A DECOMPOSABLE BRANCH-AND-PRICE FORMULATION FOR OPTIMAL CLASSIFICATION TREES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

By
Elif Rana Yöner
July 2024

A DECOMPOSABLE BRANCH-AND-PRICE FORMULATION FOR OPTIMAL CLASSIFICATION TREES

By Elif Rana Yöner

July 2024

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Özlem Karsu(Advisor)

Taghi Khaniyev(Co-Advisor)

Enis Kayış

Ezgi Karabulut Türkseven

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

A DECOMPOSABLE BRANCH-AND-PRICE FORMULATION FOR OPTIMAL CLASSIFICATION TREES

Elif Rana Yöner
M.S. in Industrial Engineering
Advisor: Özlem Karsu
Co-Advisor: Taghi Khaniyev
July 2024

Construction of Optimal Classification Trees (OCTs) using mixed-integer programs, is a promising approach as it returns a tree with minimum classification error. Yet solving integer programs to optimality is known to be computationally costly, especially as the size of the instance and the depth of the tree grow, calling for efficient solution methods. Our research presents a new, decomposable model which lends itself to efficient solution algorithms such as Branch-and-Price. We model the classification tree using a "pattern-based" formulation, deciding which feature should be used to split data at each branching node of each leaf. Our results are promising, illustrating the potential of decomposition in the domain of binary OCTs.

Keywords: Optimal Classification Trees, Branch-and-Price, Classification, Combinatorial Optimization, Machine Learning, Decomposition, Mixed- Integer Programming.

ÖZET

EN İYİ SINIFLANDIRMA AĞAÇLARINI BULMAK İÇİN GELİŞTİRİLMİŞ AYRIŞTIRILABİLİR DAL-FİYAT FORMÜLASYONU

Elif Rana Yöner
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Özlem Karsu
İkinci Tez Danışmanı: Taghi Khaniyev
Temmuz 2024

Karma-tamsayılı programlama kullanılarak en az sınıflandırma hatasına sahip sınıflandırma ağaçları elde edilebilir. Ancak, kullanılan veri kümelerinin boyutu ve ağacın derinliği arttıkça, ilgili tamsayılı programlama modellerinin çözülmesi hesaplama açısından maliyetlidir. Bu da, verimli çözüm yöntemlerinin kullanılmasını gerektirir. Bu çalışmada, optimal sınıflandırma ağacı problemlerinin çözülmesi için özgün bir dal-fiyat algoritması yaklaşımı sunulmaktadır. Problem, her seviyedeki karar düğümünde hangi özelliğin hangi yaprağa bölünmesi gerektiğine karar veren "desen tabanlı" bir formülasyon kullanılarak modellenmiş ve dal-fiyat yaklaşımı ile çözülmüştür. Elde edilen sonuçlar, önerilen yöntemin, enküçük hatayı veren sınıflandırma ağaçlarının oluşturulması için etkin bir yaklaşım olduğunu göstermektedir.

Anahtar sözcükler: En İyi Sınıflandırma Ağaçları, Dal-Fiyat, Sınıflandırma, Kombinatoryal Optimizasyon, Makine Öğrenmesi, Ayrıştırma, Karışık-Tamsayılı Programlama..

Acknowledgement

First and foremost, I would like to express my immense gratitude for my advisors Assoc. Prof. Özlem Karsu and Asst. Prof. Taghi Khaniyev, for their treasured support and remarkable guidance throughout my thesis journey. Their unwavering support, immeasurable encouragement, and endless wisdom kept me motivated as well as focused throughout my thesis journey. It was truly an honor working with them and making their acquaintance.

I would also like to thank the members of my thesis committee, Asst. Prof. Enis Kayış and Asst. Prof. Ezgi Karabulut Türkseven, for their constructive feedback.

I would like to acknowledge the financial support of The Scientific and Technological Research Council of Turkey (TUBITAK) for the 2210-National Graduate Study Scholarship Program they awarded.

I wish to thank my friends, Doğa Şahin, Beyza Bakır, Buse Özkan, Sena Özpınar, Elif Ece Arslan, Cansu Bacak, Beyza Doğan, Revna Demirci, Simge Çınar, and my cousin Mert Umurhan.

I would like to extend my appreciation to the faculty for their assistance and support during my studies, especially, Prof. Oya Karaşan, Prof. Bahar Yetiş, and Prof. M. Selim Aktürk. I also thank Vegün Ekmekçi and Yeşim Gülseren for all of their help.

I am thankful for all of my fellow graduate students, colleagues, and office mates; especially Göksu Ece Okur, Deniz Şahin, Deniz Akkaya, Zehranaz Dönmez Varol, Sena Aslı Bozkurt, and Dilruba Sultan Haliloğlu for their fun companionship and encouragement. To my cohort especially, I extend my deepest thanks for sharing many hours of endless studying. So, I thank many times over to Elif Sena Işık, Gülin Yurter, Gizem İkizler, İrem Bahtiyar, Doğuş Berk Koçak, Mustafa Çolak, and Ali Eren Demir.

I also thank Ali İlhan Haliloğlu, who has been with me throughout this journey, and tread alongside. He has made what would have been the challenge of a lifetime such a cheerful and bright experience. It is such a blessing to have him by my side as this chapter of our lives comes to an end.

Lastly, my heartfelt thanks go to my beloved family for their inspiration, understanding, and endless support. To my parents, I can easily say your love and encouragement have been my driving force. I thank my mother for always believing in me and fueling my passions, instilling in me a great sense of self to choose whatever I please with my life. I thank my father for his bright outlook on life, always encouraging me to push further and strive higher. It is an honor to complete this degree in his field and follow his footsteps, as well as share insightful discussions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Interpretability in machine learning has attracted significant attention, due to the need for understanding complex models and fostering trust in decision-making systems. Various sources emphasize the importance of interpretability, particularly highlighting the utility of classification trees in many contexts involving healthcare, policy-making, and business intelligence [1, 2]. Furthermore, in terms of the trade-off between model accuracy and interpretability, decision trees are recognized as models that maintain a favorable balance [3].

For binary split classification trees, which we will focus on, there are two main components of the tree structure: decision nodes and terminal nodes. The aim is to construct trees that accurately predict outcomes by partitioning the data into meaningful segments. In each decision node of the tree, a specific feature is subjected to binary tests, or *feature-splits*, leading to two branches per node. Based on this test outcome, data points passing (resp. failing) the test are directed to the left (resp. right) branch. Every leaf node is assigned a predicted label, making each root-to-leaf path a unique decision rule. Building a classification tree involves choosing a series of feature-splits and a label for each leaf to maximize prediction accuracy, while maintaining the tree structure. The traditional construction algorithms often involve a greedy approach, which may lead to suboptimal results. In this study, we propose several new approaches to construct *Optimal Classification Trees* (OCTs) by exploiting the decomposable structure of the problem.

We provide an overview of heuristic and optimization methods to find OCTs in the literature in Chapter 2. Then, we go on to give a detailed explanation of our problem

through the lens of our proposed integer programming approaches, and define our terminology extensively before presenting our models named *Compact Formulation*, *Pattern Based Formulation*, and *Branch-and-Price Formulation* in Chapter 3. In the following Chapter 4, we display the adaptable nature of our model to fairness metrics, by providing Fairness-Aware versions of *Compact Formulation*, *Pattern Based Formulation*, and *Branch-and-Price Formulation* addressing a fairness metric ODM. We proceed to explain the caveats of our formulation, and explain some of our modeling decisions in Chapter 5. Then, we describe a Branch-and-Price approach, well suited for OCTs in Chapter 6. In Chapter 7, we provide pairwise comparison of the proposed and existing methods in a number of numerical analyses and the contributions of our methods are hihlighted. We conclude our work in Chapter 8 via an overall discussion of our methods' performance and intended future studies.

Chapter 2

Literature Review

In the literature, one way to approach finding high-accuracy classification trees is to use heuristic approaches that choose which feature to split on greedyly level by level, to minimize one-step cost (Misclassification or Gini Impurity Index), until maximum allowed depth level is reached. *Classification and Regression Trees (CART)* Algorithm, introduced by Breiment et al. [4]. Similarly, the *Iterative Dichotomise* (ID3) Algorithm introduced by Quanlan which determines splits based on increasing Information Gain, which measures the reduction in entropy [5]. Another extension is *C4.5*, an algorithm using Information Gain Ratio, instead of Information Gain, for choosing splits, again presented by Quanlan [6]. There are many other heuristic algorithms that are based on similar foundations of a greedy logic, which yield good- yet suboptimal solutions.

Besides heuristics, there are also methods designed to find the classification tree with the least misclassification value, defined over a search space of all trees. As per their combinatorial nature, building OCTs are known to be $\mathcal{NP}$-complete [7]. However, over the past several years, a few attempts have been made at solving classification trees to optimality. In 2017, Bertsimas and Dunn introduced the notion of OCTs as a mathematical optimization framework to construct decision trees that minimize specific cost functions, thereby enhancing the efficiency of decision-making processes [8]. Their work, functioning as a showcase encouraging the use of Mixed-Integer Programming (MIP) to construct OCTs, gained ground quickly. This is followed by the works of Günlük et al. (2021), Aghaei et al. (2019), and Verwer and Zhang (2019). Günlük et al. proposed an Integer Programming (IP) formulation for constructing optimal binary classification trees for data consisting of categorical features. They consider the problem structure to reflect feature-splits as combinatorial decisions apparent at each decision node, and display the

solvability of this structure up to a certain depth as well as the empirical quality of the solution found within a time limit [9]. Aghai et al. augment the former MIP formulations to incorporate notions of fairness [10]. In all of these former formulations, a constraint is added for each data point to dictate the feature splits at each level. Verwer and Zhang proposed a different approach where the number of variables is largely independent of data size via the use of binary variables, titled *BinOCT* [11]. Moving forward, several papers have addressed scalability issues in MIP formulations involving constraining of data point flows. Aghaei et. al utilize their flow-based formulation, *FlowOCT*, and harness max-flow/min-cut duality to develop a Benders' decomposition method, *BendersOCT*, enhancing computational efficiency [12, 13].

Another decompositional approach was made by Firat et al. [14]. They use an IP formulation based on root-to-leaf paths in decision trees to construct binary decision trees for classification tasks. They solve the model via a Column Generation-based heuristic, reporting solutions that are competitive with CART [14]. Lumadjeng et al. also use Column Generation-based heuristics to generate rules for classification trees [15]. They also introduce two-notions of fairness, *Disparate Mistreatment per Class* (DMC) and *Overall Disparate Mistreatment* (ODM), which are extensions of fairly well covered notions of *Equalized Odds* (EOD) and *Disparate Mistreatment* (DM).

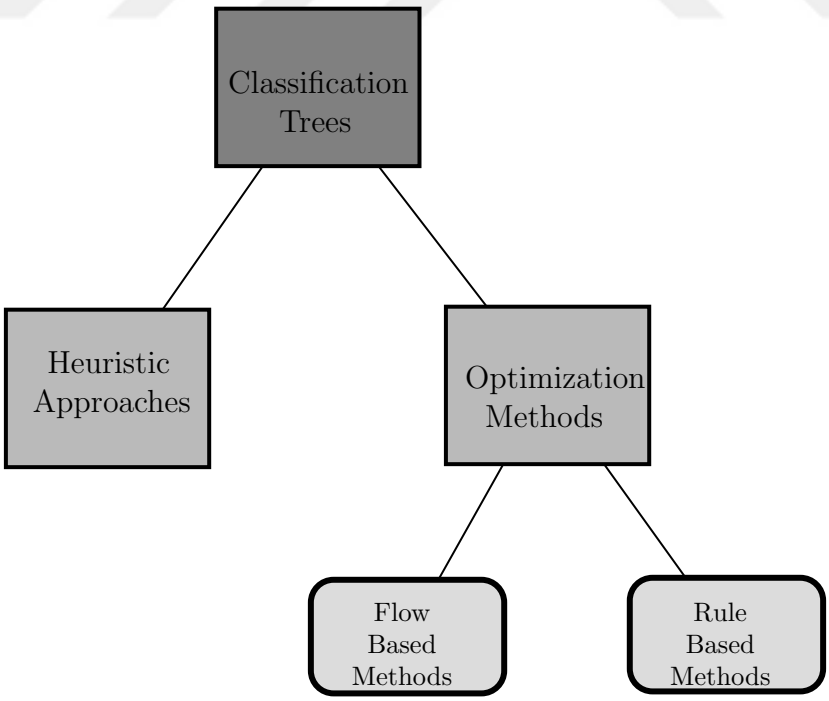

Figure 2.1: Visualization of Methods in Literature for Building Optimal Classification Trees

For a general overview of the methods in literature used to build classification trees, please see Figure 2.1. This categorization is inspired by a survey on mathematical optimization classification trees by Carrizosa et al. [16]. Greedy Heuristics follow a one-step greedy logic to choose splits, thus, algorithms such as CART, ID3, and C4.5 fall under this category. When we look at the Optimization Model Approaches, we see Flow-Based Models and Rule-Based Models. Their main distinction is that Flow-Based Models frame the problem as an Mixed Integer Program (MIP) where each data point must *flow* to their respective leaf node, based on the split-node feature decisions. This requires the addition of binary decision variables for potential split-nodes features and data points, as well as binary indicators of misclassified data points. On the other hand, Rule-Based Models view the problem as a choice amongst a given set of *rules* defined for the tree, and do not worry about the integrality of the decision variables defined for these rules. Instead, regularization parameters (e.g. a penalty term for each rule used in the tree) are exploited to "force" the model to choose as little rules as needed to approximate integrality. For the methods dicussed above, Bertsimas and Dunn's Formulation , Günlük et al.'s Formulation, BinOCT, FlowOCT, and BendersOCT fall under Flow-Based Models; whereas Firat et al.'s and Röber et al.'s Formulations follow as Rule-Based Models.

With a broad overview of Classification Trees methods found in literature introduced, let us focus on a computational optimization tool which will be used amply in the rest of our discussion. Branch-and-Price (also referred to as Branch & Price throughout this text), is a versatile computational optimization method that is used to solve computationally exhaustive optimization problems. It extends the idea of column generation to problems with integrality constraints [17]. Firstly, the Integer Programing (IP) or Mixed Integer Programming (MIP) problem at hand is modeled using the Dantzig-Wolfe Reformulation. Then, a proper branching rule is chosen, which is compatible with column generation [18]. Column generation, or the process of repeatedly solving pricing problems until convergence (proven master problem optimality of the current optimal solution to the restricted master problem) is repeated at every node of what would normally be the Branch-and-Bound tree, hence the name Branch-and-Price. This methodology has proven to be useful on a myriad of problem types, ranging from multi-commodity origin-destination network problems to bin-packing problems [19, 20]. Consequently, there are many established implementations of Branch-and-Price, in notoriously difficult application fields such as Vehicle Routing Problem with time windows, Airline Fleet Management, and Facility Location Planning [21, 22, 23].

Our research presents an approach to solving OCT programs to optimality using a novel formulation, which combines elements of both Flow-Based and Rule-Based formulations, then presents a Branch-and-Price algorithm to accompany our proposed model . Firstly, we present a decomposable integer programming formulation, that decides which feature each leaf must be split upon at each level's decision node, depending on the . Then, we model the classification tree using a pattern-based formulation which picks from a set of patterns for each leaf. We proceed to apply column generation to the LP-Relaxation of the pattern-based formulation and go on to use Branch-and-Price. Finally, we compare these methods computationally by reporting their performance on various data sets.

Chapter 3

Models

In this Chapter, we will present our proposed solution approach to building OCTs. Before, we proceed, let us cover some additional terminology which shall be used extensively for the remaining part of this thesis. *Split-features* denote the feature we partition the data on for each decision node in the tree (A, B, and, C in Figure 3.1). *Split-values* are the properties that a data point must satisfy (either having the data entry 1 or 0 for the split-feature) to pass to a specific leaf in the tree. *Leaves* are the terminal nodes at the end of the tree (The rectangular nodes depicted in Figure 3.1). For each leaf, a level's split-feature and split-value are concatenated to compose a *split-condition*. In Figure 3.1, $x_{ij_A} = 0$ and $x_{ij_A} = 1$ are split-conditions for leaf 1 and leaf 2, in the first level. The split conditions will be conjoined level-wise to create a *decision rule* for each leaf. Fpr example, in Figure 3.1, $x_{ij_A} = 0$ and $x_{ij_B} = 0$ represent leaf 1. Hence, each decision rule will contain a combination of split-features and split-values defining a path to a leaf. A *fork* refers to a triple of parent decision node and two resulting child nodes (For example B and $1, 2$ or A and B, C in Figure 3.1).

We will discuss three types of formulations for the OCT, which we classify as compact and pattern based respectively. For the pattern based formulations, we provide an IP formulation, and an accompanying Branch-and-Price scheme to the IP. Our first (compact) model treats the split-features at each decision node for each leaf as a decision variable to minimize total misclassification cost incurred while respecting the tree structure. The second (pattern-based) model acts as though we have access to the set of all possible decision rules (paths) for each leaf, and chooses amongst the combinations to create a binary classification tree minimizing the misclassification. The last (pattern-based) model takes the LP-relaxation of the pattern-based IP, and applies Branch-and-Price to solve to

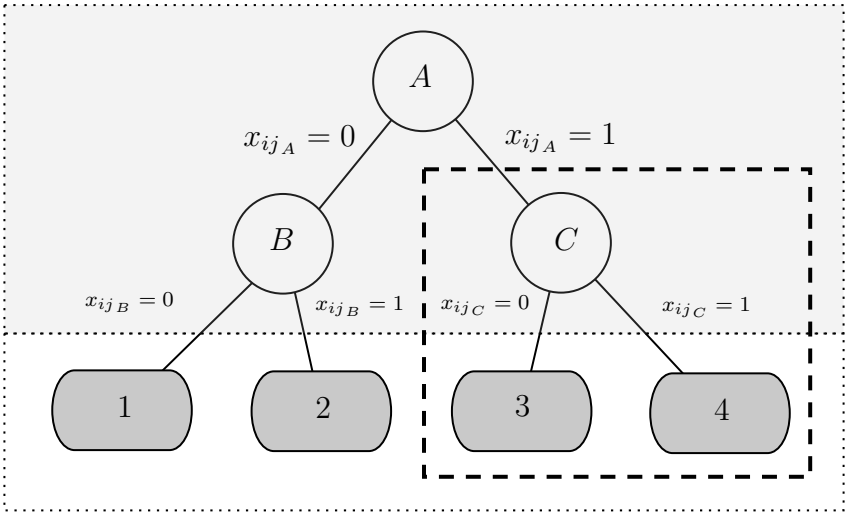

Figure 3.1: Structure of the Classification Tree for Depth 2.

optimality. Common sets and parameters associated with all of our models are provided in Table 3.1.

Table 3.1: Common Sets and Parameters Used Throughout the OCT Models.

Index sets	
$\mathcal{N}$	The set of data points
$\mathcal{K}$	The set of all classes for data points $\mathcal{N}$
$\mathcal{J}$	The set of all features for data points $\mathcal{N}$
$\mathcal{L}$	The set of leaves in the classification tree
$\mathcal{S}$	The set of all internal split nodes for tree of depth D, $\{1, ..., 2^D - 1\}$
$\mathcal{S}_l^0$	The set of left split nodes associated with leaf $l \in \mathcal{L}$
$\mathcal{S}_l^1$	The set of right split nodes associated with leaf $l \in \mathcal{L}$

Parameters	
D	Maximum allowed depth for the classification tree
N	The number of data points, $\lvert\mathcal{N}\rvert$
K	The number of all classes for data points, $\lvert\mathcal{K}\rvert$
J	The number of all features for data points, $\lvert\mathcal{J}\rvert$
L	The number of leaves in the classification tree, $\lvert\mathcal{L}\rvert = 2^D$
S	The number of all internal split nodes in the classification tree, $\lvert\mathcal{S}\rvert = 2^D - 1$
S_l^0	The number of negative internal split nodes associated with leaf $l \in \mathcal{L}$
S_l^1	The number of positive internal split nodes associated with leaf $l \in \mathcal{L}$
x_{ij}	The binary entry in feature $j \in \mathcal{J}$ in data point $i \in \mathcal{N}$
y_{ik}	The binary indicator label of data point $i \in \mathcal{N}$ for class $k \in \mathcal{K}$

3.1 Compact Formulation

In our first formulation, by deciding on the split-features at each level for each leaf through w_{js} variables, we also decide on which leaf each data point will flow to. a_{jl} and b_{jl} trace what features are selected as zero (left) or one (right) splits for each leaf. Moreover, we also assign a class to each leaf based on the majority label with c_{kl} variables. For each data point i, we keep track of whether it is misclassified using variables t_{il}. Table 3.2 summarizes the decision variables introduced for this formulation.

Table 3.2: Decision Variables for the Compact Formulation Model.

Decision Variables

$t_{il} = \begin{cases} 1, & \text{if data point } i \in \mathcal{N} \text{ falls under the leaf } l \in \mathcal{L} \text{ and is misclassified} \\ 0, & \text{otherwise} \end{cases}$

$c_{kl} = \begin{cases} 1, & \text{if leaf } l \in \mathcal{L} \text{ is assigned class } k \in \mathcal{K} \\ 0, & \text{otherwise} \end{cases}$

$w_{js} = \begin{cases} 1, & \text{if feature } j \in \mathcal{J} \text{ is chosen as split feature for split node } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases}$

a_{jl} The number of times feature $j \in \mathcal{J}$ is selected as a negative split-feature for leaf l

b_{jl} The number of times feature $j \in \mathcal{J}$ is selected as a positive split-feature for leaf l

CompactOCT Model:

$$\text{minimize} \quad \frac{1}{N} \sum_{i \in \mathcal{N}} \sum_{l \in \mathcal{L}} t_{il} \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}} w_{js} = 1 \qquad \forall s \in \mathcal{S} \tag{3.1b}$$

$$t_{il} \geq \sum_{j \in \mathcal{J}} x_{ij} b_{jl} + \sum_{j \in \mathcal{J}} (1 - x_{ij}) a_{jl} - \sum_{k \in \mathcal{K}} y_{ik} c_{kl} - (D-1) \quad \forall i \in \mathcal{N}, \forall l \in \mathcal{L} \tag{3.1c}$$

$$\sum_{k \in \mathcal{K}} c_{kl} = 1 \qquad \forall l \in \mathcal{L} \tag{3.1d}$$

$$\sum_{s \in \mathcal{S}_l^0} w_{js} = a_{jl} \qquad \forall j \in \mathcal{J}, \quad \forall l \in \mathcal{L} \tag{3.1e}$$

$$\sum_{s \in \mathcal{S}_l^1} w_{js} = b_{jl} \qquad \forall j \in \mathcal{J}, \quad \forall l \in \mathcal{L} \tag{3.1f}$$

$$\sum_{j \in \mathcal{J}} a_{jl} = S_l^0 \qquad \forall l \in \mathcal{L} \tag{3.1g}$$

$$\sum_{j \in \mathcal{J}} b_{jl} = S_l^1 \qquad \forall l \in \mathcal{L} \tag{3.1h}$$

$$a_{jl}, b_{jl} \in Z_+ \cup \{0\} \qquad \forall j \in \mathcal{J}, \forall l \in \mathcal{L} \tag{3.1i}$$

$$w_{js} \in \{0, 1\} \qquad \forall j \in \mathcal{J}, \forall s \in \mathcal{S} \tag{3.1j}$$

$$t_{il} \in \{0, 1\} \qquad \forall i \in \mathcal{N}, \forall l \in \mathcal{L} \tag{3.1k}$$

The objective is to minimize misclassification as given by (3.1a). Constraints (3.1b) ensure that for all split nodes, exactly one split-feature is chosen among all features. Constraints (3.1c) control the flow of data points to leaves based on the split conditions.

Constraint (3.1d) makes sure each leaf is assigned a single class. The next sets of constraints (3.1e) and (3.1f) control that chosen zero (left)/ one (right) split-features for split-nodes match with S_l^0 and S_l^1 values for each leaf. Constraints (3.1g) and (3.1h) set the number of negative and positive split features that should be selected for each leaf. Finally,(3.1i-3.1k) define the domains of decision variables.

Notice that, if not for the constraints linking the leaves through shared nodes (3.1e and 3.1f), this model would be decomposable and into L independent subproblems. Hence, if we were to relax Constraints (3.1e) and (3.1f) in a Lagrangian manner, or convexify the remaining constraints, while keeping (3.1e) and (3.1f)in a master problem; we would obtain a Dantzig-Wolfe Reformulation of the same problem. We now present this reformulation in the next subsection.

3.2 Pattern Based Model

In Table 3.3, we give the additional sets, parameters, and decision variables used in the Dantzig-Wolfe Reformulation of Compact OCT

Our patterns, defined for each leaf separately, contain 3 key information: split-features which will be used in the zero-splits, split-features which will be used in the one-splits, and the overall misclassification rate ($\frac{E_l^{(t)}}{N}$) that is associated with the pattern, referred to as the misclassification cost $m_l^{(t)}$, where $E_l^{(t)}$ represents the number of incorrectly classified points in the data set, out of the data that falls under leaf l with the 0-splits and 1-splits.

To clarify our definition of a pattern, let us state the following: When the tree's maximum allowed depth is quantified by D, each pattern for a tree leaf l consists of an ordered 2-dimensional tuple of tuples of sizes S_l^0 and S_l^1 , corresponding to the number of zero and one splits associated with leaf l respectively. Features that are used for zero-split nodes belong to the first tuple and one-split nodes belong to the second. Based on the leaf's position, number of zero and one split-values S_l^0 and S_l^1 are predetermined. The selected patterns form a tree adhering to the split values of all leaves and their relations to each other. An example tree in a two-depth setting would be {[1, 2],[-]} for leaf 1, {[1],[2]} for leaf 2, {[3],[1]} for leaf 3, and {[3, 1],[-]} for leaf 4.

Table 3.3: Sets, Parameters, and Decision Variables Introduced for the Pattern Based Model.

Index sets

$\mathcal{T}_l$	The set of defined patterns for leaf $l \in \mathcal{L}$
$\mathcal{T}$	The union of all defined patterns $\mathcal{T}_l$ for leaf $l \in \mathcal{L}$

Parameters

$\alpha_j^{(t)}$	The number of times in which feature $j \in \mathcal{J}$, $x_j = 0$ is the split condition for pattern $t \in \mathcal{T}_l$ for leaf $l \in \mathcal{L}$
$\beta_j^{(t)}$	The number of times in which feature $j \in \mathcal{J}$, $x_j = 1$ is the split condition for pattern $t \in \mathcal{T}_l$ for leaf $l \in \mathcal{L}$
$m_l^{(t)}$	Misclassification (cost) of pattern $t \in \mathcal{T}_l$, for leaf $l \in \mathcal{L}$

Decision Variables

$$z_l^{(t)} = \begin{cases} 1, & \text{If pattern } t \in \mathcal{T}_l \text{ is used in the tree for leaf } l \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$$

$$w_{js} = \begin{cases} 1, & \text{If feature } j \in \mathcal{J} \text{ is the split-feature on the intermediate node } s \in \mathcal{S} \\ 0, & \text{otherwise} \end{cases}$$

P_OCT Model:

$$\text{minimize} \quad \sum_{l \in \mathcal{L}} \sum_{t \in \mathcal{T}_l} m_l^{(t)} z_l^{(t)} \tag{3.2a}$$

$$\text{s.t.} \quad \sum_{t \in \mathcal{T}_l} z_l^{(t)} = 1 \qquad \forall l \in \mathcal{L} \tag{3.2b}$$

$$\sum_{s \in \mathcal{S}_l^0} w_{js} - \sum_{t \in \mathcal{T}_l} \alpha_j^{(t)} z_l^{(t)} = 0 \qquad \forall j \in \mathcal{J}, \quad \forall l \in \mathcal{L} \tag{3.2c}$$

$$\sum_{s \in \mathcal{S}_l^1} w_{js} - \sum_{t \in \mathcal{T}_l} \beta_j^{(t)} z_l^{(t)} = 0 \qquad \forall j \in \mathcal{J}, \quad \forall l \in \mathcal{L} \tag{3.2d}$$

$$z_l^{(t)} \in \{0, 1\} \qquad \forall t \in \mathcal{T}_l, \forall l \in \mathcal{L} \tag{3.2e}$$

$$w_{js} \in \{0, 1\} \qquad \forall j \in \mathcal{J}, \forall s \in \mathcal{S} \tag{3.2f}$$

As in the previously introduced model, our objective is to minimize total misclassification over all selected patterns belonging to each leaf (3.2a). In a similar fashion to what we have done so far, for all leaves, we enforce that only one pattern is selected (3.2b). In this model however, we introduce parameters denoting potential features under consideration for the internal split-nodes of the tree, through the notion of patterns. Using these parameters, we are able to relate split-features of different leaves' patterns implicitly. Hence, relations of nodes to each other are established in constraints (3.2c) and (3.2d). Constraints (3.2e) and (3.2f) are domain constraints indicating all decision variables are

binary. For the LP relaxation of the problem, we modify Constraint (3.2e) as $0 \leq z_l^{(t)} \leq 1 \quad \forall t \in \mathcal{T}_l, \forall l \in \mathcal{L}$ and Constraint (3.2f) as $0 \leq w_{js} \leq 1 \quad \forall j \in \mathcal{J}, \forall s \in \mathcal{S}$.

This model can be solved directly when the full set of patterns is easy to generate and store. We refer to this method as *Full Patterns IP* or P_OCT throughout this text. However, this would not be practical as the number of features J and the tree depth D grows. We next address this challenge using a Branch-and-Price scheme.

3.3 Branch-and-Price

We propose a new B&P scheme to solve the OCT problem. We start with an initial feasible solution set of patterns, we solve the LP-relaxation of *Pattern Based* Model, P_OCT. Using the optimal dual variable values obtained from our model, we solve the pricing problems. We seek to find patterns which violate dual feasibility the most. Once we add those patterns to our set of initial patterns, we solve the LP-Relaxation of P_OCT once again. We repeat the same procedure until convergence. If the problem is not integral, we branch until integrality. Our branching rule is to split on the first internal node for which a split variable is not yet chosen.

3.3.1 Dual of the LP Relaxation

For the dual, we define new decision variables, which can be seen in Table 3.4. We realize that constraints (3.3b) are independent of $\mathcal{T}$. Hence, the initial set of feasible patterns used in the primal ($\mathcal{T}^{(0)} \in \mathcal{T}$) will not affect the feasibility of the dual. Furthermore, we see that the dual feasibility is defined separately for each $l \in \mathcal{L}$. Hence, dual feasibility can be checked by considering whether or not any of these L sets of constraints are violated (3.3c) separately using independent subproblems.

Table 3.4: Decision Variables Introduced for the Dual Problem of the Pattern Based Formulation.

Decision Variables	
θ_l	= The dual decision variable associated with leaf $l \in \mathcal{L}$ for constraints (**??**)
π_{lj0}	= The dual decision variable associated with left (zero) split nodes of leaf $l \in \mathcal{L}$ and feature $j \in \mathcal{J}$ for constraints (3.2c)
π_{lj1}	= The dual decision variable associated with right (one) split nodes of leaf $l \in \mathcal{L}$ and feature $j \in \mathcal{J}$ for constraints (3.2d)

Dual of LP-Relaxation of P_OCT

$$\text{maximize} \quad \sum_{l \in \mathcal{L}} \theta_l \tag{3.3a}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}} \sum_{l: s \in \mathcal{S}_l^0} \pi_{lj0} + \sum_{j \in \mathcal{J}} \sum_{l: s \in \mathcal{S}_l^1} \pi_{lj1} \leq 0 \qquad \forall s \in \mathcal{S} \tag{3.3b}$$

$$\theta_l - \sum_{j \in \mathcal{J}} \alpha_j^{(t)} \pi_{lj0} - \sum_{j \in \mathcal{J}} \beta_j^{(t)} \pi_{lj1} \leq m_l^{(t)} \qquad \forall l \in \mathcal{L}, \forall t \in \mathcal{T}_l \tag{3.3c}$$

$$\theta_l \quad urs \qquad \forall l \in \mathcal{L} \tag{3.3d}$$

$$\pi_{lj0}, \pi_{lj1} \quad urs \qquad \forall l \in \mathcal{L}, \forall j \in \mathcal{J} \tag{3.3e}$$

If none of the L sets of constraints defined above are violated with the optimal solution obtained from the current set of patterns, $\mathcal{T}^{(0)}$, then the primal optimal solution t^* would be dual feasible, proving $t^* \in \mathcal{T}^{(0)}$ to be the optimal solution amongst set of all possible feasible patterns $\mathcal{T}$. Thus, we define our *pricing problems* for each $l \in \mathcal{L}$, which aim to find patterns with parameters $\alpha_j^{(t)}$ and $\beta_j^{(t)}$ defined over $\forall j \in \mathcal{J}$, which minimize:

$$\{m_l^{(t)} + \sum_{j \in \mathcal{J}} \alpha_j^{(t)} \pi_{lj0} + \sum_{j \in \mathcal{J}} \beta_j^{(t)} \pi_{lj1}\}$$

If the expression above is greater than $\theta_l, \forall l \in \mathcal{L}$, then dual feasibility is not violated and the optimal solution to the restricted master problem t^* is indeed optimal for the LP-relaxation of the master problem. However, if not, then, we add the patterns that minimize the expression above to their related leaf pattern sets $\mathcal{T}_l^{(0)} \subseteq \mathcal{T}^{(0)}$. Now we have updated our initial set of patterns $\mathcal{T}^{(0)}$, and this modified set will now be referred to as $\mathcal{T}^{(1)}$ .With $\mathcal{T}^{(1)}$, we solve the restricted master problem again, obtain the dual decision variables, and check once more if dual feasibility is violated. We repeat this *pricing procedure* (also referred to as the column generation) until dual feasibility holds.

To find the patterns that minimize our expression without exhaustive enumeration, we need to define an optimization problem which will yield our target patterns violating dual feasibility, if they exist. Again, as per their separable nature, each pricing optimization problem will deal with a single leaf $l \in \mathcal{L}$.

3.3.2 Pricing Problem for Leaf l:

In order to generate patterns that possibly violate dual feasibility, we introduce the following notation found in Table 3.5 and formulation:

Table 3.5: Parameters and Decision Variables Introduced for the Pricing Problems of the Pattern Based Formulation.

Parameters	
θ_l^*	= the dual decision variable values associated with leaf $l \in \mathcal{L}$ for constraints (3.2b)
π_{lj0}^*	= The dual decision variable values associated with left (zero) split nodes of leaf $l \in \mathcal{L}$ and feature $j \in \mathcal{J}$ for constraints (3.2c)
π_{lj1}^*	= The dual decision variable values associated with right (one) split nodes of leaf $l \in \mathcal{L}$ and feature $j \in \mathcal{J}$ for constraints (3.2d)
Decision Variables	
t_{il}	$= \begin{cases} 1, & \text{if data point } i \in \mathcal{N} \text{ falls under leaf } l \text{ and is misclassified} \\ 0, & \text{otherwise} \end{cases}$
a_{jl}	The number of times feature $j \in \mathcal{J}$ is selected as a left split-feature for leaf l
b_{jl}	The number of times feature $j \in \mathcal{J}$ is selected as a right split-feature for leaf l
c_{kl}	$= \begin{cases} 1, & \text{if leaf } l \text{ is assigned class } k \in \mathcal{K} \\ 0, & \text{otherwise} \end{cases}$

Pricing Problem for Leaf l:

$$\text{minimize} \quad \frac{1}{N}\sum_{i\in\mathcal{N}} t_{il} + \sum_{j\in\mathcal{J}} \pi^*_{lj0} a_{jl} + \sum_{j\in\mathcal{J}} \pi^*_{lj1} b_{jl} \tag{3.4a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{J}} a_{jl} = S^0_l \tag{3.4b}$$

$$\sum_{j\in\mathcal{J}} b_{jl} = S^1_l \tag{3.4c}$$

$$\sum_{k\in\mathcal{K}} c_{kl} = 1 \tag{3.4d}$$

$$t_{il} \geq \sum_{j\in\mathcal{J}} (1 - x_{ij}) a_{jl} + \sum_{j\in\mathcal{J}} x_{ij} b_{jl} - \sum_{k\in\mathcal{K}} y_{ik} c_{kl} - (D-1) \qquad \forall i \in \mathcal{N} \tag{3.4e}$$

$$t_{il} \in \{0,1\} \qquad \forall i \in \mathcal{N} \tag{3.4f}$$

$$c_{kl} \in \{0,1\} \qquad \forall k \in \mathcal{K} \tag{3.4g}$$

$$a_{jl}, b_{jl} \in Z_+ \cup \{0\} \qquad \forall j \in \mathcal{J} \tag{3.4h}$$

Please note that the relaxation of Constraints (3.4f) to $0 \leq t_{il} \leq 1 \quad \forall i \in \mathcal{N}$ also yields integral solutions in our pricing model.

Chapter 4

Incorporating Fairness

Only considering accuracy when building OCTs may cause unfair treatment to data samples sharing a sensitive feature, especially if they are in the minority in an imbalanced data set. Hence, evaluating the fairness metrics of our models convey great importance. Our formulations allow straightforward incorporation of fairness metrics into the models. We demonstrate this on a commonly used metric Overall Disparate Mistreatment (ODM). ODM, which is defined as "the difference of mistreatment between the sensitive and non-sensitive groups" [15] and mathematically represented in Expression (4.1a):

$$|P(\hat{y}_i \neq y_i | i \in (+)_p) - P(\hat{y} \neq y_i | i \in (-)_p)| \tag{4.1a}$$

With all of our base models proposed, we suggest the following formulations for constraining Overall Disparate Mistreatment (ODM) in our solution: We introduce the Compact Model's formulation first, then proceed to the Pattern Based Model and its pricing components. For common notations introduced for all proposed *Fairness-Aware Models* , please refer to Table 4.1.

Table 4.1: Sets and Parameters Introduced for Fairness-Aware Models.

Index sets	
$(+)_p$	The set of protected data samples (sensitive group), $i \in \mathcal{N} : x_{ip} = 1$, $(+)_p \subset \mathcal{N}$
$(-)_p$	The set of data points that do not belong in the sensitive group, $i \in \mathcal{N} :$ $x_{ip} = 0$, $(-)_p \subset \mathcal{N}$
$\mathcal{P}$	The set of protected features, $\mathcal{P} \subset \mathcal{J}$

Parameters	
ϵ	A threshold used to control ODM constraints
$N_{(+)p}$	The number of data points, belonging to the sensitive group created by feature $p \in \mathcal{P}$, $\lvert(+)_p\rvert$
$N_{(-)p}$	The number of data points, belonging to the non-sensitive group, created by feature $p \in \mathcal{P}$ $\lvert(-)_p\rvert$

4.1 Fair Compact Model:

When we add constraints (4.2a) and (4.2b) to the CompactOCT Model Defined in Section 3.1 (minimize (3.1a), s.t. constraints (3.1b)-(3.1k)), work together to linearize the ODM value defined in Expression (4.1a) and force it to be less then ϵ in the optimal solution.

Fair CompactOCT

$$\frac{1}{N_{(+)p}} \sum_{i \in (+)_p} \sum_{l \in \mathcal{L}} t_{il} - \frac{1}{N_{(-)p}} \sum_{i \in (-)_p} \sum_{l \in \mathcal{L}} t_{il} \leq \epsilon \quad \forall p \in \mathcal{P} \tag{4.2a}$$

$$\frac{1}{N_{(-)p}} \sum_{i \in (-)_p} \sum_{l \in \mathcal{L}} t_{il} - \frac{1}{N_{(+)p}} \sum_{i \in (+)_p} \sum_{l \in \mathcal{L}} t_{il} \leq \epsilon \quad \forall p \in \mathcal{P} \tag{4.2b}$$

4.2 Fair Pattern-Based Model:

As it can be observed, Constraints (4.3a) and (4.3b), work together to linearize the ODM value defined in Expression (4.1a) and force it to be less then ϵ. With the addition of Constraints (4.3a) and (4.3b) to the P_OCT Model Defined in Section 3.2 (minimize (3.2a), s.t. constraints (3.2b)-(3.2f)), we are able to produce solutions adhering to the

Table 4.2: Parameters Introduced for Fairness- Aware Pattern-Based Models.

Parameters	
$m_l^{(t)}{}_{(+)_p}$	Misclassification (cost) of pattern $t \in \mathcal{T}_l$, for leaf $l \in \mathcal{L}$, calculated over data points $i \in (+)_p$, $\forall p \in \mathcal{P}$
$m_l^{(t)}{}_{(-)_p}$	Misclassification (cost) of pattern $t \in \mathcal{T}_l$, for leaf $l \in \mathcal{L}$, calculated over data points $i \in (-)_p$, $\forall p \in \mathcal{P}$
δ_p^t	The difference between misclassification (cost) of pattern $t \in \mathcal{T}_l$, for leaf $l \in \mathcal{L}$, between data points belonging to the the sensitive group $(+)_p$ and data points belonging to the non-sensitive group, $(-)_p$, $\lvert m_l^{(t)}{}_{(+)_p} - m_l^{(t)}{}_{(-)_p} \rvert$, $\forall p \in \mathcal{P}$.

maximum allowed ϵ value for ODM. The related notation introduced for the Fairness-Aware Pattern-Based Model can be found in Table 4.2. ***Fair P_OCT***

$$\sum_{l \in \mathcal{L}} \sum_{t \in \mathcal{T}_l} \delta_p^{(t)} z_l^{(t)} \leq \epsilon \qquad \forall p \in \mathcal{P} \tag{4.3a}$$

$$\sum_{l \in \mathcal{L}} \sum_{t \in \mathcal{T}_l} -\delta_p^{(t)} z_l^{(t)} \leq \epsilon \qquad \forall p \in \mathcal{P} \tag{4.3b}$$

Our previous argument holds in the sense that this model can be solved directly again when the full set of patterns is easy to generate and store. However, this would not be practical as the number of features J and the tree depth D grows. We address this challenge using a Fairness-Aware Branch-and-Price scheme.

4.3 Fair Branch-and-Price

4.3.1 Dual of the LP Relaxation

Now, we define the dual problem of the LP-Relaxation of Fair P_OCT using the newly introduced decision variables defined in Table 4.3.

As in the previous case, we see that constraints (3.3b) are independent of $\mathcal{T}$. Hence, the initial set of feasible patterns used in the primal ($\mathcal{T}^{(0)} \in \mathcal{T}$) will not affect the feasibility of the dual, in the fairness-aware model neither. Furthermore, we see that we can still check the dual feasibility with L different subproblems that are independent of each other. Hence, dual feasibility can be checked by considering whether or not any of

Table 4.3: Decision Variables Introduced for the Dual Problem of the Fairness Aware Pattern Based Formulation.

Decision Variables	
γ_{p1}	the dual decision variable values associated with protected feature $p \in \mathcal{P}$ for constraints (4.3a)
γ_{p2}	the dual decision variable values associated with protected feature $p \in \mathcal{P}$ for constraints (4.3b)

these L sets of constraints are violated (3.3c) separately. However, we realize that the constraints have an additional component involving a linear combination of γ_{p1} and γ_{p2}, which will correspond to a slightly modified objective function for the pricing problem.

Dual of LP Relaxation of P_OCT

$$\text{maximize} \quad \sum_{l \in \mathcal{L}} \theta_l + \epsilon \sum_{p \in \mathcal{P}} (\gamma_{p1} + \gamma_{p2}) \tag{4.4a}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}} \sum_{l : s \in \mathcal{S}_l^0} \pi_{lj0} + \sum_{j \in \mathcal{J}} \sum_{l : s \in \mathcal{S}_l^1} \pi_{lj1} \leq 0 \qquad \forall s \in \mathcal{S} \tag{4.4b}$$

$$\theta_l - \sum_{j \in \mathcal{J}} (\alpha_j^{(t)} \pi_{lj0} + \beta_j^{(t)} \pi_{lj1}) + \sum_{p \in \mathcal{P}} \delta_p^{(t)} (\gamma_{p1} - \gamma_{p2}) \leq m_l^{(t)} \quad \forall l \in \mathcal{L}, \forall t \in \mathcal{T}_l \tag{4.4c}$$

$$\theta_l \quad urs \qquad \forall l \in \mathcal{L} \tag{4.4d}$$

$$\pi_{lsj} \quad urs \qquad \forall l \in \mathcal{L}, \forall s \in \mathcal{S}, \forall j \in \mathcal{J} \tag{4.4e}$$

$$\gamma_{p1}, \gamma_{p2} \leq 0 \quad \forall p \in \mathcal{P} \tag{4.4f}$$

4.3.2 Pricing Problem for Leaf l:

Seeing as there are not many structural changes to the dual formulation of the *Pattern Based* model, we make use of the previous notation introduced in Table 3.5, along with the parameters in Table 4.1. The only change in the Pricing Problems is the objective, which is as follows in Expression 4.5a. We minimize the objective, with respect to constraints (3.4b)-(3.4h) defined in Section 3.3.2

Fair Pricing Problem for Leaf l

$$\text{minimize} \quad \frac{1}{N} \sum_{i \in \mathcal{N}} t_{il} + \sum_{p \in \mathcal{P}} (\gamma_{p2} - \gamma_{p1}) \Big(\frac{\sum_{i \in (+)_p} t_i}{N_{(+)p}} - \frac{\sum_{i \in (-)_p} t_{il}}{N_{(-)p}} \Big) + \sum_{j \in \mathcal{J}} (\pi^*_{lj0} a_{jl} + \pi^*_{lj1} b_{jl}) \tag{4.5a}$$

Chapter 5

Versions of the Modeling Approach

One caveat of our formulations introduced so far is that we leave the split node feature selection to the model. This implies that, in the pattern based formulation even when we provide the set of feasible patterns, the model still needs to decide which split node should use which feature to minimize misclassification.

More intuitively, we could formulate the problems without explicitly defining a decision variable for chosen features on split nodes. This brings forward the issue of ordering, which means each pattern should carry an order component for the 0 and 1 split features. A natural way to is to use a "decompressed" version of our previously introduced patterns. We would put the split-features in an ordered list. For example a pattern previously shown as $\{0 : [j_0], 1 : [j_1]\}$ could be transformed into $[j_0, j_1]$ or $[j_1, j_0]$ which would correspond verbally to either "Split on feature j_0 in the first level, and split on j_1 in the second level" or "Split on feature j_1 in the first level, and split on j_0 in the second level" . Since each leaf has a predesignated list of split-values leading to them, the split-values of the split-features would be known depending on the level the feature appears in.

We started our work with this more *elementary* version of patterns, where pure ordering differences between them could make them register as different patterns. Although split-node decisions are eliminated, the orderings of these combinations of features explodes the size of the problem immensely. To give an example, assume a data set with $J = 15$ features. For a two-depth model, our elementary formulation chooses 4 patterns out of 9000 possible patterns, whereas the symmetry-breaking model deals with choosing 4 out of 690. When we move up a single depth level and assume a three-depth model, this disparity grows much larger with the ratios being 8 out of 27,000 compared to 8 out

of 12,160. Moreover, the latter formulation is much more easily generalized to higher depths. We will discuss the differences these two models imply by showing how this model appears in depth 2.

5.1 Elementary Models

5.1.1 Compact Formulation

Here, we modify the previously introduced sets, parameters, and decision variables slightly to put an emphasis on the leafs and their individual split-features on various levels.

Table 5.1: Decision Variables for the Two-Depth Elementary Decomposable Model.

Decision Variables	
t_{il}	$= \begin{cases} 1, & \text{if data point } i \in \mathcal{N} \text{ falls under the leaf } l \in \mathcal{L} \text{ and is misclassified} \\ 0, & \text{otherwise} \end{cases}$
c_{kl}	$= \begin{cases} 1, & \text{if leaf } l \in \mathcal{L} \text{ is assigned class } k \in \mathcal{K} \\ 0, & \text{otherwise} \end{cases}$
z_{jdl}	$= \begin{cases} 1, & \text{if split feature } j \in \mathcal{J} \text{ is chosen as split variable} \\ & \text{at level } d \in \{1,2\} \text{ for leaf } l \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$

Elementary Two-Depth CompactOCT

$$\text{minimize} \quad \frac{1}{N}\sum_{i\in\mathcal{N}}\sum_{l\in\mathcal{L}} t_{il} \tag{5.1a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{J}} z_{jdl} = 1 \qquad \forall l \in \mathcal{L}, \forall d \in \{1,2\} \tag{5.1b}$$

$$t_{i1} \geq \sum_{j\in\mathcal{J}} x_{ij} z_{j01} + \sum_{j\in\mathcal{J}} x_{ij} z_{j11} - \sum_{k\in\mathcal{K}} y_{ik} c_{k1} - 1 \qquad \forall i \in \mathcal{N} \tag{5.1c}$$

$$t_{i2} \geq \sum_{j\in\mathcal{J}} x_{ij} z_{j02} + \sum_{j\in\mathcal{J}} (1 - x_{ij}) z_{j12} - \sum_{k\in\mathcal{K}} y_{ik} c_{k2} - 1 \qquad \forall i \in \mathcal{N} \tag{5.1d}$$

$$t_{i3} \geq \sum_{j\in\mathcal{J}} (1 - x_{ij}) z_{j03} + \sum_{j\in\mathcal{J}} x_{i,j} z_{j13} - \sum_{k\in\mathcal{K}} y_{i,k} c_{k3} - 1 \qquad \forall i \in \mathcal{N} \tag{5.1e}$$

$$t_{i4} \geq \sum_{j\in\mathcal{J}} (1 - x_{ij}) z_{j04} + \sum_{j\in\mathcal{J}} (1 - x_{ij}) z_{j14} - \sum_{k\in\mathcal{K}} y_{ik} c_{k4} - 1 \quad \forall i \in \mathcal{N} \tag{5.1f}$$

$$\sum_{k\in\mathcal{K}} c_{kl} = 1 \qquad \forall l \in \mathcal{L} \tag{5.1g}$$

$$z_{j11} = z_{j12} \qquad \forall j \in \mathcal{J} \tag{5.1h}$$

$$z_{j21} = z_{j22} \qquad \forall j \in \mathcal{J} \tag{5.1i}$$

$$z_{j13} = z_{j14} \qquad \forall j \in \mathcal{J} \tag{5.1j}$$

$$z_{j23} = z_{j24} \qquad \forall j \in \mathcal{J} \tag{5.1k}$$

$$z_{j11} = z_{j13} \qquad \forall j \in \mathcal{J} \tag{5.1l}$$

$$z_{jdl} \in \{0,1\} \qquad \forall j \in \mathcal{J}, \forall l \in \mathcal{L}, \forall d \in \{1,2\} \tag{5.1m}$$

$$t_{il} \in \{0,1\} \qquad \forall i \in \mathcal{N}, \forall l \in \mathcal{L} \tag{5.1n}$$

The objective is to minimize misclassification as explained by (5.1a). Constraints (5.1b) ensure that for all leafs and levels, exactly one split-feature is chosen among all features. Constraints (5.1c-5.1f) control the flow of data points to leaves based on the split condition. Constraint (5.1g) makes sure each leaf is assigned a single class. The next sets of constraints (5.1h and 5.1i; 5.1j and 5.1k) link sibling leaf pairs together in both levels. Then we link cousin leaves at their only shared feature split level (5.1l). Constraints (5.1m-5.1n) force binary values. We see that, if not for the constraints linking the leaves, this model would be decomposable and easily solvable. Hence, if we were to relax Constraints (5.1h - 5.1l) and apply Lagrangian Decomposition. This would be equivalent to applying column generation to the Dantzig-Wolfe Reformulation.

5.1.2 Elementary Pattern Based IP

A *pattern* denotes a series of split-features represented by α and β (based on the split values) to reach a specified leaf. Hence, patterns are representations of possible decision rules. Each pattern associated with a leaf, has a known misclassification cost, since the split-features implied by the pattern will divide the data according to the split-values of the leaf. Hence, this time, our problem will involve choosing one pattern for each leaf to minimize total misclassification while ensuring the feature-splits of the selected patterns conform with the tree structure. Given that we are able to enumerate all patterns, we can solve the following IP to obtain the patterns that should be selected for each leaf. The additional sets and parameters for this formulation as well as the decision variables is found in Table 5.2.

Table 5.2: Sets, Parameters, and Decision Variables Introduced for the Two-Depth Elementary Pattern Based IP.

Index sets	
$\mathcal{T}_l$	The set of all patterns for leaf $l \in \mathcal{L}$
Parameters	
$\alpha_{jd}^{(t)}$	$= \begin{cases} 1, & \text{If for feature } j \in \mathcal{J},\ x_j = 0 \text{ is the split condition on } d^{th} \text{ level for pattern } t \in \mathcal{T}_l \\ & \text{for } l \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$
$\beta_{jd}^{(t)}$	$= \begin{cases} 1, & \text{If for feature } j \in \mathcal{J},\ x_j = 1 \text{ is the split condition on } d^{th} \text{ level for pattern } t \in \mathcal{T}_l \\ & \text{for } l \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$
$m_l^{(t)}$	misclassification (cost) of pattern $t \in \mathcal{T}_l$, for leaf $l \in \mathcal{L}$
Decision Variables	
$z_l^{(t)}$	$= \begin{cases} 1, & \text{If pattern } t \in \mathcal{T}_l \text{ is used in the tree for leaf } l \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$

Elementary Two-Depth P_OCT

$$\text{minimize} \quad \sum_{l\in\mathcal{L}}\sum_{t\in\mathcal{T}_l} m_l^{(t)} z_l^{(t)} \tag{5.2a}$$

$$\text{s.t.} \quad \sum_{t\in\mathcal{T}_l} z_l^{(t)} = 1 \qquad \forall l \in \mathcal{L} \tag{5.2b}$$

$$\sum_{t\in\mathcal{T}_1} \alpha_{j1}^{(t)} z_l^{(t)} = \sum_{t\in\mathcal{T}_2} a_{j1}^{(t)} z_l^{(t)} \qquad \forall j \in \mathcal{J} \tag{5.2c}$$

$$\sum_{t\in\mathcal{T}_1} \alpha_{j2}^{(t)} z_l^{(t)} = \sum_{t\in\mathcal{T}_2} \beta_{j2}^{(t)} z_l^{(t)} \qquad \forall j \in \mathcal{J} \tag{5.2d}$$

$$\sum_{t\in\mathcal{T}_3} \beta_{j1}^{(t)} z_l^{(t)} = \sum_{t\in\mathcal{T}_4} \beta_{j1}^{(t)} z_l^{(t)} \qquad \forall j \in \mathcal{J} \tag{5.2e}$$

$$\sum_{t\in\mathcal{T}_3} \alpha_{j2}^{(t)} z_l^{(t)} = \sum_{t\in\mathcal{T}_4} \beta_{j2}^{(t)} z_l^{(t)} \qquad \forall j \in \mathcal{J} \tag{5.2f}$$

$$\sum_{t\in\mathcal{T}_1} \alpha_{j1}^{(t)} z_l^{(t)} = \sum_{t\in\mathcal{T}_3} \beta_{j1}^{(t)} z_l^{(t)} \qquad \forall j \in \mathcal{J} \tag{5.2g}$$

$$z_l^{(t)} \in \{0,1\} \qquad \forall t \in \mathcal{T}_l, \forall l \in \mathcal{L} \tag{5.2h}$$

As seen in (5.2a), the objective is to minimize misclassification. Analogous to the previously introduced model, constraints (5.2b) ensure that for all leaves and levels, exactly one split-feature is chosen among all features. Constraints (5.2c-5.2f) link sibling leaf pairs together at all levels. Then we link cousin leaves at their only shared feature split level 5.2g. Constraints (5.2h) are binary constraints. For the LP relaxation of the problem, we modify Constraint (5.2h) as $0 \leq z_l^{(t)} \leq 1 \quad \forall t \in \mathcal{T}_l, \forall l \in \mathcal{L}$.

5.1.3 Branch-and-Price

We propose a B&P scheme to solve the OCT problem. Starting with an initial feasible solution set of patterns, we solve the LP relaxation of *the Pattern-Based IP* Model. Using the optimal dual variable values obtained from our model, we solve the pricing problems. We repeat the same procedure until convergence as standard. We repeat the same procedure until convergence. If the problem is not integral, we branch until integrality. Our branching rule is to split on the first level for which a split variable is not chosen for any leaf, which is different than our previous Branch& Price algorithm.

5.1.3.1 Dual of the LP -Relaxation of the Elementary Pattern-Based Formulation

Now we investigate the dual of the LP relaxation. The notation can be found in Table 5.3

Table 5.3: Decision Variables Introduced for the Dual Problem of the Pattern Based Formulation for the Two-Depth Elementary Pattern Based Model.

Decision Variables	
θ_l	= The dual decision variable associated with leaf $l \in \mathcal{L}$ for constraints (5.2b)
π_f	= The dual decision variable associated with constraints (5.2c-5.2g), with f representing the order of the constraint

Dual of the LP -Relaxation of the Elementary Two-Depth P_OCT

$$\text{maximize} \quad \theta_1 + \theta_2 + \theta_3 + \theta_4 \tag{5.3a}$$

$$\text{s.t.} \quad \theta_1 + \sum_{j\in\mathcal{J}} \alpha_{j1}^{(t)}\pi_{1j} + \sum_{j\in\mathcal{J}} \alpha_{j2}^{(t)}\pi_{2j} + \sum_{j\in\mathcal{J}} \alpha_{j1}^{(t)}\pi_{5j} \leq m_l^{(t)} \qquad \forall t \in \mathcal{T}_1 \tag{5.3b}$$

$$\theta_2 - \sum_{j\in\mathcal{J}} \alpha_{j1}^{(t)}\pi_{1j} - \sum_{j\in\mathcal{J}} \beta_{j2}^{(t)}\pi_{2j} \leq m_l^{(t)} \qquad \forall t \in \mathcal{T}_2 \tag{5.3c}$$

$$\theta_3 + \sum_{j\in\mathcal{J}} \beta_{j1}^{(t)}\pi_{3j} + \sum_{j\in\mathcal{J}} \alpha_{j2}^{(t)}\pi_{4j} - \sum_{j\in\mathcal{J}} \beta_{j1}^{(t)}\pi_{5j} \leq m_l^{(t)} \qquad \forall t \in \mathcal{T}_3 \tag{5.3d}$$

$$\theta_4 - \sum_{j\in\mathcal{J}} \beta_{j1}^{(t)}\pi_{3j} - \sum_{j\in\mathcal{J}} \beta_{j2}^{(t)}\pi_{4j} \leq m_l^{(t)} \qquad \forall t \in \mathcal{T}_4 \tag{5.3e}$$

$$\theta_1, \theta_2, \theta_3, \theta_4, \pi_{1j}, \pi_{2j}, \pi_{3j}, \pi_{4j}, \pi_{5j} \quad urs \quad \forall j \in \mathcal{J} \tag{5.3f}$$

We see that the dual is decomposable into 4 different subproblems that are independent of each other. Hence, dual feasibility can be checked by considering whether or not none of these 4 sets of constraints are violated. Each pricing problem will deal with a single leaf $l \in \mathcal{L}$.

5.1.3.2 Pricing Problem for Leaf l:

In order to generate patterns that possibly violate dual feasibility, we introduce the following notation found in Table 5.4 and formulation:

Table 5.4: Parameters and Decision Variables Introduced for the Pricing Problems of the Two-Depth Elementary Pattern Based Formulation.

Parameters	
S_d^l	$= \begin{cases} 1, & \text{If the split value is 1 on level } d \in \{1,2\} \text{ for leaf } l \\ 0, & \text{otherwise} \end{cases}$
θ_l^*	The dual decision variable values associated with leaf $l \in \mathcal{L}$ for constraints (5.2b)
η_{ldj}^*	The dual variable values associated with leaf l, for level $d \in \{1,2\}$ for feature $j \in \mathcal{J}$
v_{idj}^l	$= 1 - S_d^l + (-1)^{(S_d^l+1)} * x_{ij} \qquad \forall i \in \mathcal{N}, d \in \{1,2\}, j \in \mathcal{J}$
	An indicator coefficient derived using split-values of leaf l and feature entries of data points, used in the data-flow constraints
Decision Variables	
t_{il}	$= \begin{cases} 1, & \text{if data point } i \in \mathcal{N} \text{ falls under leaf } l \text{ and is misclassified} \\ 0, & \text{otherwise} \end{cases}$
a_{jd}	$= \begin{cases} 1, & \text{if feature } j \in \mathcal{J} \text{ is selected as the split-feature at level } d \in \{1,2\} \text{ for leaf } l \\ 0, & \text{otherwise} \end{cases}$
c_{kl}	$= \begin{cases} 1, & \text{if leaf } l \text{ is assigned class } k \in \mathcal{K} \\ 0, & \text{otherwise} \end{cases}$

Elementary Two-Depth Pricing Problem for Leaf l

$$\text{minimize} \quad \frac{1}{N}\sum_{i\in\mathcal{N}} t_i - \sum_{j\in\mathcal{J}} \eta_{l1j}^* a_{j1} - \sum_{j\in\mathcal{J}} \eta_{l2j}^* a_{j2} \tag{5.4a}$$

$$\text{s.t.} \quad \sum_{j\in p} a_{j1} = 1 \tag{5.4b}$$

$$\sum_{j\in\mathcal{J}} a_{j2} = 1 \tag{5.4c}$$

$$\sum_{k\in\mathcal{K}} c_{kl} = 1 \tag{5.4d}$$

$$t_i \geq \sum_{j\in\mathcal{J}} v_{i1j}^{(l)} a_{j1} + \sum_{j\in\mathcal{J}} v_{i2j}^{(l)} a_{j2} - y_{ik} c_k - 1 \qquad \forall k \in \mathcal{K}, \forall i \in \mathcal{N} \tag{5.4e}$$

$$t_{il} \in \{0,1\} \qquad \forall i \in n \tag{5.4f}$$

$$c_{kl} \in \{0,1\} \qquad \forall k \in \mathcal{K} \tag{5.4g}$$

$$a_{j1}, a_{j2} \in \{0,1\} \qquad \forall j \in \mathcal{J} \tag{5.4h}$$

Each pricing problem's objective is implied by the violation of each constraint in the dual problem. We use η^*_{ldj} to represent the linear combinations of the optimal dual variable values π^*_f, as defined in Table 5.3. As mentioned above, we consider only one leaf $l \in \mathcal{L}$ in each pricing problem, separately. Here, the first two constraints ensure the selection of a split feature for each level (Constraints 5.4b, 5.4c). The next constraint assigns our leaf l to a single class (Constraint 5.4d). Since our problem is to minimize, we define a logical constraint setting a lower bound on the value of t_i based on whether the data point falls under leaf l (Constraint 5.4e). We conclude each pricing problem with the usual binary constraints (Constraints 5.4f, 5.4g, and 5.4h). We note that the relaxation of binary $t_i l$ also yields a binary outcome for $t_i l$ at optimality.

Again, although we started with the pattern notion that included depth, as defined above, we realized that their permutative nature makes the set of all feasible patterns explode as depth grows. We deem *symmetry breaking* through the use of split-node variables as the better way to approach our modeling, and continue with this idea from now on. As already noted, the models introduced in the previous chapters made use of symmetry breaking.

5.2 A Tighter Formulation

As explained above, we move forward with the symmetry breaking formulation. However, we were also curious as to whether it was be possible to formulate the IP in a different manner, which would yield integral values for the LP relaxation. This would immensely help computation times, as branching would be unnecessary and we would finish at root node after completing the necessary column generation iterations.

In our literature review, we see there are ways to tighten the formulation with a set of valid inequalities/equalities [14, 24]. One of them being the addition of a set of constraints, which assure that each data point should fall under a single pattern. So, in the context of our Pattern Based Model, introduced in Chapter 3, this would imply the inclusion of the following parameter (shown in Table 5.5) and constraint set (5.5) in the model:

Table 5.5: New Parameters Introduced for the LP Tightening Cuts.

Parameters	
$\rho_i^{(t)}$	$= \begin{cases} 1, & \text{if data point } i \in \mathcal{N} \text{ falls under leaf } l \text{ for pattern } t \in \mathcal{T}_l \\ 0, & \text{otherwise} \end{cases}$

$$\sum_{t \in \mathcal{T}_l} \rho_i^{(t)} z_l^{(t)} = 1 \qquad \forall l \in \mathcal{L}, \quad \forall i \in \mathcal{N} \tag{5.5}$$

It is easily understood that constraints (5.5) are implied by constraints (3.2b) and (3.2e) in the original model. Yet, when appended to the LP relaxation, constraints (5.5) help eliminate the choice of multiple patterns for the same leaf with non-integral $z_l^{(t)}$, preferred by the LP relaxation as they prompt lesser total misclassification values. However, the addition of all these cuts is rather costly, as we add as many constraints as the number of data points, multiplied by the number of leafs L in the tree. The computational performance of these cuts will be discussed in Chapter 7, along with other computational metrics for all methods.

Chapter 6

Branch-and-Price Implementation

Using the master problem and pricing problems derived in the previous sections, we define the Branch& Bound algorithm. Our branching rule is to split the search space according to whether a feature j is used in split node w of the classification tree. If multiple branching node feature decision variable values are non-integral, we choose the node at the highest point of the tree to split on.

As seen in pricing problem models introduced in the previous sections, our formulation allows pricing problems to be independent of the split-node decision variables, w_{js}. Hence, we note that our branching schema does not add additional constraints to the pricing problem, but it modifies the master problem. Thus, we solve the same pricing problem across all nodes of the Branch & Price tree, with re-evaluated objective coefficients retrieved from the dual of the constrained master problem that considers the branching history.

We make a number of improvements to increase the efficiency of the Brach-and-Price algorithm, which can be summarized under the headings: *Pooling*, *Parallelization*, *Warm-Up Heuristic*, and *P_OCT Incumbent Search*.

Pooling. Pricing problems are costly to solve and the increase in solution times is exponentially linked with the number of data points and number of features considered. Thus, we make use of the solution pool feature to store more than only the optimal solution during the optimization process. We keep 4 solutions in the pool, and use the pool parameter 2. Based on the descriptions found in Gurobi Documentations, this implies that optimal solution is searched, but solutions found during the optimization process are stored in memory [25]. This does increase optimization time, however, it can

also decrease the number of column generation instances needed for convergence.

Parallelization. Due to the separable nature of the individual pricing problems belonging to each leaf, they can be handled simultaneously. Hence, in our implementation, we solve each leaf's pricing problem in parallel. Once a new pattern is generated for a leaf, it would be added to the corresponding set. After every pricing problem is solved to optimality, the master problem is resolved with the added patterns. Each node is independent of another as their relevant branching history is self-contained. This nature of B&P nodes enables us to process them simultaneously. The number of maximum nodes to process is a parameter that must be adjusted since handling too many nodes at once may result in the tree growing rapidly, increasing the overall computational time. Currently, we handle at most 4 nodes at once.

Warm-Up Heuristic. Since the solution time of each pricing problem may be high when using the pricing IP, we devise a warm-up heuristic to generate patterns. For each leaf, we start with an empty pattern. One feature is "chosen" at a time as a split variable for each level in appearing in leaf l. To mimic the objective function of the pricing problem, the sum of associated dual variable values for the leaf/ feature/ split value combination is subtracted from the one-step cost (misclassification) function of this incomplete pattern. At iteration q, partial patterns with the best n_q values are added to the set of incomplete patterns. Data is recursively treated as it will be split only one more time on top of its previous splits. This enables the creation of $n_1 \times n_2 \times \ldots \times n_D$ patterns. If they violate dual feasibility, they are added to their prospective leaves. We run the heuristic pricing loop until no patterns violating dual feasibility are found. Then we start the pricing IP loop.

The pricing IP loop is boosted with heuristic patterns at each iteration. Once at the IP pricing loop, we run the heuristic algorithm described above with the current iteration's π^* values. With the same π^*, we also solve the pricing IPs to optimality. Hence, we increase the number of patterns added at each iteration, decreasing the number of iterations needed for convergence. Moreover, we implement an early stop mechanism, where if the lower bound obtained from the sum of the pricing problem objectives starts plateauing after many consecutive iterations, we terminate the pricing loop. The sum of the pricing problem objectives is used as a lower bound for the node.

P_OCT Incumbent Search.We note that Full Patterns IP (P_OCT) Model is manageable up to a certain depth, depending on the dataset. We take advantage of this by fully enumerating all patterns once the number of features fixed to be used in the splits

surpasses a threshold. Since we always branch on the feature at the highest unfixed level for a leaf, enumeration of the remaining levels' patterns, and concatenation of them to the other leaves' pattern sets enables us to solve the restricted IP and prune the node. In terms of incumbent solutions, we start with the CART solution when initializing the B&P tree. Furthermore, at each node, we solve the restricted IP with existing patterns to seek better incumbent solutions. As a side note, in addition to using the CART solution as an incumbent solution at the start of the B&P tree, we also use it as a starting solution in our Full Patterns IP method as well.

With the implementation of the improvement features introduced above, we obtain the final versions of our 3 models *the Compact* Model (*CompactOCT*), *the Full Patterns IP* Model (*P_OCT*), and this augmented B&P algorithm, *Branch-and-Price Optimal Classification Trees*, or briefly *BP_OCT*. Their computational performance on data sets of varying sizes will be discussed in Chapter 7.

Chapter 7

Computational Experiments

After defining our experimental setup, and noting the usefulness of the symmetry breaking approach in our models, we will report our findings under six main headings,: In the first section, we will show the contribution of CompactOCT, by comparing its results in terms of training time, accuracy, and reported optimization gap to BendersOCT. In the second section, we will display the potential of Branch&Price by comparing P_OCT with BP_OCT, and investigate whether we can declare BP_OCT as useful depending on certain problem properties. Based on observations, we propose a hybrid (compound) approach that uses P_OCT or BP_OCT based on the data set properties, which we call DWR. Thirdly, we will compare CompactOCT and DWR results, and see if our approach can help solve the cases where CompactOCT is not as useful. Then, we note the shortcomings of the tighter formulation introduced in Chapter 5. Lastly, we present computational results on how our fairness-aware models work in terms of a trade-off between accuracy and ODM.

As a general note, under the following headings, there are two types of results presented. First, we report training results, and highlight (bolder font) the computational time of the method that either finds the optimal solution faster, or finds a better solution in terms of training accuracy within the specified time limit. In a separate table, we highlight the method with better test accuracy.

7.1 Experimental Setup

The results reported in the literature indicated that FlowOCT, BinOCT, and BendersOCT fail to find solutions in certain datasets as depth exceeds 2 [26]. Our motivation is to report exact solutions in these particularly hard to solve instances. In a majority of the data sets showcased by [26], FlowOCT and BinOCT were outperformed by BendersOCT with respect to computational time, hence we benchmark against BendersOCT. We made use of the *ODTLearn* package recently published by Vossler et al., which is an implementation of the previous work of Aghai et al. on FlowOCT and BendersOCT as well as Verwer and Zhang's work on BinOCT [10, 11, 13]. In addition to BendersOCT results, we will report CompactOCT's, P_OCT's, and BP_OCT's performance with maximum depths set to 2, 3, and 4.

For our experiments, we primarily used a Macbook Pro 2020 with an M1 Chip and a running memory of 8GB (for CompactOCT, BP_OCT, and P_OCT). BendersOCT runs were taken on a computer with 32 GB RAM and an Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz 3.00 GHz chip due to a compatibility issue with our primary computing unit. For the MIP models, we use Gurobi Solver Version 10.0.3. We note that, for *BP_OCT*, a computer with an apple silicone chip may need to be used to fully exploit our parallelization technique.

The size properties of our data sets are found in Table 7.1. All of these data sets are available on the Machine Learning Repository maintained by University of California, Irvine [27]. Using [26]'s approach, we binarized the available data and split it into different training and test folds [26]. We processed all data sets in 10 folds, and tallied the performance of our algorithm on each individual fold. If tried methods reported results for the majority of the folds in each data set, we reported the results. Additionally, to observe our improvement on top of the CART solution, we used the decision trees package on the binarized data folds to get baseline CART training and test accuracy, via the scikit-learn library [28]. The training and test accuracies of the CART solution can be seen in Table 7.2.

All of our methods have a time limit of 3600 seconds (s). However, in CompactOCT and P_OCT, optimization time is constrained, and construction time is not included. In BP_OCT, generation of new nodes and patterns is prohibited after time limit is reached. The in-progress nodes are processed, even if the time limit is exceeded.

Table 7.1: Dataset Information, note that feature numbers are reflective of those after binarization. Moreover, for each fold number of data points are given approximately.

Dataset	# of Data Points	# of Features	# of Classes
adult	43957	132	2
agaricus-lepiota	7311	112	2
balance-scale	562	20	3
banknote-authentication	1234	40	2
car-evaluation	1555	21	4
diabetes	691	72	2
haberman	275	25	2
kr-vs-kp	2876	38	2
monks-1	500	15	2

Dataset	# of Data Points	# of Features	# of Classes
monks-2	540	15	2
monks-3	498	15	2
nursery	11664	26	5
seismic-bumps	2325	72	2
tae	135	32	2
tic-tac-toe	862	27	2
titanic	801	178	2
wdbc	512	300	2
wine	160	130	3

Table 7.2: CART Accuracy by depth for each data set

Dataset	Depth	Training Accuracy	Test Accuracy
adult	2	0.791	0.791
	3	0.821	0.82
	4	0.822	0.822
agaricus-lepiota	2	0.954	0.954
	3	0.985	0.985
	4	0.994	0.992
balance-scale	2	0.684	0.664
	3	0.699	0.658
	4	0.713	0.651
banknote-authentication	2	0.723	0.709
	3	0.789	0.781
	4	0.849	0.848
car-evaluation	2	0.778	0.778
	3	0.807	0.79
	4	0.817	0.796
diabetes	2	0.747	0.735
	3	0.753	0.726
	4	0.759	0.721
haberman	2	0.735	0.735
	3	0.749	0.68
	4	0.778	0.69
kr-vs-kp	2	0.776	0.768
	3	0.904	0.904
	4	0.941	0.941
monks-1	2	0.746	0.746
	3	0.816	0.798
	4	0.846	0.82

Dataset	Depth	Training Accuracy	Test Accuracy
monks-2	2	0.657	0.657
	3	0.662	0.631
	4	0.685	0.632
monks-3	2	0.964	0.964
	3	0.989	0.989
	4	0.989	0.989
nursery	2	0.764	0.763
	3	0.825	0.825
	4	0.853	0.853
seismic-bumps	2	0.934	0.934
	3	0.935	0.93
	4	0.938	0.929
tae	2	0.835	0.795
	3	0.865	0.775
	4	0.888	0.761
tic-tac-toe	2	0.707	0.666
	3	0.755	0.725
	4	0.839	0.821
titanic	2	0.789	0.772
	3	0.823	0.819
	4	0.843	0.826
wdbc	2	0.829	0.807
	3	0.909	0.888
	4	0.93	0.9
wine	2	0.593	0.5
	3	0.698	0.578
	4	0.785	0.68

7.2 Note on the Success of the Symmetry Breaking Formulation

As it was explained in Chapter 5, symmetry breaking formulation, or the main formulation proposed in this discussion, is more sensible to use in terms of scalability and clarity. Our initial results with the elementary formulation for a selected portion of our tested data sets, represented by their average over 10 folds, can be found under A, in Tables A.1 and A.2. As it can be seen from the tables, our verdict to define split-node features as decision variables is justified computationally. We achieve generally noticably lesser runtimes, and comparable or higher accuracies across all data sets.

7.3 Compact Formulation vs. BendersOCT

Table 7.3: CompactOCT metrics obtained after training compared with BendersOCT metrics for all data sets

Dataset	Depth	CompactOCT			BendersOCT		
		Time (s)	Training Accuracy	Opt Gap	Time (s)	Training Accuracy	Opt Gap
		mean ± std	*mean ± std*	*mean ± std*	*mean ± std*	*mean ± std*	*mean ± std*
adult	2	3601.3±1.7	0.771±0.009	100	**3617.6±5.6**	0.773±0.011	29.2±1.8
	3	**3601.8±0.3**	0.765±0.008	100	3647±26.3	0.764±0.007	30.9±1.3
	4	**3606.7±7.4**	0.741±0.02	100			
agaricus-lepiota	2	1349.4±337.5	0.969±0	0	**316.9±43.3**	0.969±0	0
	3	3600.3±0.1	0.979±0.007	100	**3603.9±5.6**	0.984±0.009	1.5±0.9
	4	3348±715.6	0.972±0.018	87.5±35.3	**3617±0**	0.996±0	0.3±0
balance-scale	2	**1.7±0.4**	0.686±0.003	0	18.9±11.5	0.686±0.003	0
	3	**28.6±1.9**	0.745±0.001	0	2813.1±1154.8	0.744±0.003	0.7±0.9
	4	**2630.4±784.6**	0.787±0.001	1.9±3	3601.4±0.8	0.765±0.008	20.6±6.4
banknote-authentication	2	**0.3±0**	0.739±0.002	0	13.5±4.5	0.809±0.025	0
	3	**3.2±0.6**	0.816±0.003	0	14.8±3.4	0.816±0.003	0
	4	**18.6±3.2**	0.895±0.004	0	23.1±27.3	0.824±0.023	0
car-evaluation	2	**16.3±2.8**	0.777±0.002	0	92.9±33.9	0.777±0.002	0
	3	**1108.1±454.1**	0.814±0.001	0	3602.2±1.1	0.81±0.004	14±2.8
	4	**3600±0**	0.836±0.005	99.9±00.2	3604.2±2.4	0.822±0.012	21.6±1.8
diabetes	2	**5.1±1**	0.749±0.004	0	39.5±28.5	0.749±0.004	0
	3	**727.2±241.9**	0.769±0.005	0	3346.4±553.8	0.767±0.007	3±3.7
	4	**3600±0**	0.783±0.007	50±12.5	3601.4±0.6	0.771±0.01	20.9±2.3
haberman	2	**0.5±0.1**	0.761±0.006	0	27.5±14.5	0.761±0.006	0
	3	**18.4±5.7**	0.782±0.005	0	2885±888.2	0.782±0.006	0.6±0.9
	4	**434.6±120.7**	0.807±0.004	0	3603.4±3.5	0.798±0.005	12±1.6
kr-vs-kp	2	98.4±21.2	0.869±0.003	0	**97.6±28.8**	0.869±0.003	0
	3	3518.9±256.4	0.935±0.007	78.1±31.3	**3348.5±770.3**	0.934±0.011	3.7±3.4
	4	**3600.1±0**	0.933±0.011	100	3606.8±7.2	0.929±0.026	7.7±3.2
monks-1	2	**0.7±0**	0.78±0.003	0	7.4±2.7	0.78±0.003	0
	3	**6.2±2**	0.897±0.003	0	412.4±230	0.897±0.003	0
	4	**1.9±0.9**	1±0	0	29.3±17.4	1	0
monks-2	2	**2.9±0.6**	0.657±0	0	47.7±18.6	0.657±0	0
	3	**1102.2±343.4**	0.677±0.005	0	3602.5±2.5	0.675±0.004	38.8±6.4
	4	**3600±0**	0.713±0.006	71.8±0.067	3604.4±2.2	0.687±0.007	45.5±1.5
monks-3	2	**0.1±0**	0.963±0.001	0	4±1.4	0.963±0.001	0
	3	**3.1±0.5**	0.989±0.001	0	302.2±248	0.989±0.001	0
	4	**3196.3±866.6**	0.989±0.001	34.8±20.6	3603.2±2.1	0.989±0.001	1±0.1
nursery	2	**594.9±162.3**	0.763±0	0	608.8±137.8	0.763±0	0
	3	**3600.1±0**	0.813±0.013	100	3602.3±1.6	0.808±0.016	23.7±2.4
	4	**3600.4±0**	0.79±0.017	100	3603.5±0	0.721±0	38.6±0
seismic-bumps	2	**337.1±82.2**	0.935±0	0	2950±1000	0.935±0	0.4±0.6
	3	3600±0	0.936±0	100	**3603.4±2.4**	0.936±0	6.8±0
	4	3600.1±0	0.936±0	100	**3608.3±5.8**	0.936±0.001	6.8±0.1
tae	2	**0.5±0.1**	0.844±0.005	0	21.7±15.9	0.844±0.005	0
	3	**12.7±3.3**	0.89±0.006	0	394±325.4	0.89±0.006	0
	4	**978.2±381.8**	0.937±0.006	0	3601±0.9	0.935±0.007	5.5±1.2
tic-tac-toe	2	**25.2±4.8**	0.709±0.003	0	537±403.4	0.709±0.003	0
	3	**3515.3±183.9**	0.78±0.002	28.1±25.2	3602.1±1.8	0.766±0.01	26.8±6
	4	**3600±0**	0.846±0.008	100	3604±2.9	0.796±0.031	25.7±4.9
titanic	2	**13.5±5.3**	0.811±0.003	0	75.6±68.3	0.811±0.003	0
	3	**1938.9±739.6**	0.836±0.003	0	3317.4±879.7	0.834±0.003	9.2±6
	4	**3600.1±0**	0.844±0.004	100	3603.8±4.4	0.84±0.005	18.9±0.8
wdbc	2	**0.9±0.2**	0.83±0.004	0	4.5±1.5	0.83±0.004	0
	3	40.4±15.5	0.921±0.004	0	**22.9±11.2**	0.921±0.004	0
	4	**1905.1±552**	0.968±0.003	0	2792.8±990.7	0.968±0.003	00.1±00.1
wine	2	**0.2±0**	0.62±0.006	0	1.6±0.6	0.62±0.006	0
	3	**1.5±0.7**	0.716±0.007	0	10.7±5	0.716±0.007	0
	4	**11.7±6.7**	0.802±0.005	0	17.2±8.7	0.8±0.006	0

Table 7.3 shows the results of pairwise comparison between CompactOCT and BendersOCT. CompactOCT outperforms BendersOCT in about 77% of all tried instances on average. We especially observe the contribution of CompactOCT on the following datasets, as it retrieves the optimal solution much faster: balance-scale, car-evaluation, diabetes, haberman, monks-1, nursery, titanic, and wine (50% of all data sets). In 37.5% of all data sets, CompactOCT performs mostly better or comparable in lesser depths, and retrieves a better solution in depth 4. This alone helps us detect the useful nature of CompactOCT. The test accuracies of CompactOCT can be seen in Table 7.4, which are also mostly consistent with training accuracies, showing that higher training accuracies

Table 7.4: Test Accuracies of CompactOCT vs. BendersOCT

Dataset	Depth	Test Accuracy	
		CompactOCT	BendersOCT
		mean ± std	*mean ± std*
adult	2	**0.771±0.009**	0.774±0.013
	3	**0.766±0.009**	0.764±0.009
	4	**0.761±0.001**	
agaricus-lepiota	2	0.969±0.008	0.969±0.008
	3	0.979±0.006	**0.985±0.009**
	4	0.973±0.018	**0.991±0**
balance-scale	2	**0.669±0.016**	0.635±0.032
	3	**0.705±0.015**	0.678±0.028
	4	0.724±0.029	**0.729±0.054**
banknote-authentication	2	0.739±0.017	**0.797±0.03**
	3	0.804±0.024	**0.805±0.025**
	4	**0.89±0.035**	0.815±0.046
car-evaluation	2	0.777±0.021	0.777±0.021
	3	0.789±0.014	**0.799±0.027**
	4	**0.822±0.022**	0.811±0.023
diabetes	2	**0.746±0.039**	0.733±0.046
	3	**0.757±0.038**	0.726±0.049
	4	**0.77±0.028**	0.73±0.06
haberman	2	**0.748±0.027**	0.689±0.053
	3	**0.751±0.025**	0.689±0.05
	4	**0.775±0.042**	0.686±0.037
kr-vs-kp	2	**0.869±0.026**	0.869±0.026
	3	**0.936±0.015**	0.934±0.016
	4	**0.936±0.024**	0.927±0.026
monks-1	2	**0.74±0.028**	0.739±0.031
	3	0.863±0.026	**0.868±0.029**
	4	1±0	1±0

Dataset	Depth	Test Accuracy	
		CompactOCT	BendersOCT
		mean ± std	*mean ± std*
monks-2	2	**0.663±0.012**	0.657±0.008
	3	**0.668±0.022**	0.582±0.03
	4	**0.743±0.031**	0.554±0.068
monks-3	2	**0.965±0.013**	0.964±0.016
	3	0.989±0.012	0.989±0.012
	4	**0.992±0.013**	0.989±0.012
nursery	2	0.763±0.008	0.763±0.008
	3	**0.811±0.02**	0.806±0.026
	4	**0.789±0.021**	0.723±0
seismic-bumps	2	**0.934±0**	0.93±0.004
	3	**0.934±0.001**	0.929±0.003
	4	**0.934±0**	0.931±0.002
tae	2	**0.821±0.031**	0.794±0.049
	3	**0.887±0.054**	0.834±0.072
	4	**0.919±0.044**	0.848±0.074
tic-tac-toe	2	**0.707±0.04**	0.67±0.034
	3	**0.747±0.03**	0.725±0.05
	4	**0.834±0.032**	0.761±0.05
titanic	2	0.811±0.026	0.811±0.026
	3	**0.837±0.025**	0.831±0.029
	4	**0.835±0.023**	0.812±0.025
wdbc	2	**0.79±0.03**	0.789±0.032
	3	0.896±0.062	**0.899±0.046**
	4	**0.949±0.034**	0.926±0.033
wine	2	0.483±0.06	**0.494±0.071**
	3	**0.585±0.065**	0.573±0.07
	4	**0.708±0.034**	0.703±0.096

with CompactOCT, generally translate to higher test accuracies.

Some other highlights from Table 7.3 include CompactOCT's performance on haberman (and tae) data set(s), in which CompactOCT has been able to retrieve optimal solutions respectively at worst 9 times (and 4 times) faster than BendersOCT across all depths. We must also note CompactOCTs performance on balance-scale (and wdbc) data set(s) as well, in which CompactOCT is at worse 1.5 (and 2 times) faster than BendersOCT.

Table 7.3 also shows some instances where CompactOCT is not as successful. For example, agaricus-lepiota data set, in which both CompactOCT and BendersOCT times out, however, BendersOCT finds a better solution with with higher accuracy.

Based on Table 7.3, we see that CompactOCT performs better than BendersOCT on a majority of the data sets across all tried depths. On the data sets its not the better petrforming method in all depths, it still tends to perform better than BendersOCT as depth grows. Thus, we can conclude that CompactOCT is better than BendersOCT computationally.

7.4 Full Patterns IP (P_OCT) vs. Branch-and-Price (BP_OCT)

On the introduced datasets given in Table 7.1, we ran both the Full Patterns IP (P_OCT) and ($BP\_\ OCT$). Results are seen in Tables 7.5 and 7.6. The datasets with missing values imply that results could not be retrieved for P_OCT or the BP_OCT due to computational crashes for some depths. In those instances, we deem the method returning any answer as the best performing method by default. Moreover, for the results obtained at the Root Node of BP_OCT, please refer to Table B.1, found in Appendix B.

For visualization purposes, example Branch-and-Price trees to solve two-depth models generated by our methods on three data sets can be found in Figure C.1 in Appendix C. Green represents integral nodes, red represents pruned nodes, and yellow represents infeasible nodes. Branching variables are shown on the arcs between nodes in the form of, feature, split-node number, and fixed value. Also, to give an idea of how many column generation iterations are needed after the heuristic iterations defined in Chapter 6, please refer to Figure D.1 in Appendix D which provides the iteration counts for the nodes in the trees shown in Figure C.1. Here, we also highlight the importance of using the heuristic warm-up procedure, defined in Chapter 6. In Appendix D, Figure D.2 displays how many column generation iterations would be needed when the warm-up procedure was not applied. For another comparison, please see how the Branch-and-Price trees would change without the warm-up procedure in Figure D.3.

When observing the results from Table 7.5, we see the benefit of applying decomposition and column generation on the following datasets, where as depth grows, BP_OCT beats P_OCT: adult, agaricus-lepiota, banknote-authentication, diabetes, haberman, kr-vs-kp, seismic-bumps, tae, tic-tac-toe, titanic, wdbc, and wine (75% of all data sets). Test accuracies are also reported in Table 7.6. We also see that, when we are able to generate and store all defined patterns to build P_OCT, P_OCT returns solutions overwhelming faster . However, the issue of scaling up to greater depths still persists in some data sets. In such cases, working with a specially selected set of patterns could be of use.

Using the runtime and training accuracy information we obtained after running both P_ OCT and BP_OCT over 10 folds for each dataset, we checked if it is possible to estimate whether or when BP_OCT would outperform P_OCT. For this, utilizing a simple decision tree, we observed that if the maximum tree depth allowed, $D \geq 3$ and the number of

Table 7.5: BP_OCT metrics obtained after training compared with P_OCT metrics for all data sets

Dataset	Depth	BP_OCT			P_OCT		
		Time (s)	**Training Accuracy**	**Opt Gap (%)**	**Time (s)**	**Training Accuracy**	**Opt Gap**
		mean ± std	*mean ± std*	*mean ± std*	*mean ± std*	*mean ± std*	*mean ± std*
adult	2	3776.7±180.7	0.791±0	100±0	**14.7±7.7**	0.809±0	0±0
	3	**5063.1±19.8**	0.82±0	100±0			
	4						
agaricus-lepiota	2	2734.6±184.1	0.969±0	0±0	**8±0**	0.969±0	0±0
	3	**3610.9±34.1**	0.985±0	100±0			
	4						
balance-scale	2	8.9±0.5	0.686±0.003	0±0	**0±0**	0.686±0.003	0±0
	3	200.1±13.5	0.745±0.003	0±0	**71.7±16.9**	0.745±0.003	0±0
	4						
banknote authentication	2	5±1.9	0.739±0.002	0±0	**0±0**	0.739±0.002	0±0
	3	197.7±30.1	0.816±0.003	0±0	**27.2±5.2**	0.816±0.003	0±0
	4	**3768.5±142.8**	0.849±0.022	32.8±11.2			
car-evaluation	2	5±1.9	0.739±0.002	0±0	**0±0**	0.739±0.002	0±0
	3	197.7±30.1	0.816±0.003	0±0	**27.2±5.2**	0.816±0.003	0±0
	4	3661.8±32.7	0.817±0.006	100±0	**3600.3±0**	0.839±0.004	100±0
diabetes	2	28±1.9	0.777±0.002	0±0	**0±0**	0.777±0.002	0±0
	3	**162.8±24.2**	0.753±0.007	74.2±1.1			
	4	**3945±188.2**	0.759±0.01	93.3±2.4			
haberman	2	292.9±49.8	0.749±0.004	0±0	**0.4±0**	0.749±0.004	0±0
	3	**416.7±46.9**	0.782±0.005	0±0	434.3±285.5	0.782±0.005	0±0
	4	**3648.7±27.9**	0.778±0.006	82.2±1.7			
kr-vs-kp	2	122±15.9	0.869±0.003	0±0	**0.1±0**	0.869±0.003	0±0
	3	1986±136.6	0.938±0.001	0±0	**448.6±316.1**	0.938±0.001	0±0
	4	**3747.8±136.8**	0.94±0.001	100±0			
monks-1	2	3.6±0	0.78±0.003	0±0	**0±0**	0.78±0.003	0±0
	3	36.8±2	0.897±0.003	0±0	**4.3±0.6**	0.897±0.003	0±0
	4	3550.2±148.9	0.842±0.01	60±54.7	**8±2.4**	1±0	0±0
monks-2	2	4.3±0.2	0.657±0	0±0	**0.1±0**	0.657±0	0±0
	3	165.2±13.7	0.677±0.005	0±0	**110.7±46.6**	0.677±0.005	0±0
	4						
monks-3	2	0.6±0	0.963±0.001	0±0	**0±0**	0.963±0.001	0±0
	3	27.7±1.7	0.989±0.001	0±0	**1.6±0.3**	0.989±0.001	0±0
	4						
nursery	2	737.9±50.1	0.763±0.001	0±0	**0±0**	0.763±0.001	0±0
	3	2017.7±474.8	0.837±0	0±0	**193.5±63.6**	0.837±0	0±0
	4	3733.3±59.9	0.853±0.001	100±0	**3600.7±0.3**	0.86±0.007	100±0
seismic-bumps	2	1029±76.2	0.935±0	0±0	**20.6±12.4**	0.935±0	0±0
	3	**3600±0**	0.935±0.001	100±0			
	4	**3760.9±17.4**	0.937±0	100±0			
tae	2	11.3±2.3	0.844±0.005	0±0	**0.2±0**	0.844±0.005	0±0
	3	**674.5±76.8**	0.89±0.006	0±0	747.6±295.9	0.89±0.006	0±0
	4	**3686.3±45.9**	0.888±0.017	100±0	3600.4±0.1	0.868±0.022	100±0
tic-tac-toe	2	28.6±2.1	0.708±0.003	0±0	**0.4±0**	0.709±0.003	0±0
	3	**589±23.8**	0.781±0.003	0±0	2219.8±491.7	0.781±0.003	0±0
	4	**3675±45.4**	0.839±0.009	100±0	3600.2±0.2	0.802±0.032	100±0
titanic	2	676.5±99.1	0.811±0.003	0±0	**10.2±5.1**	0.811±0.003	0±0
	3	**3600.2±0**	0.823±0.008	100±0			
	4						
wdbc	2	734.4±198.3	0.83±0.004	0±0	**9.1±0.5**	0.83±0.004	0±0
	3	**3748.3±105.8**	0.907±0.008	40.1±10			
	4						
wine	2	15.1±11.3	0.618±0.007	0±0	**0.2±0**	0.62±0.006	0±0
	3	**3636.4±21.6**	0.715±0.006	0±0			
	4						

Table 7.6: Test Accuracies of BP_OCT Vs. P_OCT

Dataset	Depth	Test Accuracy	
		BP_OCT	P_OCT
		mean ± std	*mean ± std*
adult	2	0.791±0.003	**0.809±0.003**
	3	**0.791±0.003**	
	4		
agaricus-lepiota	2	0.969±0.008	0.969±0.008
	3	**0.948±0.009**	
	4		
balance-scale	2	0.657±0.03	**0.66±0.029**
	3	**0.705±0.029**	0.705±0.029
	4		
banknote_authentication	2	0.739±0.017	0.739±0.017
	3	**0.807±0.025**	0.805±0.025
	4	**0.773±0.037**	
car-evaluation	2	0.739±0.017	0.739±0.017
	3	**0.807±0.025**	0.805±0.025
	4	0.784±0.025	**0.829±0.025**
diabetes	2	0.777±0.021	0.777±0.021
	3	**0.755±0.044**	0±0
	4	**0.748±0.034**	
haberman	2	0.743±0.036	**0.748±0.034**
	3	0.744±0.015	**0.748±0.023**
	4	**0.82±0.061**	
kr-vs-kp	2	**0.869±0.026**	0.869±0.026
	3	**0.938±0.014**	0.938±0.014
	4	**0.809±0.027**	
monks-1	2	0.74±0.028	0.735±0.031
	3	0.863±0.026	0.866±0.029
	4	0.863±0.037	**1±0**

Dataset	Depth	Test Accuracy	
		BP_OCT	P_OCT
		mean ± std	*mean ± std*
monks-2	2	**0.66±0.011**	0.668±0.017
	3	0.667±0.02	**0.677±0.032**
	4		
monks-3	2	0.964±0.016	**0.964±0.016**
	3	0.989±0.012	0.989±0.012
	4		
nursery	2	0.763±0.008	0.763±0.008
	3	0.837±0.008	0.837±0.008
	4	0.819±0.01	**0.86±0.012**
seismic-bumps	2	0.934±0	0.934±0
	3	0.934±0	**0±0**
	4	**0.936±0.003**	
tae	2	0.827±0.033	0.827±0.033
	3	0.887±0.061	0.887±0.054
	4	0.853±0.063	**0.86±0.04**
tic-tac-toe	2	0.698±0.03	**0.709±0.037**
	3	0.751±0.032	**0.753±0.035**
	4	0.753±0.024	**0.8±0.037**
titanic	2	0.811±0.026	0.811±0.026
	3	**0.813±0.027**	
	4		
wdbc	2	0.792±0.034	0.792±0.035
	3	**0.81±0.038**	
	4		
wine	2	**0.494±0.062**	0.483±0.06
	3	**0.611±0.071**	
	4		

features in the dataset, $J \geq 27$, then we deem Branch & Price to be of use (as seen in Figure 7.1).

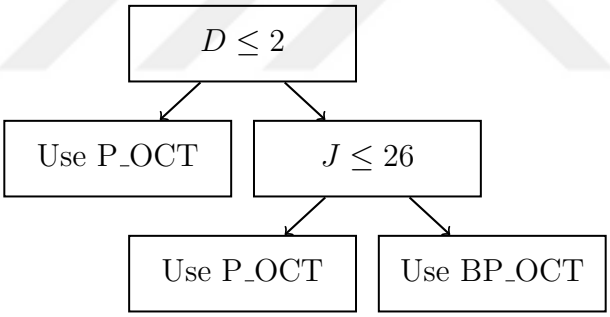

Figure 7.1: The tree we use in order to decide whether to apply Branch & Price to the data set instance with J features, to obtain a classification tree of maximum depth D

These conjoined rules comply with our initial intuition: as D and J grows, the number of patterns grows much more rapidly, inhibiting *P_OCT* from working properly, or simply driving the solver's memory to crash due to the large number of variables in the optimization problem.

Hence, we propose a hybrid approach: If the use case permits, we apply *BP_OCT*. If not, we use *P_OCT*. This combined approach is labeled as *Danztig-Wolfe Reformulation* and referred to as DWR from now on.

7.5 Compact Formulation vs. DWR

In Table 7.7, the compact formulation introduced is compared against the DWR results obtained in the previous part. On average, in about 64.6% of all data set- depth pairings, DWR provides better results compared to CompactOCT. Moreover, this success is mostly reflected on the test data set accuracies as well, as reported in Table 7.8.

Note that computational optimization techniques are very sensitive to hyper-parameters. Changing the number of patterns we initialize the Branch-and-Price tree with, or the types of patterns that are in the initial pattern set, could lead to drastic changes in computational time. We also note the success of DWR in certain data sets across all tried depths (see car evaluation, kr-vs-kp, nursery, and, seismic-bumps data sets in Table 7.7). Moreover, we can observe that DWR outperforms CompactOCT on all data sets, at least for one depth. Hence, DWR is shown to improve CompactOCT results and find better solutions under certain settings.

DWR (P_OCT and BP_OCT) seems to struggle the most with multi-class data sets (see balance-scale, car-evaluation, nursery, and wine in Table 7.7) and data sets with greater number of features (see data sets diabetes, wdbc, seismic-bumps in Table 7.7). Our limitations are suspected to arise from the combinatorial nature of tree patterns, that grow exponentially as the number of features grows and allowed tree depth grows. However, the approach seems to be less impacted by the growing number of data points, and find relatively better solutions compared to all other methods. Two such data sets are kr-vs-kp, nursery. Hence, DWR (P_OCT and BP_OCT) has the best computational edge, when working on binary classification datasets with large sample sizes and relatively less number of features as opposed to theier performance on other data instances. Thus, we conclude CompactOCT and DWR could work well together to complement each other when finding OCTs for depths up to 4.

Table 7.7: DWR Results Bench-marked Against CompactOCT

Dataset	Depth	DWR			CompactOCT		
		Time (s)	Training Accuracy	Opt Gap	Time (s)	Training Accuracy	Opt Gap
		mean ± std	*mean ± std*	*mean ± std*	*mean ± std*	*mean ± std*	*mean ± std*
adult	2	**14.7±7.7**	0.809±0	0±0	3601.3±1.7	0.771±0.009	100±0
	3	**5063.1±19.8**	0.82±0	100±0	3601.8±0.3	0.765±0.008	100±0
	4				**3606.7±7.4**	0.741±0.02	100±0
agaricus-lepiota	2	**8±0**	0.969±0	0±0	1349.4±337.5	0.969±0	0±0
	3	**3610.9±34.1**	0.985±0	100±0	3600.3±0.1	0.979±0.007	100±0
	4				**3348±715.6**	0.972±0.018	87.5±35.36
balance-scale	2	**0±0**	0.686±0.003	0±0	1.7±0.4	0.686±0.003	0±0
	3	71.7±16.9	0.745±0.003	0±0	**28.6±1.9**	0.745±0.001	0±0
	4				**2630.4±784.6**	0.787±0.001	1.94±3.05
banknote authentication	2	**0±0**	0.739±0.002	0±0	0.3±0	0.739±0.002	0±0
	3	27.2±5.2	0.816±0.003	0±0	**3.2±0.6**	0.816±0.003	0±0
	4				**18.6±3.2**	0.895±0.004	0±0
car-evaluation	2	**0±0**	0.739±0.002	0±0	16.3±2.8	0.777±0.002	0±0
	3	**27.2±5.2**	0.816±0.003	0±0	1108.1±454.1	0.814±0.001	0±0
	4	**3600.3±0**	0.839±0.004	100±0	3600±0	0.836±0.005	99.91±0.27
diabetes	2	**0±0**	0.777±0.002	0±0	5.1±1	0.749±0.004	0±0
	3	**162.8±24.2**	0.753±0.007	74.2±1.1	727.2±241.9	0.769±0.005	0±0
	4	3945±188.2	0.759±0.01	93.3±2.4	**3600±0**	0.783±0.007	50.02±12.59
haberman	2	**0.4±0**	0.749±0.004	0±0	0.5±0.1	0.761±0.006	0±0
	3	434.3±285.5	0.782±0.005	0±0	**18.4±5.7**	0.782±0.005	0±0
	4				**434.6±120.7**	0.807±0.004	0±0
kr-vs-kp	2	**0.1±0.0**	0.761±0.006	0	98.4±21.2	0.869±0.003	0±0
	3	**1986±136.6**	0.938±0.001	0±0	3518.9±256.4	0.935±0.007	78.11±31.36
	4	**3747.8±136.8**	0.94±0.001	100±0	3600.1±0	0.933±0.011	100±0
monks-1	2	**0±0**	0.78±0.003	0±0	0.7±0	0.78±0.003	0±0
	3	**4.3±0.6**	0.897±0.003	0±0	6.2±2	0.897±0.003	0±0
	4	8±2.4	1±0	0±0	**1.9±0.9**	1±0	0±0
monks-2	2	**0.1±0**	0.657±0	0±0	2.9±0.6	0.657±0	0±0
	3	**110.7±46.6**	0.677±0.005	0±0	1102.2±343.4	0.677±0.005	0±0
	4				**3600±0**	0.713±0.006	71.81±6.74
monks-3	2	**0±0**	0.963±0.001	0±0	0.1±0	0.963±0.001	0±0
	3	**1.6±0.3**	0.989±0.001	0±0	3.1±0.5	0.989±0.001	0±0
	4				**3196.3±866.6**	0.989±0.001	34.81±20.62
nursery	2	**0±0**	0.763±0.001	0±0	594.9±162.3	0.763±0	0±0
	3	**193.5±63.6**	0.837±0	0±0	3600.1±0	0.813±0.013	100±0
	4	**3600.7±0.3**	0.86±0.007	100±0	3600.4±0	0.79±0.017	100±0
seismic-bumps	2	**20.6±12.4**	0.935±0	0±0	337.1±82.2	0.935±0	0±0
	3	3600±0	0.935±0.001	100±0	**3600±0**	0.936±0	100±0
	4	**3760.9±17.4**	0.937±0	100±0	**3600.1±0**	0.936±0	100±0
tae	2	**0.2±0**	0.844±0.005	0±0	0.5±0.1	0.844±0.005	0±0
	3	674.5±76.8	0.89±0.006	0±0	**12.7±3.3**	0.89±0.006	0±0
	4	3686.3±45.9	0.888±0.017	100±0	**978.2±381.8**	0.937±0.006	0±0
tic-tac-toe	2	**0.5±0.1**	0.709±0.003	0	25.2±4.8	0.709±0.003	0±0
	3	**589.1±23.8**	0.781±0.003	0	3515.3±183.9	0.78±0.002	28.14±25.23
	4	3675.0±45.5	0.839±0.009	100	**3600±0**	0.846±0.008	100±0
titanic	2	**10.3±5.1**	0.811±0.003	0	13.5±5.3	0.811±0.003	0±0
	3	3600.3±0.1	0.823±0.008	100,	**1938.9±739.6**	0.836±0.003	0±0
	4				**3600.1±0**	0.844±0.004	100±0
wdbc	2	9.1±0.6	0.83±0.004	0	**0.9±0.2**	0.83±0.004	0±0
	3	3748.3±105.9	0.907±0.008	40.2±10.1	**40.4±15.5**	0.921±0.004	0±0
	4				**1905.1±552**	0.968±0.003	0±0
wine	2	**0.3±0.0**	0.62±0.006	0	0.2±0	0.62±0.006	0±0
	3	3636.5±21.7	0.715±0.006	1.3±1.2	**1.5±0.7**	0.716±0.007	0±0
	4				**11.7±6.7**	0.802±0.005	0±0

7.6 Note on Achieving Tighter Bounds

For most datasets, our formulations report rather high optimality gaps when a solution is not found in the specified time limit, especially as depth grows (see adult, agaricus-lepiota, diabetes, haberman, kr-vs-kp, monks-2, nursery, seismic-bumps, and tic-tac-toe data sets in Tables 7.5 and 7.7). We experimented with adding a set of valid equalities to our formulation, which tightens the LP relaxation bounds. As explained in Chapter 5, these cuts dictate that each data point must fall under exactly one pattern [24]. However, when we used these cuts in one fold of each of our tried data sets, we saw that it increased both construction time and optimization time by a noticeable amount, in a majority of the data sets (noticable exceptions are depth 3 runtimes of diabetes,car-evaluation.

Table 7.8: DWR Results Bench-marked Against Compact Formulation, In Terms of Test Accuracy

Dataset	Depth	Test Accuracy	
		BP_OCT	P_OCT
		mean ± std	*mean ± std*
adult	2	**0.809±0.003**	0.771±0.009
	3	**0.791±0.003**	0.766±0.009
	4		0.761±0.001
agaricus-lepiota	2	0.969±0.008	0.969±0.008
	3	0.948±0.009	**0.979±0.006**
	4		**0.973±0.018**
balance-scale	2	0.66±0.029	**0.669±0.016**
	3	0.705±0.029	0.705±0.015
	4		**0.724±0.029**
banknote-authentication	2	0.739±0.017	0.739±0.017
	3	**0.807±0.025**	0.804±0.024
	4	0.773±0.037	**0.89±0.035**
car-evaluation	2	0.739±0.017	**0.777±0.021**
	3	**0.805±0.025**	0.789±0.014
	4	**0.829±0.025**	0.822±0.022
diabetes	2	**0.777±0.021**	0.746±0.039
	3	**0.755±0.044**	0.757±0.038
	4	0.748±0.034	**0.77±0.028**
haberman	2	0.748±0.034	0.748±0.027
	3	0.748±0.023	**0.751±0.025**
	4		**0.775±0.042**
kr-vs-kp	2	0.869±0.026	0.869±0.026
	3	**0.938±0.014**	0.936±0.015
	4	0.809±0.027	**0.936±0.024**
monks-1	2	0.735±0.031	0.74±0.028
	3	0.866±0.029	0.863±0.026
	4	1±0	1±0

Dataset	Depth	Test Accuracy	
		BP_OCT	P_OCT
		mean ± std	*mean ± std*
monks-2	2	**0.668±0.017**	0.663±0.012
	3	**0.677±0.032**	0.668±0.022
	4		0.743±0.031
monks-3	2	0.964±0.016	0.965±0.013
	3	0.989±0.012	0.989±0.012
	4		**0.992±0.013**
nursery	2	0.763±0.008	0.763±0.008
	3	**0.837±0.008**	0.811±0.02
	4	**0.86±0.012**	0.789±0.021
seismic-bumps	2	0.934±0	0.934±0
	3	0.934±0	0.934±0.001
	4	**0.936±0.003**	0.934±0
tae	2	**0.827±0.033**	0.821±0.031
	3	0.887±0.061	0.887±0.054
	4	0.853±0.063	**0.919±0.044**
tic-tac-toe	2	**0.709±0.037**	0.707±0.04
	3	**0.751±0.032**	0.747±0.03
	4	0.753±0.024	**0.834±0.032**
titanic	2	0.811±0.026	0.811±0.026
	3	0.813±0.027	**0.837±0.025**
	4		**0.835±0.023**
wdbc	2	**0.792±0.035**	0.79±0.03
	3	0.81±0.038	**0.896±0.062**
	4		**0.949±0.034**
wine	2	0.483±0.06	0.483±0.06
	3	**0.611±0.071**	0.585±0.065
	4		**0.708±0.034**

haberman, tae, tic-tac-toe) as seen in Table 7.9. Moreover, we faced more computational crashes during optimization. Upon observing these preliminary results for depths 2 and 3, we did not move forward with the full implementation. Note that, for these runs, a more powerful computer was used (Macbook M1 Max), hence our ability to report runtimes on certain data set's IPs. .

Table 7.9: Effects of Cuts on Tried Datasets

Dataset	Depth	# of Patterns	**Runs Without Cuts**			**Runs With Cuts**		
			Cons. Time (s)	Opt Time (s)	Opt Gap (%)	Cons. Time (s)	Opt Time (s)	Opt Gap (%)
adult	2	52404	265.0	**3.1**	0	367.2	3613.6	inf
	3	7736344	-	-	-	-	-	-
agaricus-lepiota	2	37744	27.5	**0.9**	0	31.1	922.2	0
	3	4733344	2859.6	**3600.2**	100	-	-	-
balance-scale	2	1220	0.1	**0.0**	0	0.1	0.4	0
	3	28280	1.0	16.7	0	1.2	**13.1**	0
banknote-authentication	2	4840	0.3	**0.1**	0	0.4	1.3	0
	3	219760	14.2	**14.2**	0	17.2	97.3	0
car-evaluation	2	1344	0.1	**0.1**	0	0.1	1.9	0
	3	32648	2.5	65.7	0	1.8	**28.9**	0
diabetes	2	15624	1.1	**0.2**	0	1.3	25.5	0
	3	1264944	94.3	3600.5	52.8	122.1	**389.1**	0
haberman	2	1900	0.1	**0.1**	0	0.1	0.1	0
	3	54600	1.7	129.2	0	1.9	**40.7**	0
kr-vs-kp	2	4370	0.5	**0.1**	0	0.7	25.5	0
	3	188708	19.7	**305.4**	0	27.0	700.1	0
monks-1	2	690	0.0	**0.0**	0	0.0	0.2	0
	3	12160	0.4	**6.0**	0	0.5	8.1	0
monks-2	2	690	0.0	0.2	0	0.0	**0.1**	0
	3	12160	0.4	**40.5**	0	0.5	3.7	0
monks-3	2	690	0.0	**0.0**	0	0.0	0.1	0
	3	12160	0.4	**1.1**	0	0.5	2.3	0
nursery	2	2054	0.8	**0.1**	0	1.2	68.2	0
	3	61308	18.1	**50.9**	0	29.9	2329.1	0
seismic-bumps	2	15624	2.4	14.4	0	3.0	313.4	0
	3	1264944	173.5	3600.5	100	279.3	3603.4	100
tae	2	3104	0.1	**0.1**	0	0.1	0.2	0
	3	113344	3.7	204.7	0	4.0	**16.5**	0
tic-tac-toe	2	2214	0.1	**0.4**	0	0.2	1.4	0
	3	68544	3.2	477.9	0	4.0	**289.2**	0
titanic	2	95230	14.8	**11.7**	0	16.1	44.3	0
	3	18926028	-	-	-	-	-	-
wdbc	2	270300	70.8	**4.7**	0	73.9	332.3	0
	3	90360200	-	-	-	-	-	-
wine	2	50830	3.8	**0.2**	0	4.0	3.3	0
	3	7391020	-	-	-	-	-	-

7.7 Comparison of All Methods

Let us conclude with Figures 7.2 and 7.3, which provide an executive summary of our research. In Figure 7.2, on the top, the cases solved to optimality under a given time limit up to 3600 (s) are considered. At the bottom, percentage of all cases cases where tried methods time out but report an optimality gap under a given value are reported. For Figure 7.3, which represents win counts, instances which were solved to optimality are on the top, and the instances where time limit was reached are in the bottom. In the plot shown above, if two methods retrieve the optimal solution at the same time, then they both considered the winner and it is counted towards a tie. Similarly, on the bottom if time limit was reached with same best training accuracy for multiple methods, those instances are registered as ties.

By looking at Figure 7.2, we can easily see that all of our proposed methods perform better in terms of computational time, compared to BendersOCT, with CompactOCT appearing to be capable of solving more instances under a specified time, out of all our tried methods. This finding is verified by Figure 7.3 showing that when other methods time out, we have CompactOCT finding the best accuracy in most cases. Moreover, by looking at the top and bottom plots at once, we see that DWR comes in second, which can be observed in Figure 7.2. Also, again referring to Figure 7.3, we conclude by looking at the number of times each method has retrieved the optimal solution fastest (top) or the solution with higher training accuracy (bottom), that both our DWR (P_OCT & BP_OCT) and CompactOCT methods are the best performing methods for at *least one depth* both in both plots.

Moreover, by looking at Figure 7.2, we can easily see P_OCT and BP_OCT perform at least as well as BendersOCT in terms of solving a greater fraction of the instances to optimality. Furthermore, both P_OCT and BP_OCT are able to outperform CompactOCT and BendersOCT, reigning as the winners on some data sets for certain depths. These results demonstrate the potential of pattern-based tree models, and demonstrates promising results for the future use of Branch-and-Price to build OCTs.

As seen on Figure 7.2, when looking at the reported gaps for each method on the instances where optimal solutions could not be found within the specified time limit, we see that BendersOCT reports tighter optimality gaps for the solutions retrieved. Although CompactOCT and/or DWR finds better solutions in terms of training accuracy across all depths in general (more Win Counts in Figure 7.3), and also solves a greater fraction

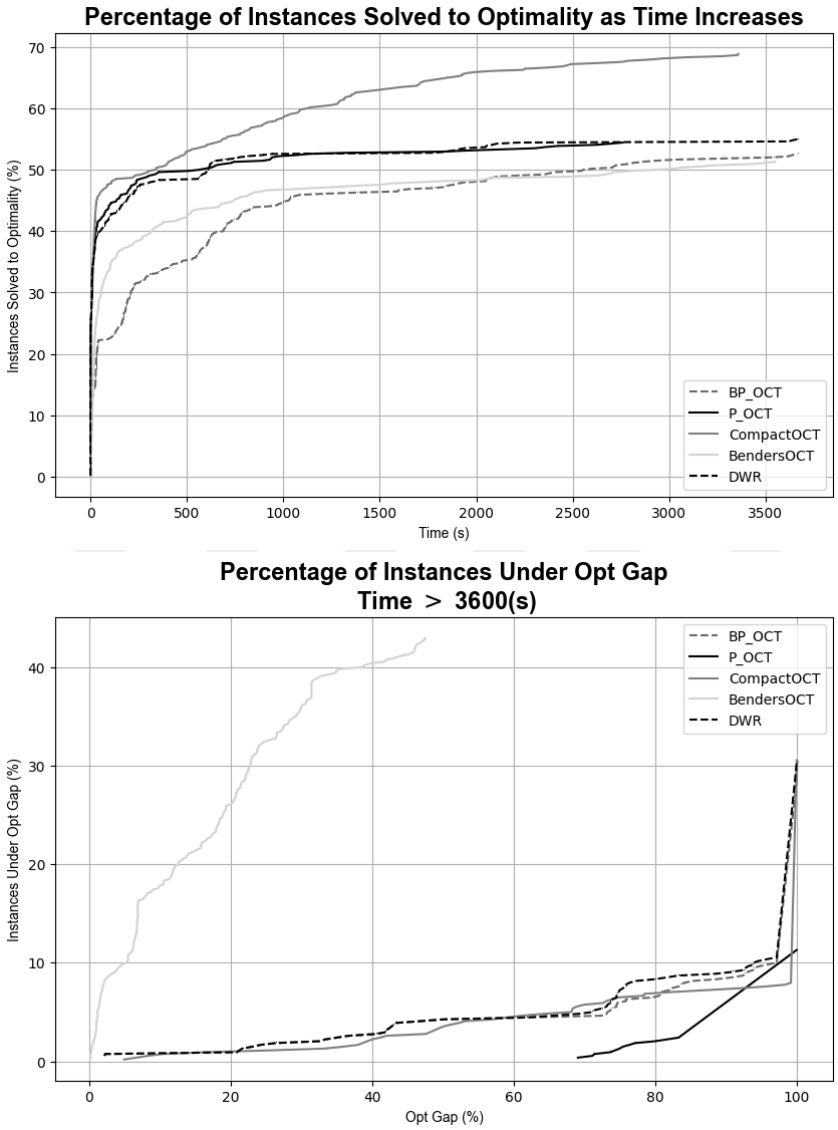

Figure 7.2: Performance Profile Plots.

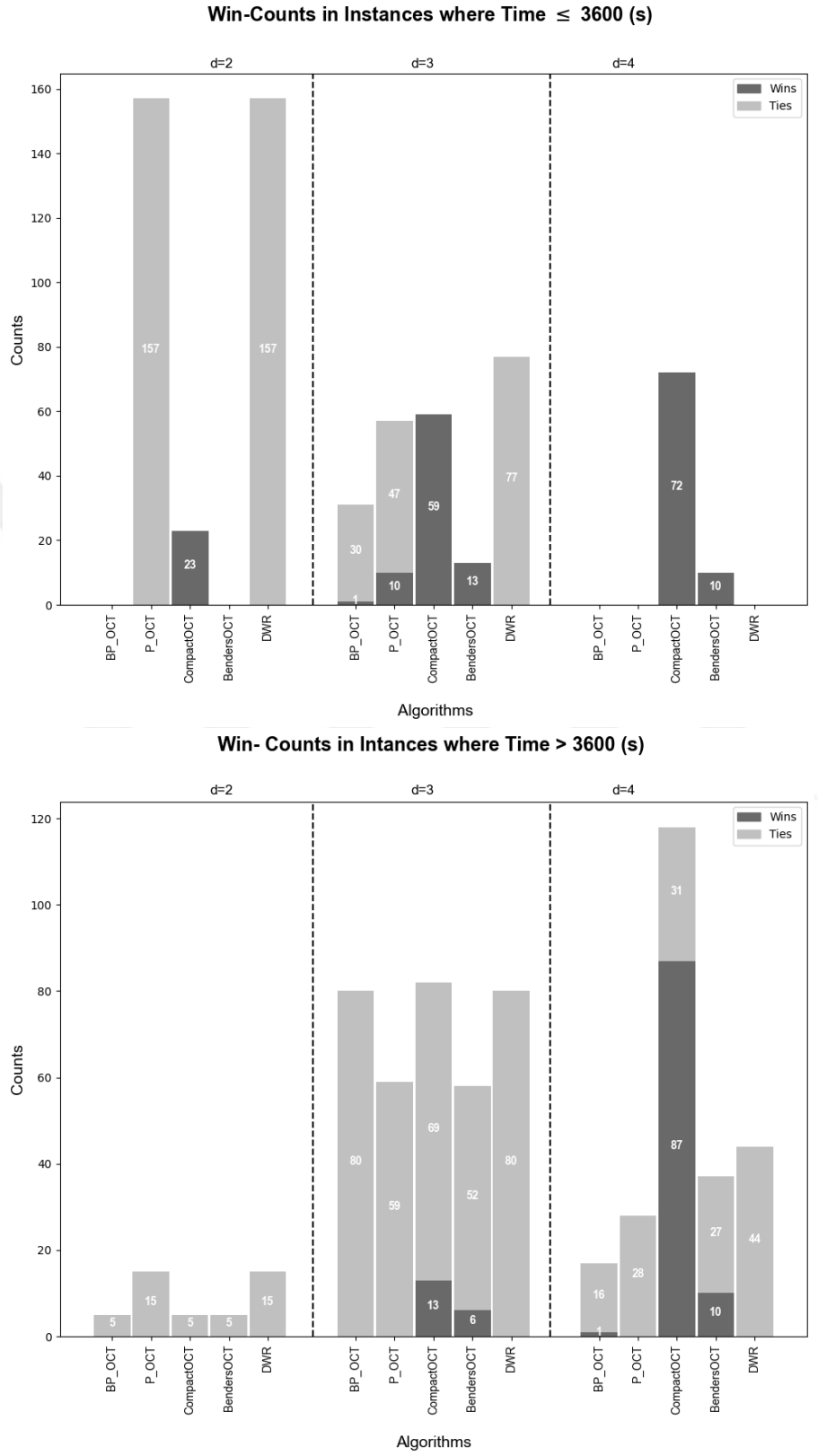

Figure 7.3: Win-Counts Tallied for Each Method tried.

of the instances to optimality (greater percentage of instances solved under given time displayed in Figure 7.2), BendersOCT seems to be more precise regarding optimality gaps. Even when the quality of the solutions retrieved are better when CompactOCT and/or DWR time out, BendersOCT provides a more accurate measure of the quality of the found solutions. As mentioned previously, we did experiment with another formulation which yields tighter bounds, as the LP relaxation of the IP provides an objective value very close to that of the IP. Nevertheless, the high construction and optimization times discouraged us from moving forward with the cuts.

7.8 Fairness

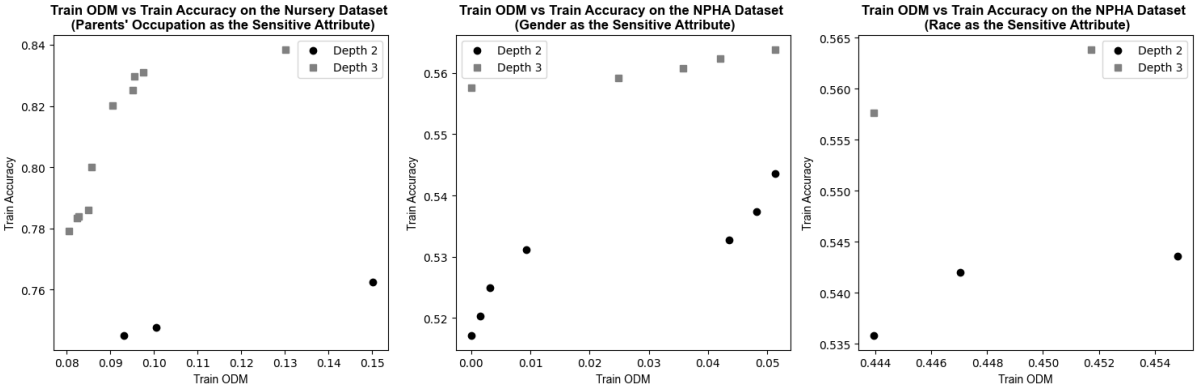

Figure 7.4: Training Accuracy and ODM pairs found $\epsilon-$ Constraint algorithm and their corresponding Test Accuracy and ODM Values, ordered by ascending ϵ values used to find the non-dominated points

For the fairness-aware models introduced previously in Cahpter 4, we ran an $\epsilon-$ Constraint algorithm. ϵ, used for constraining Train ODM, is initialized as 1, and updated until termination. The resulting Pareto-frontiers for P_OCT are provided in Figure 7.4. In the first and third plot (left to right), multiple binary features, make up the feature class, hence we constrained multiple binarized features at once. In the second plot, gender, given as a binary feature, is constrained. Detailed runtime, accuracy and ODM information for the $\epsilon-$ Constraint algorithm can be found in Table 7.10.

We note that in Figure 7.4, weakly dominated points were taken out in post-processing. Note that, post-processing of dominated points was also needed since in some iterations of the algorithm the optimal solution could not be obtained in the specified time limit of 3600 (s). Please also note that P_OCT was selected as the best method to use due to the number of features in the binarized datasets

We see that as we introduce fairness constraint to the model, there is a drop in accuracy. This trade-off was expected, however, we are able to generate more fair solutions, with slight losses in Accuracy as observed in Figure 7.4. For example, for a two-depth P_OCT model on the Nursery data set, by moving from the non-dominated point with the highest accuracy to the non-dominated point with the lowest ODM represents a 3% reduction in accuracy, and a 40% improvement in ODM. Similarly, on the NHPA data set, with gender as the sensitive attribute, we find three-depth models with 0 ODM, and $< 1\%$ deviation from the accuracy of the maximum accuracy non-dominated point. These results show that addition of ODM consraints to the model can help find more equitable solutions with similar accuracy values for senstive and non-senstive groups, while not sacrificing too much from accuracy.

As explained by our fairness section which shows potential use-cases on three data set-sensitive attribute instances, our methods can be extended to include fairness constraints easily. As shown briefly, we are easily able to find approximation of the pareto-frontiers to observe the trade-off between accuracy and fairness, represented by ODM.

Table 7.10: ϵ -Constraint results for Nursery and NPHA data sets. The first (top) table has Parents' Occupation as the sensitive attribute, the second (middle) uses gender, and the third (bottom) uses race.

Depth	ϵ	Time (s)		Training			Test	
		Cons	Opt	Acc	ODM	Opt Gap	Acc	ODM
2	1.000	2.7	0.1	0.762	0.150	0	0.772	0.156
	0.150	2.8	0.0	0.748	0.101	0	0.743	0.120
	0.100	2.8	0.1	0.745	0.093	0	0.748	0.098
3	1.000	63.9	99.5	0.838	0.309	0	0.830	0.301
	0.309	67.3	59.6	0.838	0.130	0	0.830	0.132
	0.130	66.7	99.3	0.831	0.098	0	0.829	0.107
	0.098	65.2	75.0	0.830	0.096	0	0.826	0.103
	0.095	65.1	159.7	0.825	0.095	0	0.825	0.103
	0.095	62.9	154.8	0.820	0.091	0	0.822	0.073
	0.091	64.6	146.7	0.800	0.086	0	0.797	0.096
	0.086	64.9	258.1	0.786	0.085	0	0.782	0.086
	0.085	66.4	431.6	0.784	0.083	0	0.774	0.095
	0.083	65.5	524.3	0.783	0.082	0	0.778	0.098
	0.082	65.5	416.0	0.779	0.081	0	0.774	0.098

Depth	ϵ	Time (s)		Training			Test	
		Cons	Opt	Acc	ODM	Opt Gap	Acc	ODM
2	1.000	0.6	1.5	0.544	0.051	0.000	0.569	0.069
	0.050	0.6	2.0	0.537	0.048	0.000	0.583	0.042
	0.047	0.6	1.8	0.533	0.045	0.000	0.569	0.097
	0.044	0.6	1.9	0.533	0.044	0.000	0.583	0.042
	0.042	0.6	2.0	0.531	0.042	0.000	0.569	0.014
	0.040	0.6	1.8	0.531	0.009	0.000	0.583	0.014
	0.008	0.6	1.4	0.525	0.003	0.000	0.583	0.028
	0.002	0.6	1.8	0.520	0.002	0.000	0.569	0.014
	0.000	0.6	2.2	0.517	0.000	0.000	0.583	0.056
3	1.000	25.7	3600.2	0.564	0.051	0.996	0.611	0.042
	0.050	25.7	3600.2	0.561	0.050	0.862	0.569	0.056
	0.048	25.6	3600.2	0.561	0.044	0.996	0.569	0.111
	0.042	26.1	3600.1	0.562	0.042	1.000	0.583	0.042
	0.040	25.8	3600.1	0.561	0.036	0.996	0.597	0.056
	0.034	25.8	3600.1	0.553	0.028	0.997	0.597	0.097
	0.023	26.2	3600.1	0.558	0.005	0.993	0.569	0.014
	0.026	26.0	3600.3	0.559	0.025	0.996	0.611	0.014
	0.003	26.1	3600.2	0.548	0.003	0.983	0.569	0.028
	0.002	26.1	3600.2	0.558	0.000	0.993	0.569	0.014

Depth	ϵ	Time (s)		Training			Test	
		Cons	Opt	Acc	ODM	Opt Gap	Acc	ODM
2	1.000	1.5	4.2	0.544	0.455	0.000	0.569	0.708
	0.453	1.5	3.4	0.542	0.447	0.000	0.569	0.597
	0.445	1.5	3.5	0.536	0.444	0.000	0.569	0.569
3	1.000	68.2	10800.6	0.551	0.497	0.955	0.569	0.500
	0.495	68.1	3600.9	0.564	0.453	0.989	0.611	0.694
	0.452	89.5	3600.3	0.564	0.452	0.996	0.611	0.931
	0.450	67.7	3600.3	0.558	0.444	0.996	0.583	0.722

Chapter 8

Conclusion and Future Work

In this thesis, the problem of building OCTs was considered extensively. We presented four methods to solve the problem: CompactOCT, P_OCT, BP_OCT, and our compound method, DWR. CompactOCT models frames the problem as choosing split-variables for the split-nodes in the tree to minimize the misclassification error which incurs when data points falling under leafs to not match the assigned class of their respective leaf (CompactOCT). We proceed to introduce a pattern-based formulation which views collection of features that are paths to the leaf nodes as patterns, and chooses as many patterns as the number of leafs to construct the tree (P_OCT). Building on this pattern-based formulation, we use the dual of the LP-relaxation of P_OCT, and develop a Branch-and-Price scheme with decomposable pricing problems (BP_OCT). We then demonstrate the computational feasibility of our approaches by comparing them to well known heuristic and optimization approaches, CART and BendersOCT.

As mentioned repeatedly in the literature and observed on our computational experiments, the biggest issue in using tree based models for OCTs arises from the requirement to include constraints for each data point in order to detect which leaf they fall under. This approach rapidly grows the model, making instances involving larger datasets virtually intractable. However, on an overwhelming majority of the data sets we experimented on, we found better solutions in lesser time with CompactOCT compared to BendersOCT, an algorithm coined as the best-performing optimal classification tree builder, even on highly populated data sets. Motivated by the CompactOCT formulation's success, we introduced a pattern-based model (P_OCT), amenable to column generation, and proposed a Branch&Price scheme (BP_OCT) which decomposes the flow constraints to be handled simultaneously and in parallel. In addition to making use of the fast and convenient

CART heuristic for an initial incumbent solution in P_OCT and BP_OCT, we also exploit its algorithmic logic to leverage our BP_OCT with an initial warm-up loop. Finally, we composed a compound algorithm, deciding when to solve the P_OCT formulation or when to apply BP_OCT for the problem depending on instance-specific properties such as number of features and desired maximum depth. This algorithm, labeled as DWR, was also able to contribute to the CompactOCT solution on average in most of the data set/ depth instances. We also present a possible addition of fairness metrics, as showcased by the use of ODM to create fairness aware models that are capable of finding solutions while adhering to fairness constraints.

In the future, our aim is to work on finding better lower bounds by tightening our formulation, and thus improve optimality gaps for CompactOCT and DWR on the instances where time limit is reached. However, as it stands, we believe we present an extensive study of a novel tree-based modeling approach for OCTs, and how its decomposable nature could be exploited with a novel Branch & Price algorithm.

Bibliography

[1] A. Bell, I. Solano-Kamaiko, O. Nov, and J. Stoyanovich, "It's just not that simple: An empirical study of the accuracy-explainability trade-off in machine learning for public policy," *2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022.

[2] H. Kaur, H. Nori, S. Jenkins, R. Caruana, H. Wallach, and J. Wortman Vaughan, "Interpreting interpretability: Understanding data scientists' use of interpretability tools for machine learning," *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020.

[3] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare," *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone *Classification and regression trees*, 2017.

[5] J. Quinlan *Machine Learning*, vol. 1, no. 1, p. 81–106, 1986.

[6] J. R. Quinlan, *C4.5*. Morgan Kaufmann, 1993.

[7] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, p. 15–17, 1976.

[8] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, p. 1039–1082, 2017.

[9] O. Günlük, J. Kalagnanam, M. Li, M. Menickelly, and K. Scheinberg, "Optimal decision trees for categorical data via integer programming," *Journal of Global Optimization*, vol. 81, no. 1, p. 233–260, 2021.

[10] S. Aghaei, M. J. Azizi, and P. Vayanos, "Learning optimal and fair decision trees for non-discriminative decision-making," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, p. 1418–1426, 2019.

[11] S. Verwer and Y. Zhang, "Learning optimal classification trees using a binary linear program formulation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, p. 1625–1632, 2019.

[12] S. Aghaei, A. Gómez, and P. Vayanos, "Strong optimal classification trees," *CoRR*, vol. abs/2103.15965, 2021.

[13] P. Vossler, S. Aghaei, N. Justin, N. Jo, A. Gómez, and P. Vayanos, "Odtlearn: A package for learning optimal decision trees for prediction and prescription," Nov 2023.

[14] M. Firat, G. Crognier, A. F. Gabor, C. Hurkens, and Y. Zhang, "Column generation based heuristic for learning classification trees," *Computers Operations Research*, vol. 116, p. 104866, 2020.

[15] T. E. Röber, A. C. Lumadjeng, M. H. Akyüz, and İlker Birbil, "Rule generation for classification: Scalability, interpretability, and fairness," 2024.

[16] E. Carrizosa, C. Molero-Río, and D. Romero Morales, "Mathematical optimization in classification and regression trees," *TOP*, vol. 29, p. 5–33, Mar 2021.

[17] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, 02 1970.

[18] F. Vanderbeck, "Branching in branch-and-price: A generic scheme," *Mathematical Programming*, vol. 130, p. 249–294, Jan 2010.

[19] C. Barnhart, C. A. Hane, and P. H. Vance, "Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems," *Operations Research*, vol. 48, p. 318–326, Apr 2000.

[20] J. Valério de Carvalho *Annals of Operations Research*, vol. 86, p. 629–659, 1999.

[21] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud, "Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 249, no. 2, pp. 551–559, 2016.

[22] A. Sarac, R. Batta, and C. M. Rump, "A branch-and-price approach for operational aircraft maintenance routing," *European Journal of Operational Research*, vol. 175, no. 3, pp. 1850–1869, 2006.

[23] W. Ni, J. Shu, M. Song, D. Xu, and K. Zhang, "A branch-and-price algorithm for facility location with general facility cost functions," *INFORMS Journal on Computing*, vol. 33, no. 1, pp. 86–104, 2021.

[24] K. K. Patel, G. Desaulniers, and A. Lodi, "An improved column-generation-based matheuristic for learning classification trees," *Computers amp; Operations Research*, vol. 165, p. 106579, May 2024.

[25] Gurobi, Jun 2024.

[26] Z. B. Organ, E. Kayış, and T. Khaniyev, "Rolling lookahead learning for optimal classification trees," *arXiv.org*, Apr 2023.

[27] M. Kelly, K. Nottingham, and R. Longjohn, "The uci machine learning repository."

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix A

Elementary Pattern Formulation Runtimes for Two-Depth and Three-Depth Models of Selected Data Sets

Table A.1: Detailed Performance of *P_OCT* and *BP_OCT* Derived Using Elementary Formulation on Tested Datasets for $D = 2$

Dataset	**Metric**	**P_OCT** *mean ± stdev*	**BP_OCT** *mean ± stdev*
balance-scale	Time (s)	**0.053 ± 0.006**	19.768 ± 1.053
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.687 ± 0.004**	0.687 ± 0.004
	Test Accuracy	**0.661 ± 0.034**	0.661 ± 0.034
banknote-authentication	Time (s)	**0.030 ± 0.002**	11.733 ± 3.784
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.739 ± 0.002**	0.739 ± 0.002
	Test Accuracy	**0.739 ± 0.018**	0.739 ± 0.018
car-evaluation	Time (s)	**0.053 ± 0.010**	54.099 ± 4.227
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.778 ± 0.002**	0.778 ± 0.002
	Test Accuracy	**0.778 ± 0.022**	0.778 ± 0.022
kr-vs-kp	Time (s)	**0.217 ± 0.207**	162.478 ± 17.076
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.869 ± 0.002**	0.869 ± 0.002
	Test Accuracy	**0.869 ± 0.027**	0.869 ± 0.027
monks-1	Time (s)	**0.014 ± 0.007**	5.387 ± 1.952
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.780 ± 0.005**	0.780 ± 0.005
	Test Accuracy	**0.666 ± 0.230**	1.233 ± 1.564
monks-2	Time (s)	**0.444 ± 0.022**	8.299 ± 0.767
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.657 ± 0.001**	0.657 ± 0.001
	Test Accuracy	**0.662 ± 0.011**	0.662 ± 0.011
monks-3	Time (s)	**0.022 ± 0.000**	1.742 ± 0.157
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.964 ± 0.002**	0.964 ± 0.002
	Test Accuracy	**0.964 ± 0.017**	0.964 ± 0.017
tic-tac-toe	Time (s)	**1.444 ± 0.275**	53.261 ± 2.287
	Opt Gap	**0**	0.000 ± 0.000
	Training Accuracy	**0.709 ± 0.004**	0.708 ± 0.003
	Test Accuracy	**0.698 ± 0.032**	0.698 ± 0.032

Table A.2: Detailed Performance of *P OCT* and *BP OCT* Derived Using Elementary Formulation on Tested Datasets for $D = 3$

Dataset	**Metric**	**P_OCT** *mean ± stdev*	**BP_OCT** *mean ± stdev*
balance-scale	Time (s)	**111.204 ± 47.903**	207.958 ± 17.392
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.745 ± 0.004**	0.745 ± 0.004
	Test Accuracy	**0.715 ± 0.035**	0.715 ± 0.035
banknote-authentication	Time (s)	**10.236 ± 4.100**	998.180 ± 113.961
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.817 ± 0.003**	0.817 ± 0.003
	Test Accuracy	**0.807 ± 0.025**	0.807 ± 0.025
car-evaluation	Time (s)	350.835 ± 24.855	**196.076 ± 12.157**
	Opt Gap	0.000 ± 0.000	**0.000 ± 0.000**
	Training Accuracy	0.815 ± 0.002	**0.815 ± 0.002**
	Test Accuracy	0.791 ± 0.014	**0.791 ± 0.014**
kr-vs-kp	Time (s)	**896.215 ± 169.417**	2776.658 ± 509.111
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.938 ± 0.002**	0.938 ± 0.002
	Test Accuracy	**0.935 ± 0.014**	0.935 ± 0.014
monks-1	Time (s)	**5.526 ± 0.317**	36.262 ± 1.741
	Opt Gap	**0.078 ± 0.246**	0.078 ± 0.246
	Training Accuracy	**0.808 ± 0.282**	0.898 ± 0.003
	Test Accuracy	**0.867 ± 0.028**	0.867 ± 0.028
monks-2	Time (s)	204.527 ± 147.779	**64.781 ± 3.539**
	Opt Gap	0.000 ± 0.000	**0.000 ± 0.000**
	Training Accuracy	0.678 ± 0.005	**0.678 ± 0.005**
	Test Accuracy	0.677 ± 0.026	**0.677 ± 0.026**
monks-3	Time (s)	**3.105 ± 0.760**	36.564 ± 2.334
	Opt Gap	**0.000 ± 0.000**	0.000 ± 0.000
	Training Accuracy	**0.989 ± 0.001**	0.989 ± 0.001
	Test Accuracy	**0.989 ± 0.013**	0.989 ± 0.013
tic-tac-toe	Time (s)	3601.032 ± 0.858	**478.080 ± 25.789**
	Opt Gap	0.614 ± 0.053	**0.000 ± 0.000**
	Training Accuracy	0.782 ± 0.003	**0.782 ± 0.003**
	Test Accuracy	0.756 ± 0.029	**0.756 ± 0.029**

Appendix B

Root Node Results for BP_OCT

Root node incumbent solution values can be found in Table B.1. When we look at the values, we see that we are able to improve the CART solution, at the root node, in around 16% of all tried instances, on average. This analysis convinces us that there is value in branching, as root node incumbents have very reported high optimality gaps in an overwhelming majority of the tried instances.

One other point to pay attention to here is that, sometimes, BP_OCT spends all of our time limit at the root node, especially depth grows. To prohibit this from happening, we tried to see what would happen if we would prioritize exploring nodes, instead of improving our bounds. Referring back to our formulations shown in Chapter 3, we can see that at any given time, the sum of dual variables $\sum_{l \in L} \theta_l$ provides a lower bound to the master problem. Hence, we could apply column generation for a few iterations, stop, branch the node, and continue from child nodes. However, this method yields weak lower bounds, and grows the tree size very rapidly due to a lack of pruning. Even when we wait until convergence, the root node optimality gaps are quite high, thus, the futility of this approach was expected.

Table B.1: Performance Metrics Reported for the Root Node Incumbent Solution of BP_OCT

Dataset	Depth	Root			
		Time (s)	**Training Accuracy**	**Opt Gap**	**Test Accuracy**
		mean ±std	*mean ±std*	*mean ±std*	*mean ±std*
adult	2	337.3±26.8	0.791±0	100±0	0.791±0.003
	3	2234±17.7	0.821±0	100±0	0.791 ±0.004
	4				
agaricus-lepiota	2	24.5±0.5	0.954±0	100±0	0.945±0.009
	3	322.1±50.7	0.985±0	100±0	0.948±0.009
	4				
balance-scale	2	0.6±0.2	0.684±0.005	40±20	0.672±0.044
	3	12.3±2.3	0.699±0.005	80±0	0.697±0.038
	4				
banknote-authentication	2	1.4±0.3	0.724±0.002	0±0	0.709±0.026
	3	51.1±8.9	0.794±0.01	50±30	0.746±0.039
	4	1130.3±298.6	0.849±0.022	30±10	0.773±0.037
car-evaluation	2	1.4±0.3	0.724±0.002	0±0	0.778±0.026
	3	51.1±8.9	0.807±0.01	50±30	0.746±0.039
	4	65.9±8.1	0.817±0.006	100±0	0.784±0.025
diabetes	2	0.7±0.1	0.777±0.002	100±0	0.777±0.021
	3	3603.6±10.4	0.763±0.004	50±0	0.743±0.041
	4	3945±188.2	0.759±0.01	90±0	0.748±0.034
haberman	2	2.6±0.5	0.746±0.004	80±0	0.746±0.038
	3	6.3±2.1	0.75±0.01	60±0	0.764±0.043
	4	82.1±34.1	0.778±0.006	80±0	0.82±0.061
kr-vs-kp	2	11.1±2.2	0.776±0.005	80±0	0.767±0.031
	3	18.1±0.1	0.904±0.002	100±0	0.767±0.031
	4	138.1±0.7	0.941±0.001	100±0	0.809±0.027
monks-1	2	0.2±0	0.746±0.004	50±0	0.746±0.04
	3	2±0.2	0.849±0.016	100±0	0.827±0.05
	4				
monks-2	2	0.2±0	0.657±0	70±0	0.662±0.015
	3	2.4±0.3	0.665±0.003	100±0	0.673±0.024
	4				
monks-3	2	0.1±0	0.964±0.001	60±0	0.964±0.016
	3	1.7±0.2	0.989±0.001	100±0	0.978±0.014
	4				
nursery	2	18±4	0.764±0.001	100±0	0.763±0.008
	3	113.6±9.4	0.825±0	100±0	0.763±0.007
	4	2245.6±575.3	0.853±0.001	100±0	0.819±0.01
seismic-bumps	2	3.5±0	0.934±0	100±0	0.934±0
	3	64.2±0.1	0.935±0	100±0	0.934±0.001
	4	651.3±131.6	0.938±0	100±0	0.936±0.003
tae	2	0.6±0	0.835±0.008	90±0	0.827±0.033
	3	5.2±0.2	0.865±0.017	100±0	0.834±0.046
	4				
tic-tac-toe	2	0.6±0.1	0.707±0.005	80±0	0.705±0.042
	3	5.8±0	0.755±0.003	100±0	0.734±0.037
	4				
titanic	2	14±4.7	0.79±0.003	90±0	0.786±0.032
	3	381.9±148.2	0.823±0.008	100±0	0.813±0.027
	4				
wdbc	2	615.3±142.6	0.83±0.004	0±0	0.798±0.036
	3	3748.3±105.8	0.909±0.008	40±10	0.81±0.038
	4				
wine	2	3.1±0.9	0.607±0.015	0±0	0.546±0.083
	3	828.9±119.3	0.699±0.019	0±0	0.523±0.07
	4				

Appendix C

Example Branch-and-Price Tree

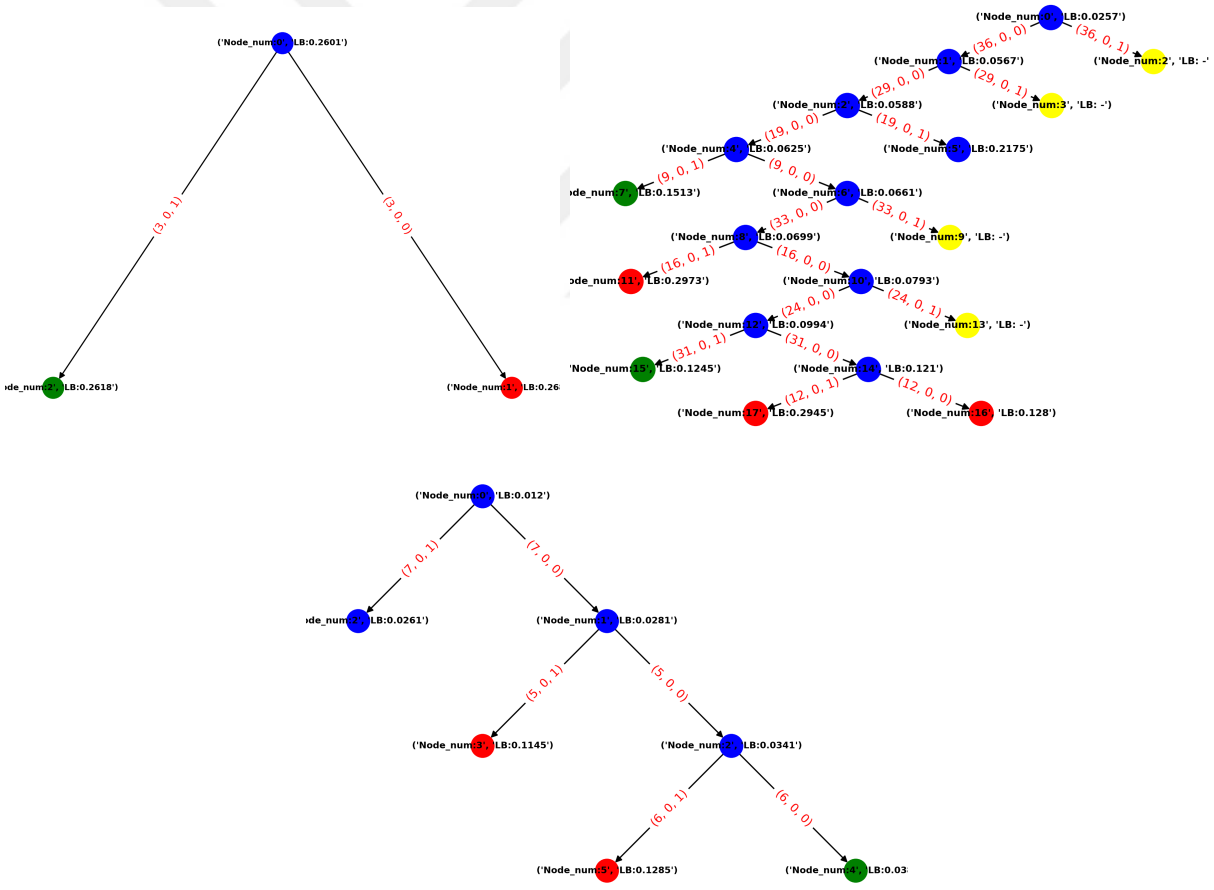

Figure C.1: Example Depth-2 BP_OCT Trees shown for data sets Banknote-Authentication, Kr-vs-Kp, and Monks-3.

Appendix D

Heuristic and Column Generation Iterations at Each Node

Please see Figure D.1, which displays number of iterations needed to column generation (CG) loops at each node, partitioned by whether the iterations are Heuristic Pattern Generations or Column Generation iterations. Percentage of the time spent in Heuristic Pattern Generation and Column Generation is displayed on each bar. As it can be seen, the Pattern Generation Heuristic defined in Chapter 6 helps decrease the number of CG iterations. Since our pricing problem used in CG is an IP with $3 + N$ linear constraints and $N + K + J$ binary constraints, with N, J, and K being quantified by the data set properties, it is prone to having high solution times. Thus, requiring lesser iterations for the convergence of the CG loop, is desired.

Moreover, the BP_OCT trees when heuristic is not used can also be found in Figure D.2. Green represents integral nodes, red represents pruned nodes, and yellow represents infeasible nodes. Branching variables are shown on the arcs between nodes in the form of, feature, split-node number, and fixed value. When Figure D.3 is compared to Figure C.1 and D.1, the trees tend to grow larger when heuristic is not used, hence the number of nodes in the tree increase. Also, let us reassert our point by mentioning the runtimes. On these three data set instances, algorithm runtimes are respectively 21.4 (s), 826.5 (s), and 4.7 (s). By referring to BP_OCT runtimes displayed on Table 7.5, we see that these runtimes are immensely greater than the average runtimes achieved when the heuristic is used (5 (s), 122 (s), and 0.6 (s)).

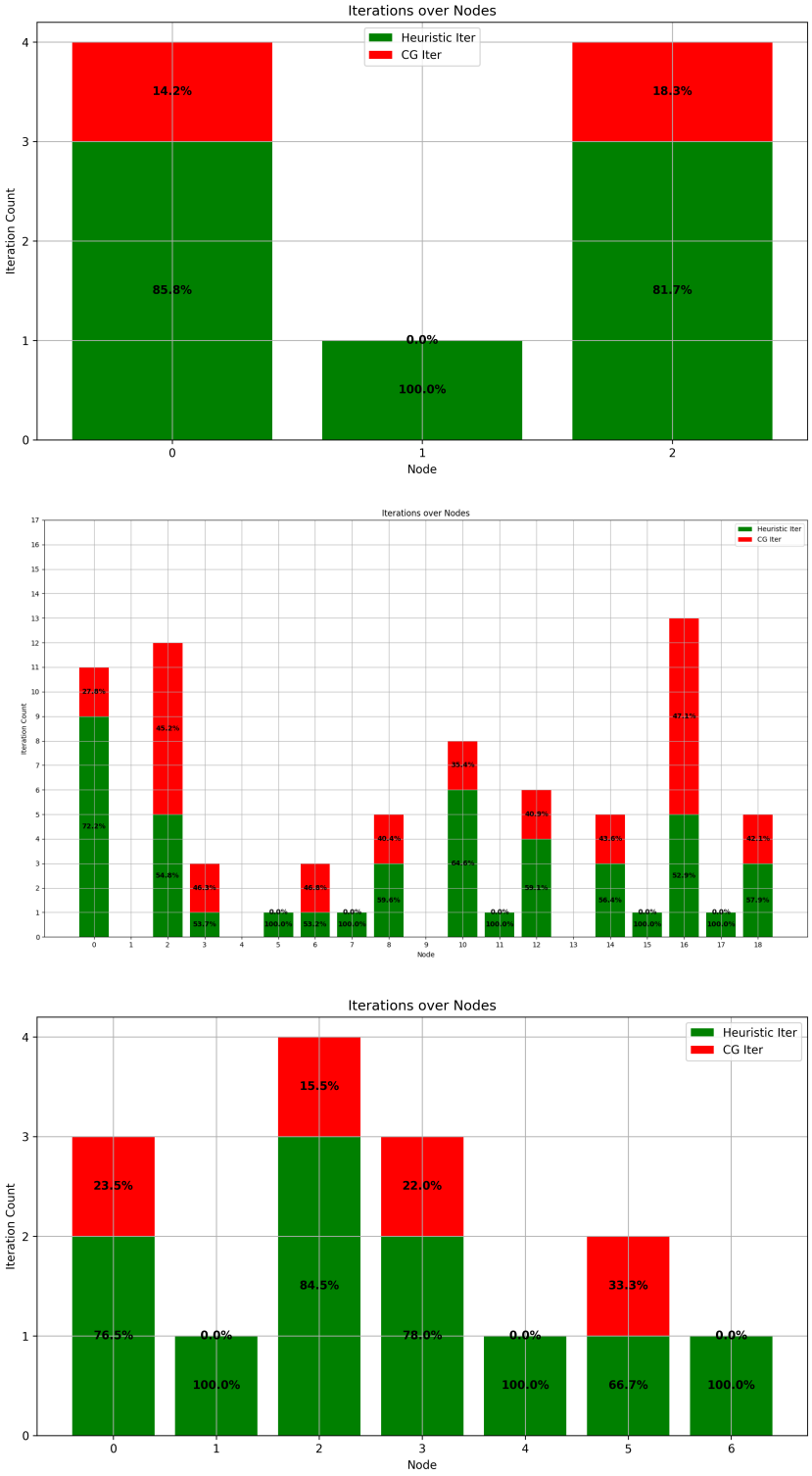

Figure D.1: Plots Showing the Number of Column Generation (Red) and Heuristic Pattern Generation (Green) Iteration Counts for Each Node Generated with the Warm-up Heuristic Pattern Generation Loop When Using Two-Depth BP_OCT, on Data Set Intances Banknote-Authentication, Kr-vs-Kp, and Monks-3.

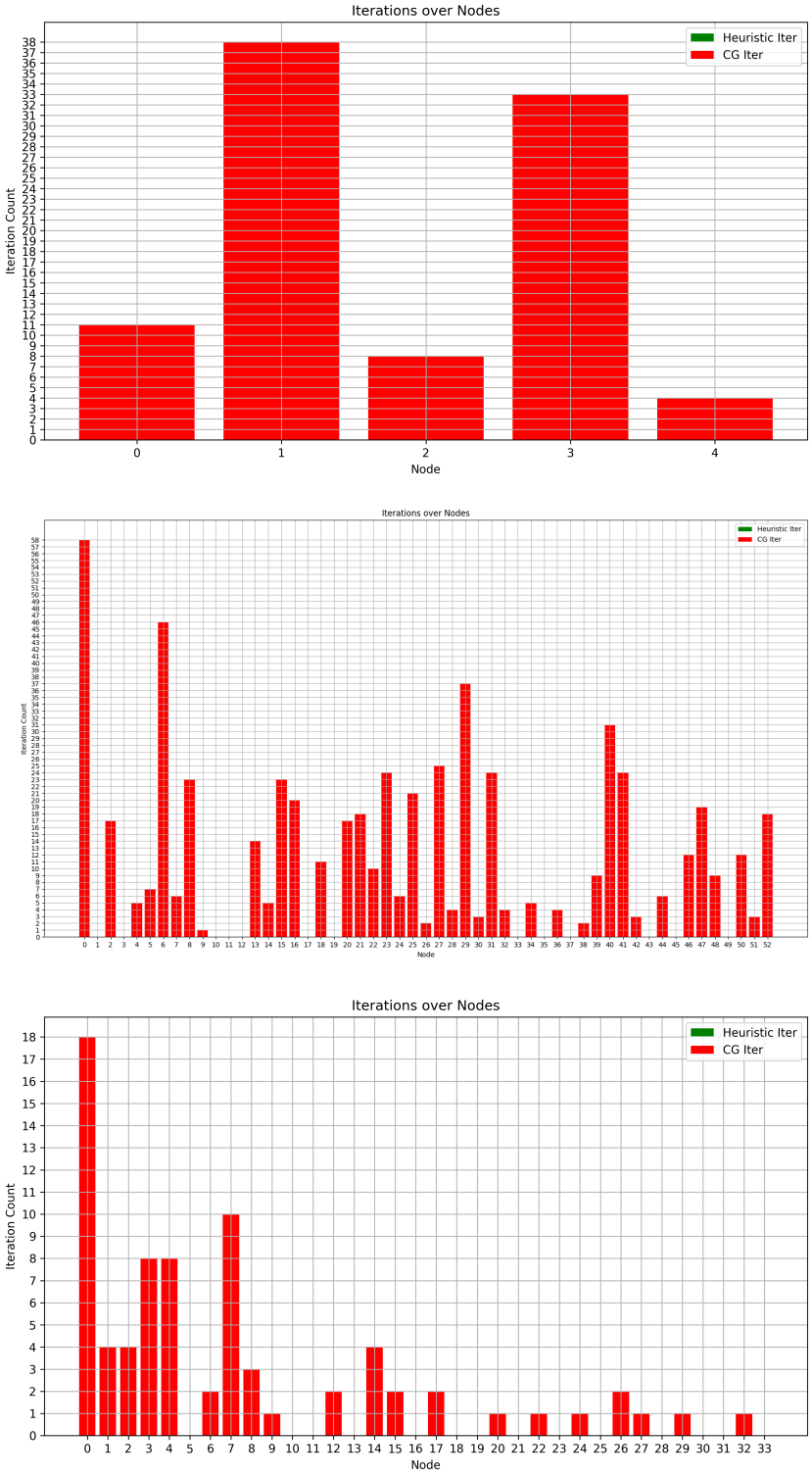

Figure D.2: Plots Showing the Number of Column Generation (CG) Iteration Counts for Each Node Generated When Using Two-Depth BP_OCT without the Warm-up Heuristic Pattern Generation Loop, on Example Data Set Intances Banknote-Authentication, Kr-vs-Kp, and Monks-3.

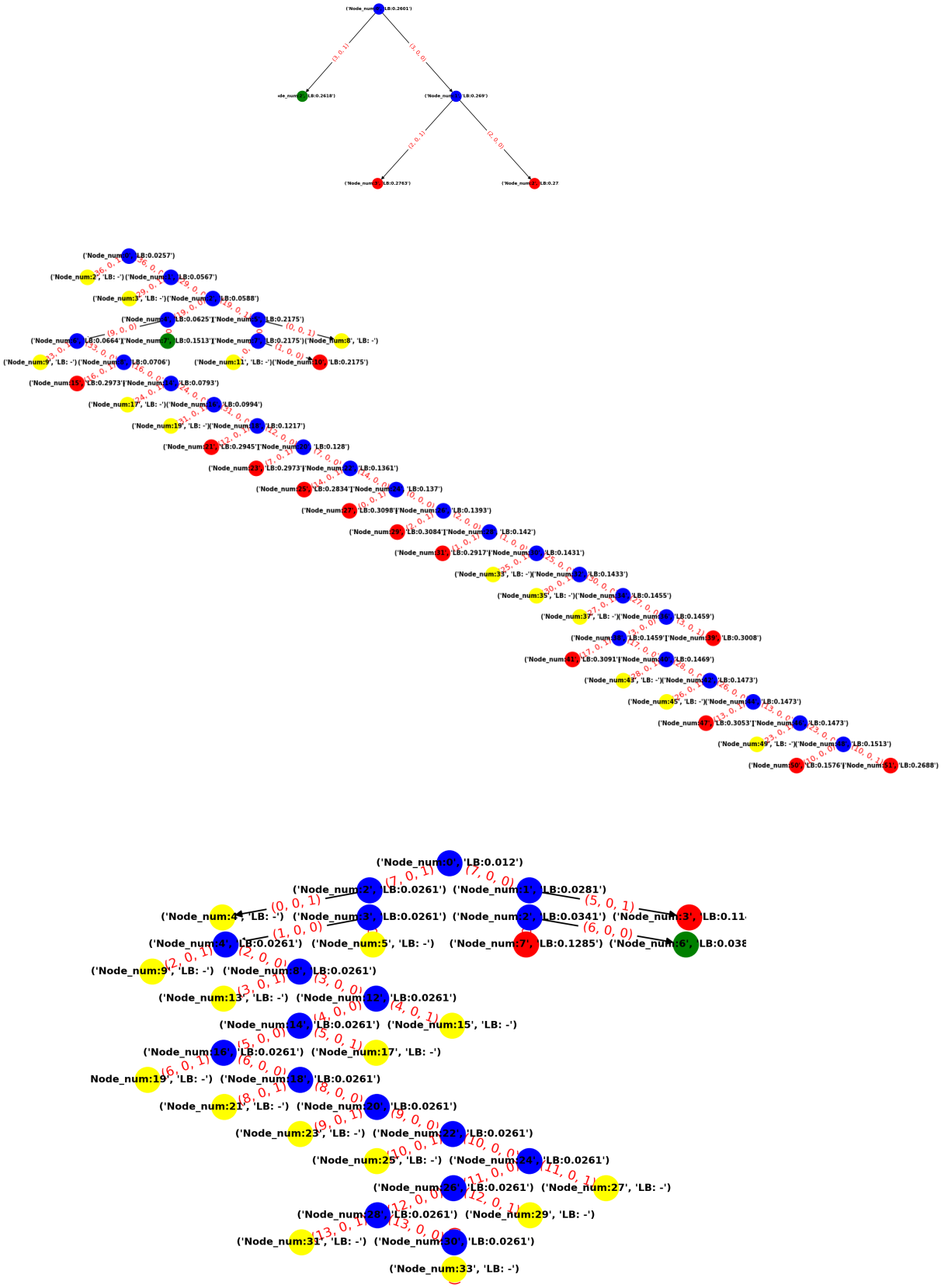

Figure D.3: Example Depth-2 BP_OCT Trees shown for data sets Banknote-Authentication, Kr-vs-Kp, and Monks-3, when When Using BP_OCT without the Warm-up Heuristic Pattern Generation Loop.