”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2021, in press.

[140] Dmitry Ignatov and Andrey Ignatov. “Decision stream: Cultivating deep decision trees”. In: *2017 ieee 29th international conference on tools with artificial intelligence (ictai)*. IEEE. 2017, pp. 905–912.

[141] Nagisa Ishiura, Hiroshi Sawada, and Shuzo Yajima. "Minimazation of Binary Decision Diagrams Based on Exchanges of Variables." In: *ICCAD*. Vol. 91. 1991, pp. 472–475.

[142] Yacine Izza and João Marques Silva. "On Explaining Random Forests with SAT". In: *30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*. International Joint Conferences on Artifical Intelligence (IJCAI). 2021.

[143] Christoph Jabs et al. "MaxSAT-Based Bi-Objective Boolean Optimization". In: *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2022.

[144] Alon Jacovi et al. "Contrastive Explanations for Model Interpretability". In: *CoRR* abs/2103.01378 (2021). arXiv: 2103.01378. URL: https://arxiv.org/abs/2103.01378.

[145] Jawahar Jain, William Adams, and Masahiro Fujita. "Sampling schemes for computing OBDD variable orderings". In: *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*. 1998, pp. 631–638.

[146] Gareth M James, Jing Wang, and Ji Zhu. "Functional Linear Regression That's Interpretable". In: *The Annals of Statistics* 37.5A (2009), pp. 2083–2108.

[147] Mikoláš Janota and António Morgado. "SAT-based encodings for optimal decision trees with explicit paths". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2020, pp. 501–518.

[148] Natasha Jaques et al. "Tuning Recurrent Neural Networks with Reinforcement Learning". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. 2017.

[149] Jiaming Ji et al. "Ai alignment: A comprehensive survey". In: *arXiv preprint arXiv:2310.19852* (2023).

[150] Ellis L Johnson, George L Nemhauser, and Martin WP Savelsbergh. "Progress in linear programming-based algorithms for integer programming: An exposition". In: *Informs journal on computing* 12.1 (2000), pp. 2–23.

[151] Milos Jovanovic et al. "Building interpretable predictive models for pediatric hospital readmission using Tree-Lasso logistic regression". In: *Artificial intelligence in medicine* 72 (2016), pp. 12–21.

[152] Stasys Jukna. "A note on read-times branching programs". In: *RAIRO-Theoretical Informatics and Applications* 29.1 (1995), pp. 75–83.

[153] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. "Discrimination aware decision tree learning". In: *2010 IEEE international conference on data mining*. IEEE. 2010, pp. 869–874.

[154] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. 1984, pp. 302–311.

[155] John D Kelleher, Brian Mac Namee, and Aoife D'arcy. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020.

[156] L.G. Khachiyan. "Polynomial algorithms in linear programming". In: *USSR Computational Mathematics and Mathematical Physics* 20.1 (1980), pp. 53–72. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(80)90061-0. URL: https://www.sciencedirect.com/science/article/pii/0041555380900610.

[157] Elias Khalil et al. "Learning combinatorial optimization algorithms over graphs". In: *Advances in neural information processing systems* 30 (2017).

[158] Elias Khalil et al. "Learning to branch in mixed integer programming". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[159] Donald E Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 2009.

[160] Ron Kohavi. "Bottom-up induction of oblivious read-once decision graphs". In: *Machine Learning: ECML-94: European Conference on Machine Learning Catania, Italy, April 6–8, 1994 Proceedings 7*. Springer. 1994, pp. 154–169.

[161] Ron Kohavi and Chia-Hsin Li. "Oblivious decision trees, graphs, and top-down pruning". In: *IJCAI*. Vol. 2. 1995, pp. 1071–1077.

[162] Wouter Kool, Herke van Hoof, and Max Welling. "Attention, Learn to Solve Routing Problems!" In: *ICLR*. 2018.

[163] Bernhard H Korte et al. *Combinatorial optimization*. Vol. 1. Springer, 2011.

[164] Sotiris Kotsiantis and Dimitris Kanellopoulos. "Association rules mining: A recent overview". In: *GESTS International Transactions on Computer Science and Engineering* 32.1 (2006), pp. 71–82.

[165] Sotiris B Kotsiantis. "Decision trees: a recent overview". In: *Artificial Intelligence Review* 39.4 (2013), pp. 261–283.

[166] R Krishnan, G Sivakumar, and P Bhattacharya. "Extracting decision trees from trained neural networks". In: *Pattern recognition* 32.12 (1999).

[167] Mikio Kubo and Hiroshi Kasugai. "The precedence constrained traveling salesman problem". In: *Journal of the Operations Research Society of Japan* 34.2 (1991), pp. 152–172.

[168] Vitaly Kurin et al. "Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver?" In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9608–9621.

[169] Ryo Kuroiwa and J Christopher Beck. "Domain-independent dynamic programming: Generic state space search for combinatorial optimization". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 33. 1. 2023, pp. 236–244.

[170] Daphné Lafleur, Sarath Chandar, and Gilles Pesant. "Combining Reinforcement Learning and Constraint Programming for Sequence-Generation Tasks with Hard Constraints". In: *CP*. 2022, 30:1–30:16.

[171] Michail G Lagoudakis and Michael L Littman. "Learning to select branching rules in the DPLL procedure for satisfiability". In: *Electronic Notes in Discrete Mathematics* 9 (2001), pp. 344–359.

[172] Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. "Interpretable decision sets: A joint framework for description and prediction". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1675–1684.

[173] Hyafil Laurent and Ronald L Rivest. "Constructing optimal binary decision trees is NP-complete". In: *Information processing letters* 5.1 (1976), pp. 15–17.

[174] Eugene L Lawler and David E Wood. "Branch-and-bound methods: A survey". In: *Operations research* 14.4 (1966), pp. 699–719.

[175] Rafael Lazimy. "Mixed-integer quadratic programming". In: *Mathematical Programming* 22 (1982), pp. 332–349.

[176] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[177] Benjamin Letham et al. "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model". In: (2015).

[178] Breiman LI et al. *Classification and Regression Trees (CART)*. Vol. 40. Sept. 1984, p. 358. DOI: 10.2307/2530946.

[179] Huayang Li et al. "Neural machine translation with noisy lexical constraints". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), pp. 1864–1874.

[180] Wenjun Li et al. "Parameterized algorithms of fundamental NP-hard problems: a survey". In: *Human-centric Computing and Information Sciences* 10 (2020), pp. 1–24.

[181] Vladimir Lifschitz. *Answer set programming*. Vol. 3. Springer Heidelberg, 2019.

[182] Zachary C Lipton. "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." In: *Queue* 16.3 (2018), pp. 31–57.

[183] Hongfu Liu and Yun Fu. "Clustering with partition level side information". In: *2015 IEEE international conference on data mining*. IEEE. 2015, pp. 877–882.

[184] Hongfu Liu, Zhiqiang Tao, and Yun Fu. "Partition level constrained clustering". In: *IEEE transactions on pattern analysis and machine intelligence* 40.10 (2017), pp. 2469–2483.

[185] Jun Liu, Jianhui Chen, and Jieping Ye. "Large-scale sparse logistic regression". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009, pp. 547–556.

[186] Rong Liu et al. "Predicting shareholder litigation on insider trading from financial text: An interpretable deep learning approach". In: *Information & Management* 57.8 (2020), p. 103387.

[187] Zuxin Liu et al. "Constrained decision transformer for offline safe reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 21611–21630.

[188] Andrea Lodi and Giulia Zarpellon. "On learning and branching: a survey". In: *Top* 25 (2017), pp. 207–236.

[189] Gang Lu et al. "A survey on exact algorithms for dominating set related problems in arbitrary graphs". In: *Chinese Journal of Computers* 33.6 (2010), pp. 1073–1087.

[190] Hao Lu, Xingwen Zhang, and Shuang Yang. "A learning-based iterative method for solving vehicle routing problems". In: *ICLR*. 2019.

[191] Maher Maalouf. "Logistic regression in data analysis: an overview". In: *International Journal of Data Analysis Techniques and Strategies* 3.3 (2011), pp. 281–299.

[192] Marcus A Maloof. "Learning when data sets are imbalanced and when costs are unequal and unknown". In: *ICML-2003 workshop on learning from imbalanced data sets II*. Vol. 2. 2003, pp. 2–1.

[193] Joao Marques-Silva and Alexey Ignatiev. "Delivering Trustworthy AI through formal XAI". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 11. 2022, pp. 12342–12350.

[194] Joao Marques-Silva and Inês Lynce. "Towards robust CNF encodings of cardinality constraints". In: *Principles and Practice of Constraint Programming–CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings 13*. Springer. 2007, pp. 483–497.

[195] Joao P Marques-Silva and Karem A Sakallah. "GRASP: A search algorithm for propositional satisfiability". In: *IEEE Transactions on Computers* 48.5 (1999), pp. 506–521.

[196] Ruben Martins, Vasco Manquinho, and Inês Lynce. "Open-WBO: A modular MaxSAT solver". In: Springer. 2014, pp. 438–445.

[197] Kenneth L McMillan. "Bayesian Interpolants as Explanations for Neural Inferences". In: *arXiv preprint arXiv:2004.04198* (2020).

[198] Larry R Medsker and LC Jain. "Recurrent neural networks". In: *Design and Applications* 5.64-67 (2001), p. 2.

[199] Ninareh Mehrabi et al. "A Survey on Bias and Fairness in Machine Learning". In: *ACM Computing Surveys (CSUR)* 54 (2019), pp. 1 –35. URL: https://api.semanticscholar.org/CorpusID:201666566.

[200] Ning Miao et al. "Cgmh: Constrained sentence generation by metropolis-hastings sampling". In: *AAAI*. Vol. 33. 2019, pp. 6834–6842.

[201] Tom M Mitchell. "Does machine learning really work?" In: *AI magazine* 18.3 (1997), pp. 11–11.

[202] Bernard ME Moret. "Decision trees and diagrams". In: *ACM Computing Surveys (CSUR)* 14.4 (1982), pp. 593–623.

[203] Michal Moshkovitz et al. "Explainable k-means and k-medians clustering". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7055–7065.

[204] Lawrence Mosley. "A balanced approach to the multi-class imbalance problem". PhD thesis. Iowa State University, 2013.

[205] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[206] W James Murdoch et al. "Definitions, methods, and applications in interpretable machine learning". In: *Proceedings of the National Academy of Sciences* 116.44 (2019), pp. 22071–22080.

[207] Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. "Ultimately incremental SAT". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2014, pp. 206–218.

[208] Géraldin Nanfack, Paul Temple, and Benoît Frénay. "Constraint enforcement on decision trees: A survey". In: *ACM Computing Surveys (CSUR)* 54.10s (2022), pp. 1–36.

[209] Nina Narodytska et al. "Learning Optimal Decision Trees with SAT." In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2018, pp. 1362–1368.

[210] MohammadReza Nazari et al. "Reinforcement Learning for Solving the Vehicle Routing Problem". In: *NeurIPS*. 2018, pp. 9861–9871.

[211] Daniel A Neira et al. "New compact integer programming formulations for the multi-trip vehicle routing problem with time windows". In: *Computers & Industrial Engineering* 144 (2020), p. 106399.

[212] George L Nemhauser. "Integer programming: The global impact". In: (2013).

[213] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. "Abstract DPLL and abstract DPLL modulo theories". In: *International Conference on Logic for Programming Artificial Intelligence and Reasoning*. Springer. 2005, pp. 36–50.

[214] Artur Niewiadomski et al. “Applying modern sat-solvers to solving hard problems”. In: *Fundamenta Informaticae* 165.3-4 (2019), pp. 321–344.

[215] Siegfried Nijssen and Elisa Fromont. “Mining optimal decision trees from itemset lattices”. In: *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2007, pp. 530–539.

[216] Siegfried Nijssen and Elisa Fromont. “Optimal constraint-based decision tree induction from itemset lattices”. In: *Data Mining and Knowledge Discovery* 21.1 (2010), pp. 9–51.

[217] Nils J Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[218] Laurent Orseau and Levi HS Lelis. “Policy-guided heuristic search with guarantees”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 14. 2021, pp. 12382–12390.

[219] Emilio Parisotto et al. “Neuro-Symbolic Program Synthesis”. In: *ArXiv* abs/1611.01855 (2016). URL: `https://api.semanticscholar.org/CorpusID:15904815`.

[220] Peter S Park et al. “AI deception: A survey of examples, risks, and potential solutions”. In: *Patterns* 5.5 (2024).

[221] Damian Pascual et al. “Directed beam search: Plug-and-play lexically constrained language generation”. In: *arXiv preprint arXiv:2012.15416* (2020).

[222] Santiago Paternain et al. “Constrained reinforcement learning has zero duality gap”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[223] Miguel Patrício et al. “Using Resistin, glucose, age and BMI to predict the presence of breast cancer”. In: *BMC cancer* 18 (2018), pp. 1–8.

[224] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[225] Dan Pelleg and Dorit Baras. “K-means with large and noisy constraint sets”. In: *European Conference on Machine Learning*. Springer. 2007, pp. 674–682.

[226] Gregory Plumb, Denali Molitor, and Ameet S Talwalkar. “Model agnostic supervised local explanations”. In: *Advances in neural information processing systems* 31 (2018).

[227] Matt Post and David Vilar. “Fast Lexically Constrained Decoding with Dynamic Beam Allocation for Neural Machine Translation”. In: *NAACL*. 2018, pp. 1314–1324.

[228] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. “A comparative study of categorical variable encoding techniques for neural network classifiers”. In: *International journal of computer applications* 175.4 (2017), pp. 7–9.

[229] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[230] J Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[231] Francesco Ranzato, Caterina Urban, and Marco Zanella. “Fair training of decision tree classifiers”. In: *arXiv preprint arXiv:2101.00909* (2021).

[232] Machel Reid, Vincent Josua Hellendoorn, and Graham Neubig. “Diffuser: Diffusion via edit-based reconstruction”. In: *The Eleventh International Conference on Learning Representations*. 2023.

[233] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[234] Ronald L Rivest. “Learning decision lists”. In: *Machine learning* 2 (1987), pp. 229–246.

[235] Aaron M Roth et al. “Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy”. In: *arXiv preprint arXiv:1907.01180* (2019).

[236] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[237] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat Mach Intell 1: 206–215”. In: *DOI: https://doi. org/10.1038/s42256-019-0048-x* (2019).

[238] Cynthia Rudin and Şeyda Ertekin. “Learning customized and optimized lists of rules with mathematical programming”. In: *Mathematical Programming Computation* 10.4 (2018), pp. 659–702.

[239] Stuart Russell et al. “Ethics of artificial intelligence”. In: *Nature* 521.7553 (2015), pp. 415–416.

[240] Omer Sagi and Lior Rokach. “Approximating XGBoost with an interpretable decision tree”. In: *Information sciences* 572 (2021), pp. 522–542.

[241] Harvey M Salkin and Cornelis A De Kluyver. “The knapsack problem: a survey”. In: *Naval Research Logistics Quarterly* 22.1 (1975), pp. 127–144.

[242] DR Sarvamangala and Raghavendra V Kulkarni. “Convolutional neural networks in medical image understanding: a survey”. In: *Evolutionary intelligence* 15.1 (2022), pp. 1–22.

[243] Petr Savickỳ and Ingo Wegener. “Efficient algorithms for the transformation between different types of binary decision diagrams”. In: *Acta Informatica* 34.4 (1997), pp. 245–256.

[244] Tadeusz Sawik. “Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers”. In: *Mathematical and Computer Modelling* 31.13 (2000), pp. 39–52.

[245] Ramprasaath R Selvaraju et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[246] Luciano Serafini, Ivan Donadello, and Artur S. d'Avila Garcez. "Learning and reasoning in logic tensor networks: theory and application to semantic image interpretation". In: *Proceedings of the Symposium on Applied Computing* (2017). URL: https://api.semanticscholar.org/CorpusID:8663914.

[247] Lei Sha. "Gradient-guided unsupervised lexically constrained text generation". In: *EMNLP*. 2020, pp. 8692–8703.

[248] Junming Shao et al. "Robust prototype-based learning on data streams". In: *IEEE Transactions on Knowledge and Data Engineering* 30.5 (2017), pp. 978–991.

[249] Pouya Shati, Eldan Cohen, and Sheila McIlraith. "Neural Sequence Generation with Constraints via Beam Search with Cuts: A Case Study on VRP". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 17. 2024, pp. 118–126.

[250] Pouya Shati, Eldan Cohen, and Sheila McIlraith. "Optimal decision trees for interpretable clustering with constraints". In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 2023, pp. 2022–2030.

[251] Pouya Shati, Eldan Cohen, and Sheila McIlraith. "SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features". In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021.

[252] Pouya Shati, Eldan Cohen, and Sheila McIlraith. "SAT-Based Learning of Compact Binary Decision Diagrams for Classification". In: *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2023.

[253] Pouya Shati, Eldan Cohen, and Sheila A McIlraith. "SAT-based optimal classification trees for non-binary data". In: *Constraints* 28.2 (2023), pp. 166–202.

[254] Jamie Shotton et al. "Decision jungles: Compact and rich models for classification". In: *Advances in neural information processing systems* 26 (2013).

[255] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[256] Carsten Sinz. "Towards an optimal CNF encoding of boolean cardinality constraints". In: *International conference on principles and practice of constraint programming*. Springer. 2005, pp. 827–831.

[257] Michael Sipser. "Introduction to the Theory of Computation". In: *ACM Sigact News* 27.1 (1996), pp. 27–29.

[258] Anna Slobodova and Christoph Meinel. "Sample method for minimization of OBDDs". In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer. 1998, pp. 419–428.

[259] Wen Song et al. "Learning variable ordering heuristics for solving constraint satisfaction problems". In: *Engineering Applications of Artificial Intelligence* 109 (2022), p. 104603.

[260] Gregor Stiglic et al. "Interpretability of machine learning-based prediction models in healthcare". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10.5 (2020), e1379.

[261] Alexander Strehl and Joydeep Ghosh. "Cluster ensembles—a knowledge reuse framework for combining multiple partitions". In: *Journal of Machine Learning Research* 3 (2002), pp. 583–617.

[262] Agus Sudjianto and Aijun Zhang. "Designing inherently interpretable machine learning models". In: *arXiv preprint arXiv:2111.01743* (2021).

[263] Tingni Sun and Cun-Hui Zhang. "Scaled sparse linear regression". In: *Biometrika* 99.4 (2012), pp. 879–898.

[264] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *NeurIPS*. 2014, pp. 3104–3112.

[265] Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. "The complexity of the optimal variable ordering problems of a shared binary decision diagram". In: *IEICE TRANSACTIONS on Information and Systems* 79.4 (1996), pp. 271–281.

[266] Anbupalam Thalamuthu et al. "Evaluation and comparison of gene clustering methods in microarray analysis". In: *Bioinformatics* 22.19 (2006), pp. 2405–2412.

[267] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria scheduling: theory, models and algorithms*. Springer Science & Business Media, 2006.

[268] Hazem Torfah et al. "Synthesizing pareto-optimal interpretations for black-box models". In: *2021 Formal Methods in Computer Aided Design (FMCAD)*. IEEE. 2021, pp. 153–162.

[269] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

[270] Geoffrey G Towell and Jude W Shavlik. "Knowledge-based artificial neural networks". In: *Artificial intelligence* 70.1-2 (1994), pp. 119–165.

[271] Eduardo Uchoa et al. "New benchmark instances for the capacitated vehicle routing problem". In: *EJOR* 257.3 (2017), pp. 845–858.

[272] Alfred Ultsch and Jörn Lötsch. "The fundamental clustering and projection suite (fcps): A dataset collection to test the performance of clustering and data projection algorithms". In: *Data* 5.1 (2020), p. 13.

[273] Antony Unwin and Kim Kleinman. "The iris data set: In search of the source of virginica". In: *Significance* 18.6 (2021), pp. 26–29.

[274] William TB Uther and Manuela M Veloso. "Tree based discretization for continuous state space reinforcement learning". In: *Aaai/iaai* 98 (1998), pp. 769–774.

[275] Michal Valko, Nuno C Marques, and Marco Castellani. "Evolutionary feature selection for spiking neural network pattern classifiers". In: *2005 portuguese conference on artificial intelligence*. IEEE. 2005, pp. 181–187.

[276] Moshe Y Vardi. "An automata-theoretic approach to linear temporal logic". In: *Logics for concurrency: structure versus automata* (2005), pp. 238–266.

[277] Kush R Varshney. "Engineering safety in machine learning". In: *2016 Information Theory and Applications Workshop (ITA)*. IEEE. 2016, pp. 1–5.

[278] Ashish Vaswani et al. "Attention is All you Need". In: *NeurIPS*. 2017, pp. 5998–6008.

[279] Vijay V Vazirani. *Approximation algorithms*. Vol. 1. Springer, 2001.

[280] Hélene Verhaeghe et al. "Learning optimal decision trees using constraint programming". In: *Constraints* 25.3 (2020), pp. 226–250.

[281] Sicco Verwer and Yingqian Zhang. "Learning optimal classification trees using a binary linear program formulation". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 1625–1632.

[282] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer Networks". In: *NeurIPS*. 2015, pp. 2692–2700.

[283] Kiri Wagstaff and Claire Cardie. "Clustering with Instance-level Constraints". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer. 2000, pp. 1103–1110.

[284] Kiri Wagstaff et al. "Constrained K-means Clustering with Background Knowledge". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. 2001, pp. 577–584.

[285] Qi Wang and Chunlei Tang. "Deep reinforcement learning for transportation network combinatorial optimization: A survey". In: *Knowledge-Based Systems* 233 (2021), p. 107526.

[286] Qinglong Wang et al. "An Empirical Evaluation of Rule Extraction from Recurrent Neural Networks". In: *Neural Computation* 30 (2017), pp. 2568–2591. URL: https://api.semanticscholar.org/CorpusID:27720983.

[287] Xiang Wang and Ian Davidson. "Flexible constrained spectral clustering". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2010, pp. 563–572.

[288] Dennis Wei et al. "On the safety of interpretable machine learning: A maximum deviation approach". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 9866–9880.

[289] Gary M Weiss. "Mining with rarity: a unifying framework". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pp. 7–19.

[290] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8 (1992), pp. 229–256.

[291] Gerhard J Woeginger. "Exact algorithms for NP-hard problems: A survey". In: *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*. Springer. 2003, pp. 185–207.

[292] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[293] SN Wood, Y Goude, and M Fasiolo. "Interpretability in generalized additive models". In: *Interpretability for Industry 4.0: Statistical and Machine Learning Approaches*. Springer, 2022, pp. 85–123.

[294] Min Wu, Atsushi Yamashita, and Hajime Asama. "Rule abstraction and transfer in reinforcement learning by decision tree". In: *2012 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2012, pp. 529–534.

[295] Roung-Shiunn Wu and Po-Hsuan Chou. "Customer segmentation of multiple category data in e-commerce using a soft-clustering approach". In: *Electronic Commerce Research and Applications* 10.3 (2011), pp. 331–341.

[296] Yaoxin Wu et al. "Learning improvement heuristics for solving routing problems". In: *IEEE transactions on neural networks and learning systems* 33.9 (2021), pp. 5057–5069.

[297] Feiyu Xu et al. "Explainable AI: A brief survey on history, research areas, approaches and challenges". In: *Natural language processing and Chinese computing: 8th cCF international conference, NLPCC 2019, dunhuang, China, October 9–14, 2019, proceedings, part II 8*. Springer. 2019, pp. 563–574.

[298] Haoyu Yang, Lianmeng Jiao, and Quan Pan. "A survey on interpretable clustering". In: *2021 40th Chinese Control Conference (CCC)*. IEEE. 2021, pp. 7384–7388.

[299] Chao Yin, Quentin Cappart, and Gilles Pesant. "An Improved Neuro-Symbolic Architecture to Fine-Tune Generative AI Systems". In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2024, pp. 279–288.

[300] Dongjie Yu et al. "Reachability constrained reinforcement learning". In: *International conference on machine learning*. PMLR. 2022, pp. 25636–25655.

[301] Jinqiang Yu et al. "Computing Optimal Decision Sets with SAT". In: *International Conference on Principles and Practice of Constraint Programming (CP)*. Springer. 2020, pp. 952–970.

[302] Jinqiang Yu et al. "Learning Optimal Decision Sets and Lists with SAT". In: *Journal of Artificial Intelligence Research* 72 (2021), pp. 1251–1279.

[303] Jinqiang Yu et al. “Optimal decision lists using SAT”. In: *arXiv preprint arXiv:2010.09919* (2020).

[304] Deniz Yuret and Michael De La Maza. “The greedy prepend algorithm for decision list induction”. In: *International Symposium on Computer and Information Sciences*. Springer. 2006, pp. 37–46.

[305] Xin Zhang et al. “K-nearest neighbors rule combining prototype selection and local feature weighting for classification”. In: *Knowledge-Based Systems* 243 (2022), p. 108451.

[306] Xu Zhang et al. “Interpretable machine learning models for crime prediction”. In: *Computers, Environment and Urban Systems* 94 (2022), p. 101789.

[307] Yichi Zhang et al. “Using decision lists to construct interpretable and parsimonious treatment regimes”. In: *Biometrics* 71.4 (2015), pp. 895–904.

[308] Yu Zhang et al. “A survey on neural network interpretability”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.5 (2021), pp. 726–742.

[309] Lina Zhou et al. “Machine learning on big data: Opportunities and challenges”. In: *Neurocomputing* 237 (2017), pp. 350–361.

[310] Rong Zhou and Eric A Hansen. “Beam-Stack Search: Integrating Backtracking with Beam Search.” In: *ICAPS*. 2005, pp. 90–98.

[311] Zhi-Hua Zhou and Xu-Ying Liu. “Training cost-sensitive neural networks with methods addressing the class imbalance problem”. In: *IEEE Transactions on knowledge and data engineering* 18.1 (2005), pp. 63–77.

[312] Honglei Zhuang et al. “Interpretable ranking with generalized additive models”. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 2021, pp. 499–507.

[313] José R Zubizarreta. “Using mixed integer programming for matching in an observational study of kidney failure after surgery”. In: *Journal of the American Statistical Association* 107.500 (2012), pp. 1360–1371.