Learning Optimal Classification Trees: Strong Max-Flow Formulations

Sina Aghaei[1], Andrés Gómez[2], Phebe Vayanos[1]

[2]Department of Industrial and Systems Engineering, Viterbi School of Engineering

[1]Center for Artificial Intelligence in Society

[1,2]University of Southern California

{saghaei,gomezand,phebe.vayanos}@usc.edu

Abstract

We consider the problem of learning optimal binary classification trees. Literature on the topic has burgeoned in recent years, motivated both by the empirical suboptimality of heuristic approaches and the tremendous improvements in mixed-integer programming (MIP) technology. Yet, existing approaches from the literature do not leverage the power of MIP to its full extent. Indeed, they rely on *weak* formulations, resulting in slow convergence and large optimality gaps. To fill this gap in the literature, we propose a flow-based MIP formulation for optimal binary classification trees that has a *stronger* linear programming relaxation. Our formulation presents an attractive decomposable structure. We exploit this structure and max-flow/min-cut duality to derive a Benders' decomposition method, which scales to larger instances. We conduct extensive computational experiments on standard benchmark datasets on which we show that our proposed approaches are 50 times faster than state-of-the art MIP-based techniques and improve out of sample performance up to 13.8%.

1. Introduction

1.1. Motivation & Related Work

Since their inception over 30 years ago, decision trees have become among the most popular techniques for interpretable machine learning (classification and regression), see Breiman (1984). A decision tree takes the form of a *binary tree*. In each *internal* node of the tree, a binary test is performed on a specific feature. Two branches emanate from each internal node, with each branch representing the outcome of the test. If a datapoint passes (resp. fails) the test, it is directed to the left (resp. right) branch. A predicted label is assigned to all *leaf* nodes. Thus, each path from root to leaf represents a classification rule that assigns a unique label to all datapoints that reach

that leaf. The goal in the design of optimal decision trees is to select the tests to perform at each internal node and the labels to assign to each leaf to maximize prediction accuracy (classification) or to minimize prediction error (regression). Not only are decision trees popular in their own right; they also form the backbone for more sophisticated machine learning models. For example, they are the building blocks for random forests, one of the most popular and stable machine learning techniques available, see e.g., Liaw and Wiener (2002). They have also proved useful to provide explanations for the solutions to optimization problems, see e.g., Bertsimas and Stellato (2018).

The problem of learning optimal decision trees is an $\mathcal{NP}$-hard problem, see Hyafil and Rivest (1976) and Breiman (2017). It can intuitively be viewed as a combinatorial optimization problem with an exponential number of decision variables: at each internal node of the tree, one can select what feature to branch on (and potentially the level of that feature), guiding each datapoint to the left or right using logical constraints.

Traditional Methods. Motivated by these hardness results, traditional algorithms for learning decision trees have relied on heuristics that employ very intuitive, yet ad-hoc, rules for constructing the decision trees. For example, CART uses the Gini Index to decide on the splitting, see Breiman (1984); ID3 employs entropy, see Quinlan (1986); and C4.5 leverages normalized information gain, see Quinlan (2014). The high quality and speed of these algorithms combined with the availability of software packages in many popular languages such as `R` or `Python` has facilitated their popularization, see e.g., Kuhn et al. (2018), Therneau et al. (2015). They are now routinely used in commercial, medical, and other applications.

Mathematical Programming Techniques. Motivated by the heuristic nature of traditional approaches, which provide no guarantees on the quality of the learned tree, several researchers have proposed algorithms for learning provably *optimal* trees based on techniques from mathematical optimization. Approaches for learning optimal decision-trees rely on enumeration coupled with rules to prune-out the search space. For example, Nijssen and Fromont (2010) use itemset mining algorithms and Narodytska et al. (2018) use satisfiability (SAT) solvers. Verhaeghe et al. (2019) propose a more elaborate implementation combining several ideas from the literature, including branch-and-bound, itemset mining techniques and caching. Hu et al. (2019) use analytical bounds (to aggressively prune-out the search space) combined with a tailored bit-vector based implementation.

The Special Case of MIP. As an alternative approach to conducting the search for optimal trees, Bertsimas and Dunn (2017) recently proposed to use mixed-integer programming (MIP) to learn optimal classification trees. Following this work, using MIP to learn decision trees gained a lot of traction in the literature with the works of Günlük et al. (2018), Aghaei et al. (2019), and Verwer and Zhang (2019). This is no coincidence. First, MIP comes with a suit of off-the shelf solvers and

algorithms that can be leveraged to effectively prune-out the search space. Indeed, solvers such as CPLEX (2009) and Gurobi Optimization (2015) have benefited from decades of research, see Bixby (2012), and have been very successful at solving a broad class of MIP problems. Second, MIP comes with a highly expressive language that can be used to tailor the objective function of the problem or to augment the learning problem with constraints of practical interest. For example, Aghaei et al. (2019) leverage the power of MIP to incorporate fairness and interpretability constraints into learned classification and regression trees. They also show how MIP technology can be exploited to learn decision trees with more sophisticated structure (linear branching and leafing rules). Similarly, Günlük et al. (2018) use MIP to solve classification trees with combinatorial branching decisions. MIP formulations have also been leveraged to design decision trees for decision- and policy-making problems, see Azizi et al. (2018) and Ciocan and Mišić (2018), and for optimizing decisions over tree ensembles, see Mišic (2017).

Discussion & Motivation. The works of Bertsimas and Dunn (2017), Günlük et al. (2018), Aghaei et al. (2019), and Verwer and Zhang (2019) have served to showcase the modeling power of using MIP to learn decision trees and the potential suboptimality of traditional algorithms. Yet, we argue that they have not leveraged the power of MIP to its full extent. A critical component for efficiently solving MIPs is to pose good formulations, but determining such formulations is no simple task. The standard approach for solving MIP problems is the branch-and-bound method, which partitions the search space recursively and solves Linear Programming (LP) relaxations for each partition to produce lower bounds for fathoming sections of the search space. Thus, since solving a MIP requires solving a large sequence of LPs, small and compact formulations are desirable as they enable the LP relaxation to be solved faster. Moreover, formulations with tight LP relaxations, referred to as *strong* formulations, are also desirable, as they produce higher quality lower bounds which lead to a faster pruning of the search space, ultimately reducing the number of LPs to be solved. Unfortunately, these two goals are at odds with one another, with stronger relaxations often requiring additional variables and constraints than *weak* ones. For example, in the context of decision trees, Verwer and Zhang (2019) propose a MIP formulation with significantly fewer variables and constraints than the formulation of Bertsimas and Dunn (2017), but in the process weaken the LP relaxation. As a consequence, neither method consistently dominates the other.

We note that in the case of MIPs with large numbers of decision variables and constraints, classical decomposition techniques from the Operations Research literature may be leveraged to break the problem up into multiple tractable subproblems of benign complexity. A notable example of a decomposition algorithm is Benders' (Benders 1962). Bender's decomposition exploits the structure of mathematical programming problems with so-called *complicating variables* which couple constraints with one another and which, once fixed, result in an attractive decomposable structure that is leveraged to speed-up computation and alleviate memory consumption, allowing the solution

of large-scale MIPs. To the best of our knowledge, existing approaches from the literature have not sought explicitly strong formulations, neither have they attempted to leverage the potentially decomposable structure of the problem. This is precisely the gap we fill with the present work.

1.2. Proposed Approach & Contributions

Our approach and main contributions in this paper are:

(a) We propose an intuitive flow-based MIP formulation for learning optimal classification trees with binary data. Notably, our proposed formulation does not use big-M constraints, which are known to lead to weak LP relaxations. We also show that the resulting LP relaxation is stronger than existing alternatives.

(b) Our proposed formulation is amenable to Bender's decomposition. In particular, binary tests are selected in the master problem and each subproblem guides each datapoint through the tree via a max-flow formulation. We leverage the max-flow structure of the subproblems to solve them efficiently via min-cut procedures.

(c) We present the first polyhedral results concerning the convex hull of the feasible region of decision trees: we show that all cuts added in our proposed Benders method are *facets* of this decision tree polytope.

(d) We conduct extensive computational studies, showing that our formulations improve upon the state-of-the-art MIP algorithms, both in terms of in-sample solution quality (and speed) and out-of-sample performance.

The proposed modeling and solution paradigm can act as a building block for the faster and more accurate learning of more sophisticated trees. Continuous data can be discretized and binarized to address problems with continuous labels, see Breiman (2017). Regression trees can be obtained via minor modifications of the formulation, see e.g., Verwer and Zhang (2017). Fairness and interpretability constraints can naturally be incorporated into the problem, see Aghaei et al. (2019). We leave these studies to future work.

The rest of the paper is organized as follows. We introduce our flow-based formulation and our Bender's decomposition method in §2 and §3, respectively. We report in §5 computational experiments with popular benchmark datasets.

2. Decision Tree Formulation

2.1. Problem Formulation

We are given a training dataset $\mathcal{T} := \{\boldsymbol{x}^i, y^i\}_{i\in\mathcal{I}}$ consisting of datapoints indexed in the set $\mathcal{I}$. Each row $i \in \mathcal{I}$ of this dataset consists of F binary features indexed in the set $\mathcal{F}$ and collected

in the vector $\boldsymbol{x}^i \in \{0,1\}^F$ and a label y^i drawn from the finite set $\mathcal{K}$ of classes. We consider the problem of designing an optimal decision tree that minimizes the misclassification rate based on MIP technology.

The key idea behind our model is to augment the decision tree with a single source node s that is connected to the root node (node 1) of the tree and a single sink node t connected to all nodes of the tree, see Figure 1. This modification enables us to think of the decision tree as a *directed acyclic graph with a single source and sink node.* Datapoints *flow* from source to sink through a single path and only reach the sink if they are correctly classified (they will face a "road block" if incorrectly classified which will prevent the datapoint from traversing the graph at all). Similar to traditional algorithms for learning decision trees, we allow labels to be assigned to internal nodes of the tree. In that case, correctly classified datapoints that reach such nodes are directly routed to the sink node (as if we had a "short circuit").

Next, we introduce our notation and conventions that will be useful to present our model. We denote by $\mathcal{N}$ and $\mathcal{L}$ the sets of all internal and leaf nodes in the tree, respectively. For each node $n \in \mathcal{N} \cup \mathcal{L}$, we let $a(n)$ be the direct ancestor of n in the graph. For $n \in \mathcal{N}$, let $\ell(n)$ (resp. $r(n)$) $\in \mathcal{N} \cup \mathcal{L}$ represent the left (resp. right) direct descendant of node n in the graph. In particular, we have $a(1) = s$. We will say that we *branch on feature* $f \in \mathcal{F}$ *at node* $n \in \mathcal{N}$ if the binary test performed at n asks "Is $x^i_f = 0$"? Datapoint i will be directed left (right) if the answer is affirmative (negative).

The decision variables for our formulation are as follows. The variable $b_{nf} \in \{0,1\}$ indicates if we branch on (i.e., perform a binary test on) feature $f \in \mathcal{F}$ at node $n \in \mathcal{N}$. If $\sum_{f \in F} b_{nf} = 0$ for some node $n \in \mathcal{N}$, no feature is selected to branch on at that node, and a class is assigned to node n. We let the variable $w_{nk} \in \{0,1\}$ indicate if the predicted class for node $n \in \mathcal{N} \cup \mathcal{L}$ is $k \in \mathcal{K}$. A datapoint i is correctly classified iff it reaches some node n such that $w_{nk} = 1$ with $k = y^i$. Points that arrive at that node and that are correctly classified are directed to the sink. For each node $n \in \mathcal{N}$ and for each datapoint $i \in \mathcal{I}$, we introduce a binary valued decision variable $z^i_{a(n),n}$ which equals 1 if and only if the ith datapoint is correctly classified (i.e., reaches the sink) and traverses the edge between nodes $a(n)$ and n. We let $z^i_{n,t}$ be defined accordingly for each edge between node $n \in \mathcal{N} \cup \mathcal{L}$ and sink t.

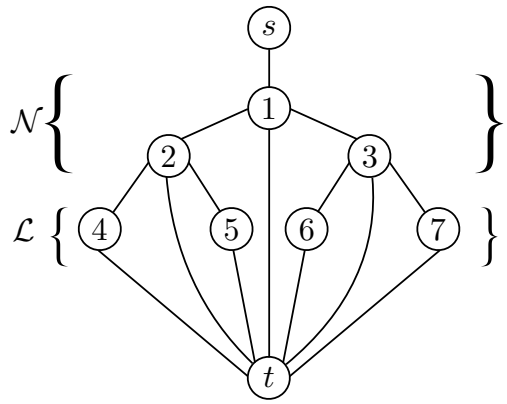


Figure 1: A classification tree of depth 2 viewed as a directed acyclic graph with a single source and sink.

The flow-based formulation for decision trees reads

$$\max \ (1-\lambda)\sum_{i\in\mathcal{I}}\sum_{n\in\mathcal{N}\cup\mathcal{L}} z^i_{n,t} - \lambda \sum_{n\in\mathcal{N}}\sum_{f\in\mathcal{F}} b_{nf} \tag{1.1}$$

$$\text{s.t.} \ \sum_{f\in\mathcal{F}} b_{nf} + \sum_{k\in\mathcal{K}} w_{nk} = 1 \qquad \forall n\in\mathcal{N} \tag{1.2}$$

$$\sum_{k\in\mathcal{K}} w_{nk} = 1 \qquad \forall n\in\mathcal{L} \tag{1.3}$$

$$z^i_{a(n),n} = z^i_{n,\ell(n)} + z^i_{n,r(n)} + z^i_{n,t} \qquad \forall n\in\mathcal{N}, i\in\mathcal{I} \tag{1.4}$$

$$z^i_{a(n),n} = z^i_{n,t} \qquad \forall i\in\mathcal{I}, n\in\mathcal{L} \tag{1.5}$$

$$z^i_{s,1} \le 1 \qquad \forall i\in\mathcal{I} \tag{1.6}$$

$$z^i_{n,\ell(n)} \le \sum_{f\in\mathcal{F}:x^i_f=0} b_{nf} \qquad \forall n\in\mathcal{N}, i\in\mathcal{I} \tag{1.7}$$

$$z^i_{n,r(n)} \le \sum_{f\in\mathcal{F}:x^i_f=1} b_{nf} \qquad \forall n\in\mathcal{N}, i\in\mathcal{I} \tag{1.8}$$

$$z^i_{n,t} \le w_{nk} \qquad \forall i\in\mathcal{I}: \ y^i = k, n\in\mathcal{N}\cup\mathcal{L} \tag{1.9}$$

$$b_{nf} \in \{0,1\} \qquad \forall n\in\mathcal{N}, f\in\mathcal{F} \tag{1.10}$$

$$w_{nk} \in \{0,1\} \qquad \forall n\in\mathcal{N}\cup\mathcal{L}, k\in\mathcal{K} \tag{1.11}$$

$$z^i_{a(n),n} \in \{0,1\} \qquad \forall n\in\mathcal{N}\cup\mathcal{L}, i\in\mathcal{I} \tag{1.12}$$

$$z^i_{n,t} \in \{0,1\} \qquad \forall n\in\mathcal{L}, i\in\mathcal{I}, \tag{1.13}$$

where $\lambda \in [0,1]$ is a regularization weight. The objective (1.1) maximizes the total number of correctly classified points $\sum_{i\in\mathcal{I}}\sum_{n\in\mathcal{N}\cup\mathcal{L}} z^i_{n,t}$ while minimizing the number of splits $\sum_{n\in\mathcal{N}}\sum_{f\in\mathcal{F}} b_{nf}$. Thus, λ controls the trade-off between these competing objectives, with larger values of lambda

corresponding to greater regularization. An interpretation of the constraints is as follows. Constraint (1.2) ensures that at each node we either branch on a feature or assign a class label to it (but not both, the label is only used if we do not branch at that node). Constraint (1.3) guarantees that each leaf has a unique predicted class label. Constraint (1.4) is a flow conservation constraint for each datapoint i and node $n \in \mathcal{N}$: it ensures that if a datapoint arrives at a node, it must also leave the node through one of its descendants, or be correctly classified and routed to t. Similarly, constraint (1.5) enforces flow conservation for each node $n \in \mathcal{L}$. The inequality constraint (1.6) ensures that at most one unit of flow can enter the graph through the source. Constraints (1.7) and (1.8) ensure that if a datapoint is routed to the left (right) at node n, then one of the features such that $x_f^i = 0$ ($x_f^i = 1$) must have been selected for branching at the node. Constraint (1.9) ensures that datapoints routed to the sink node t are correctly classified.

Given a choice of branching and labeling decisions, b and w, each datapoint is allotted one unit of flow which it attempts to guide through the graph from the source node to the sink node. If the datapoint cannot be correctly classified, the flow that will reach the sink (and by extension enter the source) will be zero. In particular note that once the b and w variables have been fixed, optimization of the flows can be done separately for each datapoint. This implies that the problem can be decomposed to speed-up computation, an idea that we leverage in Section 3. In particular, note that the optimization over flow variables can be cast as a max-flow problem for each datapoint, implying that the integrality constraint on the z variables can be relaxed to yield an equivalent formulation. We leverage this idea in our computational experiments.

Formulation (1) has several distinguishing features relative to existing MIP formulations for training decision trees

i) It does not use big-M constraints.

ii) It includes *flow variables* indicating whether each datapoint is directed to the left or right at each branching node.

iii) It only tracks datapoints that are correctly classified.

The number of variables and constraints in formulation (1) is $\mathcal{O}(2^d(|\mathcal{I}| + |\mathcal{F}|))$, where d is the tree depth. Thus, its size is of the same order as the one proposed by Bertsimas and Dunn (2017). Nonetheless, as we discuss in §2.2, the LP relaxation of formulation (1) is tighter, and therefore results in a more aggressive pruning of the search space without incurring in significant additional costs.

2.2. Strength of the Flow-Based Formulation

We now argue that formulation (1), which we henceforth refer to as *flow-based formulation*, is stronger than existing formulations from the literature. The BinOCT formulation of Verwer and

Zhang (2019) is obtained by aggregating constraints from the OCT formulation of Bertsimas and Dunn (2017) (using big-M constants). As a consequence, its relaxation is weaker. Thus, it suffices to argue that the proposed formulation is stronger than OCT.

Proposition 1. *If $\lambda = 0$, then formulation* (1) *has a stronger relaxation than OCT.*

A formal proof of Proposition 1 is given in online companion C. In the following, we provide some intuition in how formulation (1) is stronger. We work with a simplified version of the formulation of Bertsimas and Dunn (2017) specialized to the case of binary data. We provide this formulation in the online companion B.

2.2.1. No big-Ms

In this section, we argue that the absence of big-M constraints in our formulation induces a stronger formulation. In the OCT formulation, for $i \in \mathcal{I}$ and $n \in \mathcal{L}$, there are binary variables ζ such that $\zeta^i_{a(n),n} = 1$ if datapoint i is assigned to leaf node n (regardless of whether that point is correctly classified or not), and $\zeta^i_{a(n),n} = 0$ otherwise. In addition, the authors introduce a variable L_n that represents the number of missclassified points at leaf node n, and this variable is defined via constraints $L_n \geq 0$ and

$$L_n \geq \sum_{i \in \mathcal{I}} \zeta^i_{a(n),n} - \sum_{\substack{i \in \mathcal{I}: \\ y^i = k}} \zeta^i_{a(n),n} - |\mathcal{I}|(1 - w_{nk}) \;\; \forall k \in \mathcal{K}.$$

Thus, the number of correctly classified points is $|\mathcal{I}| - \sum_{n \in \mathcal{L}} L_n$. Note that this is a big-M constraint, with $M = |\mathcal{I}|$, which is activated or deactivated depending on whether $w_{nk} = 1$ or not.

The LP relaxation induced from counting correctly classified points can be improved. The number of such points, using the variables above, is

$$|\mathcal{I}| - \sum_{n \in \mathcal{L}} L_n = \sum_{n \in \mathcal{L}} \sum_{i \in \mathcal{I}} \zeta^i_{a(n),n} w_{ny^i}. \tag{2}$$

The right hand side of (2) is nonlinear (quadratic). Nonetheless, the quadratic function is *supermodular*, see Nemhauser et al. (1978), and its concave envelop can be described by introducing variables $z^i_{a(n),n} := \zeta^i_{a(n),n} w_{ny^i}$ via the system

$$\begin{aligned} &|\mathcal{I}| - \sum_{n \in \mathcal{L}} L_n \leq \sum_{n \in \mathcal{L}} \sum_{i \in \mathcal{I}} z^i_{a(n),n} \\ &z^i_{a(n),n} \leq \zeta^i_{a(n),n}, \; z^i_{a(n),n} \leq w_{ny^i} \quad \forall n \in \mathcal{N}, \; i \in \mathcal{I}. \end{aligned}$$

The additional variables z are precisely the variables used in formulation (1). Note that a simple application of this idea would require the introduction of additional variables for each pair (i, n).

However, by noting that the desired tree structure can be enforced using the new variables z only, and the original variables ζ can be dropped, we achieve this strengthening without incurring the cost of a larger formulation.

2.2.2. Improved branching constraints

To correctly enforce the branching structure of the decision-tree, Bertsimas and Dunn (2017) use (after specializing their formulation to the case of binary data) constraints of the form

$$z^i_{a(m),m} \leq 1 - b_{nf} \quad \forall i \in \mathcal{I}, m \in \mathcal{L}, n \in \mathcal{AL}(m), f \in \mathcal{F} : x^i_f = 1, \tag{3}$$

where $\mathcal{AL}(m)$ denotes the set of ancestors of m whose left branch was followed on the path from the root to m. An intrepretation of this constraint is as follows: if datapoint i reaches leaf node m, then for all nodes in the path where i took the left direction, no branching decision b_{nf} can be made that would cause the point to go right. Instead, we use constraint (1.7).

We now show that (1.7) induces a stronger LP relaxation. First, we focus on the left hand side of (1.7): due to flow conservation constraints (1.4), we find that

$$z^i_{n,\ell(n)} = \sum_{m \in \mathcal{L}: m \in \mathcal{LD}(n)} z^i_{a(m),m}$$

where, following the notation of Bertsimas and Dunn (2017), $\mathcal{LD}(n)$ is the set of left descendants of n. In particular, the left hand side of constraint (1.7) is larger than the left hand side of (3). Now, we focus on the right hand side: from constraints (1.2), we find that

$$\sum_{f \in \mathcal{F}: x^i_f = 0} b_{nf} = 1 - \sum_{k \in \mathcal{K}} y_{nk} - \sum_{f \in \mathcal{F}: x^i_f = 1} b_{nf}.$$

In particular, the right hand side of (1.7) is smaller than the right hand side of (3). Similar arguments can be made for constraint (1.8). As a consequence, the linear inequalities for branching induced from formulation (1) dominate those proposed by Bertsimas and Dunn (2017).

2.2.3. Further Strengthening of the Formulation

Formulation (1) can be strengthened even more through the addition of cuts.

Let $n \in \mathcal{N}$ be any node such that $\ell(n)$ and $r(n) \in \mathcal{L}$. Also, let $f \in \mathcal{F}$ and define $\mathcal{H} \subseteq \mathcal{I}$ as any subset of the rows such that: *a)* $i \in \mathcal{H} \Rightarrow x^i_f = 1$, and *b)* $i, j \in \mathcal{H} \Rightarrow y^i \neq y^j$. Intuitively, $\mathcal{H}$ is a set of points belonging to different classes that would all be assigned to the right branch if feature f is

selected for branching. Then, the constraint

$$\sum_{i \in \mathcal{H}} z^i_{n,\ell(n)} \leq 1 - b_{n,f} \tag{4}$$

is valid: indeed, if $b_{n,f} = 1$, then none of the points in $\mathcal{H}$ can be assigned to the left branch; and, if $b_{n,f} = 0$, then at most one of the points in $\mathcal{H}$ can be correctly classified.

None of the constraint in (1) implies (4). As a matter of fact, if all constraints (4) are added for all possible combinations of sets $\mathcal{H}$, nodes n and features f, then variables w_{nk} with $n \in \mathcal{L}$ could be dropped from the formulation, along with constraints (1.9) and (1.3). Naturally, we do not add all constraints (4) a priori, but instead use cuts to enforce them as needed.

3. A Benders' Decomposition Approach

The flow-based formulation (1) is effective at reducing the number of branch-and-bound nodes required to prove optimality when compared with existing formulations, and results in a substantial speedup in small- and medium-sized instances, see §5. However, in larger instances, the computational time required to solve the LP relaxations may become prohibitive, impairing its performance in branch-and-bound.

Recall from §2 that, if variables b and w are fixed, then the problem decomposes into $|\mathcal{I}|$ independent *subproblems*, one for each datapoint. Additionally, each problem is a maximum flow problem, for which specialized polynomial-time methods exist. Due to these characteristics, formulation (1) can be naturally tackled using Benders' decomposition, see Benders (1962). In what follows, we describe the Benders' decomposition approach.

Observe that problem (1) can be written in an equivalent fashion by making the subproblems explicit as follows:

$$\max \ (1-\lambda) \sum_{i \in \mathcal{I}} g^i(b,w) - \lambda \sum_{n \in \mathcal{N}} \sum_{f \in \mathcal{F}} b_{nf} \tag{5.1}$$

$$\text{s.t.} \ \sum_{f \in \mathcal{F}} b_{nf} + \sum_{k \in \mathcal{K}} w_{nk} = 1 \qquad \forall n \in \mathcal{N} \tag{5.2}$$

$$\sum_{k \in \mathcal{K}} w_{nk} = 1 \qquad \forall n \in \mathcal{L} \tag{5.3}$$

$$b_{nf} \in \{0,1\} \qquad \forall n \in \mathcal{N}, f \in \mathcal{F} \tag{5.4}$$

$$w_{nk} \in \{0,1\} \qquad \forall n \in \mathcal{N} \cup \mathcal{L}, k \in \mathcal{K}, \tag{5.5}$$

where, for any fixed $i \in \mathcal{I}$, w and b, $g^i(b,w)$ is defined as the optimal objective value of the max-flow

problem

$$
\begin{aligned}
\max \quad & \sum_{n \in \mathcal{N} \cup \mathcal{L}} z^i_{n,t} & & (6.1)\\
\text{s.t.} \quad & z^i_{a(n),n} = z^i_{n,\ell(n)} + z^i_{n,r(n)} + z^i_{n,t} & \forall n \in \mathcal{N} \quad & (6.2)\\
& z^i_{a(n),n} = z^i_{n,t} & \forall n \in \mathcal{L} \quad & (6.3)\\
& z^i_{s,1} \leq c^i_{s,1}(b,w) & & (6.4)\\
& z^i_{n,\ell(n)} \leq c^i_{n,\ell(n)}(b,w) & \forall n \in \mathcal{N} \quad & (6.5)\\
& z^i_{n,r(n)} \leq c^i_{n,r(n)}(b,w) & \forall n \in \mathcal{N} \quad & (6.6)\\
& z^i_{n,t} \leq c^i_{n,t}(b,w) & \forall n \in \mathcal{N} \cup \mathcal{L} \quad & (6.7)\\
& z^i_{a(n),n} \geq 0 & \forall n \in \mathcal{N} \cup \mathcal{L} \quad & (6.8)\\
& z^i_{n,t} \geq 0 & \forall n \in \mathcal{N} \cup \mathcal{L}. \quad & (6.9)
\end{aligned}
$$

In formulation (6) we use the shorthand $c_{nn'}(b,w)$ to represent upper bounds on the decision variables z. These values can be interpreted as edge capacities in the flow problem, and are given as $c^i_{s,1}(b,w) := 1$ for all $n \in \mathcal{N}$, $c^i_{n,\ell(n)}(b,w) := \sum_{f \in \mathcal{F}: x^i_f = 0} b_{nf}$ and $c^i_{n,r(n)}(b,w) := \sum_{f \in \mathcal{F}: x^i_f = 1} b_{nf}$ for all $n \in \mathcal{N} \cup \mathcal{L}$, and finally $c^i_{n,t}(b,w) := w_{ny^i}$. Note that $g^i(b,w) = 1$ if point i is correctly classified given the tree structure and class labels induced by (b,w).

From the well-known max-flow/min-cut duality, we find that $g^i(b,w)$ also equals the optimal value of the dual of the above max-flow problem, which is expressible as

$$
\begin{aligned}
\min \quad & c^i_{s,1}(b,w)q_{s,1} + \sum_{n \in \mathcal{N}} c^i_{n,\ell(n)}(b,w)q_{n,\ell(n)} + \sum_{n \in \mathcal{N}} c^i_{n,r(n)}(b,w)q_{n,r(n)} + \sum_{n \in \mathcal{N} \cup \mathcal{L}} c^i_{n,t}(b,w)q_{n,t} & & (7.1)\\
\text{s.t.} \quad & q_{s,1} + p_1 \geq 1 & & (7.2)\\
& q_{n,\ell(n)} + p_{\ell(n)} - p_n \geq 0 & \forall n \in \mathcal{N} \quad & (7.3)\\
& q_{n,r(n)} + p_{r(n)} - p_n \geq 0 & \forall n \in \mathcal{N} \quad & (7.4)\\
& q_{n,t} - p_n \geq 0 & \forall n \in \mathcal{N} \cup \mathcal{L} \quad & (7.5)\\
& q_{s,1} \geq 0 & & (7.6)\\
& q_{n,\ell(n)}, q_{n,r(n)} \geq 0 & \forall n \in \mathcal{N} \quad & (7.7)\\
& q_{n,t} \geq 0 & \forall n \in \mathcal{N} \cup \mathcal{L} \quad & .(7.8)
\end{aligned}
$$

Problem (7) is a minimum cut problem, where variable p_n is one if and only if node n is in the source set (we implicitly fix $p_s = 1$), and variable $q_{i,j}$ is one if and only if arc (i,j) is part of the minimum cut. Note that the feasible region (7.2)-(7.8) of the minimum cut problem does not depend on the variables (b,w); we denote this feasible region by $\mathcal{P}$.

We can now reformulate the master problem (5) as follows:

$$\max \ (1-\lambda)\sum_{i\in\mathcal{I}} g^i - \lambda \sum_{n\in\mathcal{N}}\sum_{f\in\mathcal{F}} b_{nf} \tag{8.1}$$

$$\text{s.t.} \ g^i \leq c^i_{s,1}(b,w)q_{s,1} + \sum_{n\in\mathcal{N}} c^i_{n,\ell(n)}(b,w)q_{n,\ell(n)} + \sum_{n\in\mathcal{N}} c^i_{n,r(n)}(b,w)q_{n,r(n)} + \sum_{n\in\mathcal{N}\cup\mathcal{L}} c^i_{n,t}(b,w)q_{n,t} \qquad \forall q:(p,q)\in\mathcal{P} \tag{8.2}$$

$$\sum_{f\in\mathcal{F}} b_{nf} + \sum_{k\in\mathcal{K}} w_{nk} = 1 \qquad \forall n\in\mathcal{N} \tag{8.3}$$

$$\sum_{k\in\mathcal{K}} w_{nk} = 1 \qquad \forall n\in\mathcal{L} \tag{8.4}$$

$$g^i \leq 1 \qquad \forall i\in\mathcal{I} \tag{8.5}$$

$$b_{nf}\in\{0,1\} \qquad \forall n\in\mathcal{N}, f\in\mathcal{F} \tag{8.6}$$

$$w_{nk}\in\{0,1\} \qquad \forall n\in\mathcal{N}\cup\mathcal{L}, k\in\mathcal{K}. \tag{8.7}$$

In the above formulation, we have added constraint (8.5) to make sure we get bounded solutions in the relaxed master problem. Note that constraint (8.2) can be relaxed to only hold $\forall q : (p,q)\in \text{ext}(\mathcal{P})$, where $\text{ext}(\mathcal{P})$ denotes the extreme points of $\mathcal{P}$. These extreme points correspond to cuts induced by (7.2)-(7.8) in the graph. Moreover, observe that equalities (8.3) and (8.4) can be relaxed to inequalities without loss of generality. Indeed, in any feasible solution where $\sum_{f\in\mathcal{F}} b_{nf} + \sum_{k\in\mathcal{K}} w_{nk} < 1$ for some $n\in\mathcal{N}$, it is possible to set any w_{nk} to unity to obtain a feasible solution with identical objective value and where (8.3) is satisfied at equality. We define $\mathcal{H}_=$ as the set of (b,w,g) satisfying constraints (8.2)-(8.7), and define $\mathcal{H}_\leq$ as the set of points satisfying the inequality version of (8.2)-(8.7). In the next section we discuss effective implementations of problem (8).

4. Generating Strong Cuts on the Fly

Formulation (8) contains an exponential number of inequalities (8.2), and needs to be implemented using row generation, wherein constraints (8.2) are initially dropped and added as cuts on the fly during optimization. Row generation can be implemented in modern MIP optimization solvers via callbacks, by adding lazy constraints at relevant nodes of the branch-and-bound tree. Identifying which constraint (8.2) to add can in general be done by solving a minimum cut problem, and could in principle be solved via well-known algorithms, such as Goldberg and Tarjan (1988) and Hochbaum (2008).

Row generation methods for integer programs may require a long time to converge to an optimal solution if each cut added is weak for the feasible region of interest, as illustrated for example by the poor performance of the pure cutting plane algorithm of Gomory (1958). Nonetheless, cutting planes have been extremely successful at solving integer programs when the cuts added are strong or, ideally, "facet-defining" for the convex hull of the feasible region. Formally, facet-defining cuts are those cuts which are necessary to describe the convex hull. For example, integer programming formulations for traveling salesman problems contain an exponential number of "subtour elimination" constraints that are added on the fly as cuts. Nonetheless, all such inequalities are facet defining for the convex hull of the feasible region, see Grötschel et al. (1985), and cutting plane methods are able to find provably optimal tours to problems with tens of thousands of variables or more Applegate et al. (2009). Unfortunately, as illustrated by the following example, several of the inequalities (8.2) may actually be weak for $\text{conv}(\mathcal{H}_=)$ and $\text{conv}(\mathcal{H}_\leq)$, where $\text{conv}(\mathcal{H})$ denotes the convex hull of $\mathcal{H}$.

Example 1. *Consider an instance of Problem* (8) *with a depth* $d = 1$ *decision-tree (i.e.,* $\mathcal{N} = \{1\}$ *and* $\mathcal{L} = \{2, 3\}$*) and a dataset involving a single feature (*$\mathcal{F} = \{1\}$*). Consider datapoint* i *such that* $x_1^i = 0$ *and* $y^i = k$*. Suppose that the solution to the master problem is such that we branch on (the unique) feature at node 1 and predict class* $k' \neq k$ *at node 2. Then, datapoint* i *is routed left at node 1 and is misclassified. A valid min-cut for the resulting graph includes all arcs incoming into the sink, i.e.,* $q_{n,n'} = 1$ *iff* $n' = t$*. The associated cut* (8.2) *reads*

$$g^i \leq w_{1k} + w_{2k} + w_{3k}. \tag{9}$$

Intuitively, (9) *states that datapoint* i *can be correctly classified if its class label is assigned to at least one node, and is valid for conv*$(\mathcal{H}_=)$ *and conv*$(\mathcal{H}_\leq)$*. However, since datapoint* i *cannot be routed to node* 3*, the stronger inequality*

$$g^i \leq w_{1k} + w_{2k} \tag{10}$$

is valid for conv$(\mathcal{H}_=)$*, conv*$(\mathcal{H}_\leq)$ *and dominates* (9). ■

Therefore, an implementation of formulation (8) using general purpose min-cut algorithms to identify constraints to add may perform poorly. This motivates us to develop a tailored algorithm that exploits the structure of the graph induced by capacities $c(b, w)$. As we will show, our algorithm exhibits substantially faster runtimes than general purpose min-cut methods and returns inequalities that are never dominated, resulting in faster convergence of the Benders' decomposition approach.

Algorithm 1 shows the proposed procedure, which can be called at *integer nodes* of the branch-and-bound tree. For notational convenience, we define $b_{n,\ell(n)} = b_{n,r(n)} = 0$ for leaf nodes $n \in \mathcal{L}$. Since at each iteration in the main loop (lines 5-23), the value of n is updated to a descendant of n, the algorithm terminates in a most $\mathcal{O}(d)$ iterations, where d is the depth of the tree – since $|\mathcal{N} \cup \mathcal{L}|$ is $\mathcal{O}(2^d)$, the complexity is logarithmic in the size of the tree. Figure 2 illustrates graphically

Algorithm 1. We now prove that Algorithm 1 is indeed a valid *separation algorithm.*

Algorithm 1 Separation procedure

Input: (b, w, g) satisfying (8.3)-(8.7);
$i \in \mathcal{I}$: datapoint used to generate the cut.
Output: -1 if all constraints (8.2) are satisfied;
values for min-cut q otherwise.

1: **if** $g^i = 0$ **return** -1
2: **Initialize** $q \leftarrow 0$ ▷ No arcs in the cut
3: **Initialize** $n \leftarrow 1$ ▷ Current node=root
4: **Initialize** $\mathcal{S} \leftarrow \{s\}$ ▷ $\mathcal{S}$ is the source set of the cut
5: **loop**
6: $\mathcal{S} \leftarrow \mathcal{S} \cup \{n\}$
7: **if** $c^i_{n,\ell(n)}(b, w) = 1$ **then**
8: $q_{n,r(n)} \leftarrow 1$ ▷ Arcs to the right are in the cut
9: $q_{n,t} \leftarrow 1$ ▷ Arcs to the sink are in the cut
10: $n \leftarrow \ell(n)$ ▷ Datapoint i is routed left
11: **else if** $c^i_{n,r(n)}(b, w) = 1$ **then**
12: $q_{n,\ell(n)} \leftarrow 1$ ▷ Arcs to the left are in the cut
13: $q_{n,t} \leftarrow 1$ ▷ Arcs to the sink are in the cut
14: $n \leftarrow r(n)$ ▷ Datapoint i is routed right
15: **else if** $c^i_{n,t}(b, w) = 0$ **then**
16: $q_{n,\ell(n)} \leftarrow 1$ ▷ Arcs to the left are in the cut
17: $q_{n,r(n)} \leftarrow 1$ ▷ Arcs to the right are in the cut
18: $q_{n,t} \leftarrow 1$ ▷ Datapoint i is misclassified
19: **return** q
20: **else** ▷ $c^i_{n,t}(b, w) = 1$ in this case
21: **return** -1 ▷ i is correctly classified
22: **end if**
23: **end loop**

Proposition 2. *Given $i \in \mathcal{I}$ and (b, w, g) satisfying* (8.3)-(8.7)*, Algorithm 1 either finds a violated inequality* (8.2) *or proves that all such inequalities are satisfied.*

Proof. Note that the right hand side of (8.2), which corresponds to the capacity of a cut in the graph, is nonnegative. Therefore, if $g^i = 0$ (line 1), all inequalities are automatically satisfied. Since (b, w) is integer, all arc capacities in formulations (6) and (7) are either 0 or 1. Moreover, since $g^i \leq 1$, we find that either the value of a minimum cut is 0 and there exists a violated inequality, or the value of a minimum cut is at least 1 and there is no violated inequality. Finally, there exists a 0-capacity cut if and only if s and t belong to different connected components in the graph induced by $c(b, w)$.

The component connected to s can be found using depth-first search. For any fixed $n \in \mathcal{N} \cup \mathcal{L}$, constraints (8.3)-(8.4) and the definition of $c(b, w)$ imply that at most one arc (n, n') outgoing from n can have capacity 1. If arc (n, n') has capacity 1 and $n' \neq t$ (lines 7-14), then n' can be

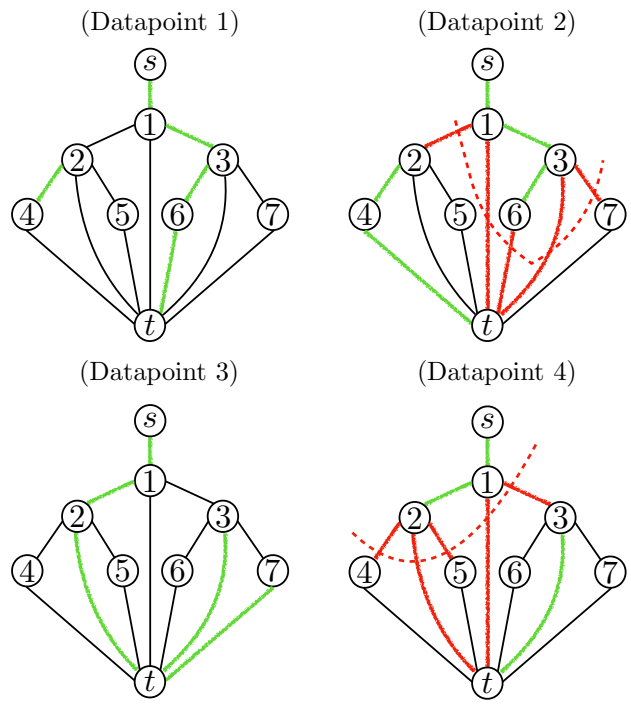


Figure 2: Pictorial description of Algorithm 1. Green arcs (n, n') have capacity $c^i_{n,n'}(b, w) = 1$ (and others capacity 0). Red arcs are those in the minimum cut. On the left, examples with minimum cut value equal to 1 (constraint (8.2) is satisfied). On the right, examples with minimum cut values of 0 (new constraints added).

added to the component connected to s (set $\mathcal{S}$) and all other outgoing arcs from n (which have capacity of 0) can be added to the min-cut (at zero cost). If all outgoing arcs from n have capacity 0, they can be added to the min-cut. In that case, the connected components to s end at node n. If the unique outgoing arc from node n that has capacity 1 is (n, t), then s and t are in the same connected component and the value of the minimum cut is at least 1. Therefore, the connected component $\mathcal{S}$ to s corresponds to a path from s to a node n where no branching is performed: if $c^i_{n,t} = 1$ then t is also in this connected component and no cut is added (line 21): otherwise, a violated cut has been found (line 19). ■

In addition to providing a very fast method for generating cuts at integer nodes of a branch-and-bound tree, Algorithm 1 is also guaranteed to generate facet-defining cuts of $\text{conv}(H_{\leq})$. Such cuts are never dominated.

Theorem 1. *All violated inequalities found by Algorithm 1 are facet-defining for conv($\mathcal{H}_{\leq}$).*

We defer the proof of Theorem 1 to the supplemental material A.

Example 2 (Example 1 Continued)**.** *In the instance considered in Example 1, if $b_{1f} = 1$ and $w_{2k} = 0$, then the cut generated by the algorithm ($q_{1,r(1)} = q_{1,t} = q_{2,t} = 1$) is precisely* (10)*. If $b_{1f} = 0$ and*

$w_{1k} = 0$ (which is feasible in conv($H_{\leq}$)) in the solution to the master problem used to generate the cut, then the cut returned by Algorithm 1 ($q_{1,\ell(1)} = q_{1,t} = 1$) is

$$g^i \leq w_{1k} + b_{1f}.$$

For all other possible values of (b, w), Algorithm 1 does not find a violated cut.

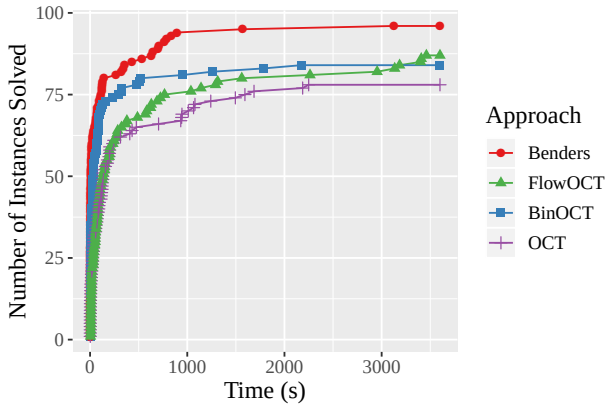
5. Experiments

Approaches and Datasets. We evaluate our two approaches on eight publicly available datasets. The number of rows ($\mathcal{I}$), number of one-hot encoded features ($\mathcal{F}$), and number of class labels ($\mathcal{K}$) for each dataset are given in Table 1. We compare the flow-based formulation (`FlowOCT`) and its Benders' decomposition (`Benders`) against the formulations proposed by Bertsimas and Dunn (2017) (`OCT`) and Verwer and Zhang (2019) (`BinOCT`). As the code used for OCT is not publicly available, we implemented the corresponding formulation (adapted for the case of binary data). The details of this implementation are given in the online companion B.

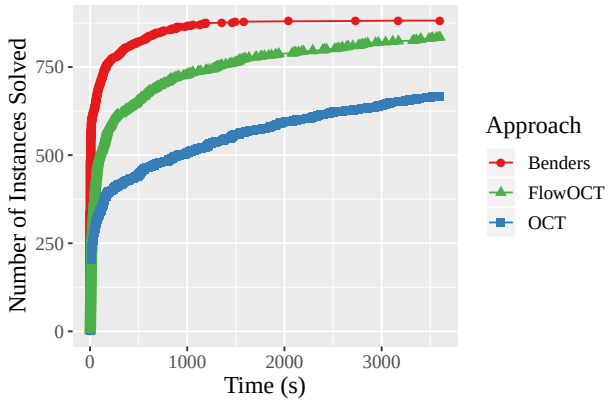
Table 1: Datasets used in the experiments.

Dataset	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{K}\rvert$
monk3	122	15	2
monk1	124	15	2
monk2	169	15	2
house-votes-84	232	16	2
balance-scale	625	20	3
tic-tac-toe	958	27	2
car_evaluation	1728	20	4
kr-vs-kp	3196	38	2

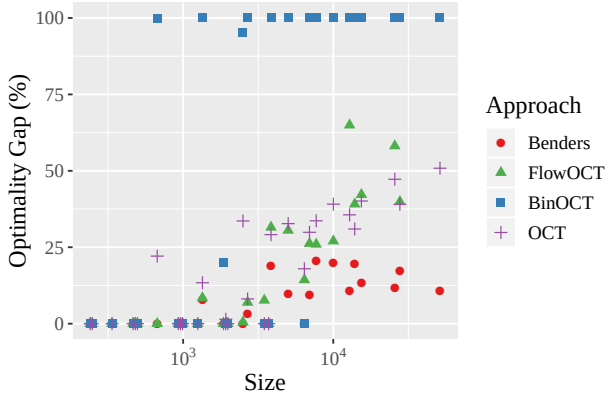
Experimental Setup. Each dataset is split into three parts: the training set (50%), the validation set (25%), and the testing set (25%). The training and validation sets are used to tune the value of the hyperparameter λ. We repeat this process 5 times with 5 different samples. We test values of $\lambda = 0.1j$ for $j = 0, \ldots, 9$. Finally, we use the best λ to train a tree using the training and evaluation sets from the previous step, which we then evaluate against the testing set to determine the out-of-sample accuracy. All approaches are implemented in Python programming language and solved by the Gurobi 8.1 solver. All problems are solved on a single core of SL250s Xeon CPUs by HPE and 4gb of memory with a one hour time limit.



(a) Performance profile with $\lambda = 0$.



(b) Performance profile with $\lambda > 0$.



(c) Optimality gaps as a function of the size$= 2^d \times |\mathcal{I}|$.

Figure 3: Summary of optimization performance.

In-Sample (Optimization) Performance. Figure 3 summarizes the in-sample performance, i.e., how good the methods are at solving the optimization problems. Detailed results are provided in the online companion B. From Figure 3(a), we observe that for $\lambda = 0$, `BinOCT` is able to solve 79 instances within the time limit (and outperforms `OCT`), but `Benders` solves the same quantity of instances in only 140 seconds, *resulting in a* 30× *speedup*. Similarly, from Figure 3(b), it can be seen

that for $\lambda > 0$, `OCT` is able to solve 666 instances within the time limit[1], while `Benders` requires only 70 seconds to do so, *resulting in a* 50× *speedup*. Finally, Figure 3(c) shows the optimality gaps proven as a function of the dimension. We observe that all methods result in a gap of 0% in small instances. As the dimension increases, BinOCT (which relies on weak formulations but fast enumeration) yields 100% optimality gaps in most cases. `OCT` and `BinOCT` prove better gaps, but the performance degrades substantially as the dimension increases. `Benders` results in the best performance, proving optimality gaps of 20% or less regardless of dimension.

Out-of-Sample (Statistical) Performance. Table 2 reports the out-of-sample accuracy after cross-validation. Each row represents the average over the five samples. We observe that the better optimization performance translates to superior statistical properties as well: `OCT` is the best method in two instances (excluding ties), `BinOCT` in six, while the new formulations `FlowOCT` and `Benders` are better in 13 (of which `Benders` accounts for 10, and is second after `FlowOCT` in an additional two).

[1] `BinOCT` does not include the option to have a regularization parameter, and is omitted.

Table 2: Out of sample accuracy

Dataset	Depth	OCT	BinOCT	FlowOCT	Benders
monk3	2	**92.3**	**92.3**	**92.3**	**92.3**
monk3	3	83.2	**91**	**91**	**91**
monk3	4	91	85.2	**92.3**	91
monk3	5	87.1	87.7	**92.3**	91.6
monk1	2	71	**72.3**	**72.3**	71
monk1	3	**83.2**	82.6	81.3	81.3
monk1	4	**100**	99.4	**100**	**100**
monk1	5	93.5	96.8	**100**	**100**
monk2	2	**56.7**	49.8	**56.7**	**56.7**
monk2	3	62.3	58.1	**63.7**	63.3
monk2	4	59.5	60.5	58.6	**64.2**
monk2	5	**63.3**	55.8	62.3	61.9
house-votes-84	2	79.3	96.2	**97.2**	**97.2**
house-votes-84	3	**97.2**	94.1	**97.2**	**97.2**
house-votes-84	4	**96.9**	94.8	**96.9**	95.5
house-votes-84	5	95.2	93.1	96.9	**97.2**
balance-scale	2	**68.7**	67.9	**68.7**	**68.7**
balance-scale	3	69	**71.5**	69.8	71
balance-scale	4	68.5	**73.9**	73.2	71.7
balance-scale	5	65.7	75.3	71.6	**76.8**
tic-tac-toe	2	**66.7**	65.9	**66.7**	**66.7**
tic-tac-toe	3	68.1	72.2	68.5	**72.6**
tic-tac-toe	4	70.4	**80.3**	68.7	77.1
tic-tac-toe	5	69.7	78.9	66.3	**79.3**
car_evaluation	2	**76.5**	**76.5**	**76.5**	**76.5**
car_evaluation	3	73.3	78.4	76.7	**79.1**
car_evaluation	4	75.2	**80.3**	71.6	79.7
car_evaluation	5	74.8	**81.3**	61.6	80.5
kr-vs-kp	2	73.7	**87.2**	70.5	**87.2**
kr-vs-kp	3	69.3	87.8	61.2	**89.9**
kr-vs-kp	4	64.7	90.8	54.3	**91**
kr-vs-kp	5	62.7	**87.1**	45.8	86.7

References

Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1418–1426, 2019.

David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal tsp tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

Mohammad Javad Azizi, Phebe Vayanos, Bryan Wilder, Eric Rice, and Milind Tambe. Designing fair, efficient, and interpretable policies for prioritizing homeless youth for housing resources. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 35–51. Springer, 2018.

Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. *arXiv preprint arXiv:1812.09991*, 2018.

Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

Leo Breiman. Classification and regression trees. Technical report, 1984.

Leo Breiman. *Classification and regression trees*. Routledge, 2017.

Dragos Ciocan and Velibor Mišić. Interpretable optimal stopping. 2018.

IBM ILOG CPLEX. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53): 157, 2009.

Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

Martin Grötschel, Manfred W Padberg, et al. Polyhedral theory. *The traveling salesman problem*, pages 251–305, 1985.

Oktay Günlük, Jayant Kalagnanam, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *arXiv preprint arXiv:1612.03225*, 2018.

Incorporate Gurobi Optimization. Gurobi optimizer reference manual. *URL http://www. gurobi. com*, 2015.

Dorit S Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations research*, 56(4):992–1009, 2008.

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. *arXiv preprint arXiv:1904.12847*, 2019.

Laurent Hyafil and Ronald L Rivest. Constructing optimal binary search trees is NP complete. *Information Processing Letters*, 1976.

Max Kuhn, Steve Weston, Mark Culp, Nathan Coulter, and Ross Quinlan. Package 'c50', 2018.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Velibor V Mišic. Optimization of tree ensembles. *arXiv preprint arXiv:1705.10883*, 2017.

Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao Marques-Silva. Learning optimal decision trees with SAT. In *IJCAI*, pages 1362–1368, 2018.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

Siegfried Nijssen and Elisa Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1):9–51, 2010.

John Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

John Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

Terry Therneau, Beth Atkinson, Brian Ripley, and Maintainer Brian Ripley. Package 'rpart'. 2015.

Hélene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. In *The 25th International Conference on Principles and Practice of Constraint Programming (CP2019)*, 2019.

Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–103. Springer, 2017.

Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *33rd AAAI Conference on Artificial Intelligence*, 2019.

A. Proof of Theorem 1

The proof proceeds in three steps. We fix $i \in \mathcal{I}$. We derive the specific structure $(p, q) \in \mathcal{P}$ of the cuts associated with datapoint i generated by our procedure. We then provide $|\mathcal{N} \times \mathcal{F}| + |\mathcal{L} \times \mathcal{K}| + |\mathcal{I}|$ points that lie in $\text{conv}(\mathcal{H}_{\leq})$ and at each of which the cut generated holds with equality. Since the choice of $i \in \mathcal{I}$ is arbitrary and since the cuts generated by our procedure are valid (by construction), this will conclude the proof.

To minimize notational overhead, we assume throughout this proof that $\lambda = 0$. In this case, an optimal solution to the master problem where $\sum_{k \in \mathcal{K}} w_{nk} = 0$ for all $n \in \mathcal{N}$ can always be obtained. Given a set A and a point $a \in A$, we use $A \setminus a$ as a shorthand for $A \setminus \{a\}$. Finally, we let $e_{ij} = 1$ be a vector (whose dimensions will be clear from the context) with a 1 in coordinates (i, j) and 0 elsewhere.

Fix $i \in \mathcal{I}$. Let $(\bar{b}, \bar{w}, \bar{g})$ be optimal in the (relaxed) master problem and assume $\sum_{k \in \mathcal{K}} \bar{w}_{nk} = 0$ for all $n \in \mathcal{N}$. Given $j \in \mathcal{I}$, let $n(j) \in \mathcal{L}$ be the leaf of the tree defined by $(\bar{b}, \bar{w})$ that datapoint j is assigned to. Given $n \in \mathcal{N}$, let $f(n) \in \mathcal{F}$ be the feature selected for testing at node n under $(\bar{b}, \bar{w})$, i.e., $\bar{b}_{nf(n)} = 1$.

We now derive the structure of the cuts (8.2) generated by Algorithm 1 (see also the proof of Proposition 2) when $(\bar{b}, \bar{w}, \bar{g})$ is input. A minimum cut is returned by Algorithm 1 if and only if s and t belong to different connected components in the tree induced by $(\bar{b}, \bar{w})$. Under this assumption, since $\sum_{k \in \mathcal{K}} \bar{w}_{nk} = 0$ for all $n \in \mathcal{N}$, the connected component $\mathcal{S}$ constructed in Algorithm 1 forms a path from s to $n_d = n(i) \in \mathcal{L}$, i.e., $\mathcal{S} = \{s, n_1, n_2, \ldots, n_d\}$. The minimum cut q obtained from Algorithm 1 then corresponds to the arcs adjacent to nodes in $\mathcal{S}$ that do not belong to the path formed by $\mathcal{S}$. Therefore, $q_{s,1} = 0$, $q_{n,t} = 1$ iff $n = n(i)$, and for each $n \in \mathcal{N}$,

$$q_{n,\ell(n)} = 1 \quad \text{iff} \quad n \in p \quad \text{and} \quad \sum_{f \in \mathcal{F}: x_f^i = 1} \bar{b}_{nf} = 1, \text{ and}$$

$$q_{n,r(n)} = 1 \quad \text{iff} \quad n \in p \quad \text{and} \quad \sum_{f \in \mathcal{F}: x_f^i = 0} \bar{b}_{nf} = 1.$$

Therefore, the cut (8.2) returned by Algorithm 1 reads

$$g_i \;\leq\; w_{n(i)y^i} + \sum_{n \in \mathcal{S}} \sum_{\substack{f \in \mathcal{F}: \\ x_f^i \neq x_{f(n)}^i}} b_{nf}. \tag{11}$$

Next, we give $|\mathcal{N} \times \mathcal{F}| + |\mathcal{L} \times \mathcal{K}| + |\mathcal{I}|$ affinely independent points in $\mathcal{H}_{\leq}$ for which (11) holds with equality. Given a vector $b \in \{0, 1\}^{|\mathcal{N}| \times |\mathcal{F}|}$, we let $b_{\mathcal{S}}$ (resp. $b_{\mathcal{N} \setminus \mathcal{S}}$) collect those elements of b whose first index $n \in \mathcal{S}$ (resp. $n \notin \mathcal{S}$). We now describe the points, which are also summarized in

Table 3.

Table 3: Companion table for the proof of Theorem 1: list of affinely independent points that live on the cut generated by inputing $i \in \mathcal{I}$ and $(\bar{b}, \bar{w}, \bar{g})$ in Algorithm 1.

#	condition	dim= / sol=	$\lvert\mathcal{S}\rvert \times \lvert\mathcal{F}\rvert$ $b_{\mathcal{S}}$	$\lvert\mathcal{N} \setminus \mathcal{S}\rvert \times \lvert\mathcal{F}\rvert$ $b_{\mathcal{N}\setminus\mathcal{S}}$	$\lvert\mathcal{L}\rvert \times \lvert\mathcal{K}\rvert$ w	$\lvert\mathcal{I}\rvert$ g
1	"baseline" point		$\bar{b}_{\mathcal{S}}$	0	0	0
2	$n \in \mathcal{L}, k \in \mathcal{K} \setminus y^i$		$\bar{b}_{\mathcal{S}}$	0	e_{nk}	0
3	$n \in \mathcal{L} \setminus n(i)$		$\bar{b}_{\mathcal{S}}$	0	e_{ny^i}	0
4	$n = n(i)$		$\bar{b}_{\mathcal{S}}$	0	$e_{n(i)y^i}$	e_i
5	$n \in \mathcal{N} \setminus \mathcal{S}, f \in \mathcal{F}$		$\bar{b}_{\mathcal{S}}$	e_{nf}	0	0
6	$n \in \mathcal{S}$		$\bar{b}_{\mathcal{S}} - e_{nf(n)}$	0	0	0
7	$n \in \mathcal{S}, f \in \mathcal{F} : f \neq f(n), x^i_f = x^i_{f(n)}$		$\bar{b}_{\mathcal{S}} - e_{nf(n)} + e_{nf}$	0	0	0
8	$n \in \mathcal{S}, f \in \mathcal{F} : f \neq f(n), x^i_f \neq x^i_{f(n)}$		$\bar{b}_{\mathcal{S}} - e_{nf(n)} + e_{nf}$	0	$\sum_{n \in \mathcal{L}: n \neq n(i)} e_{ny^i}$	e_i
9	$j \in \mathcal{I} \setminus i : y^j \neq y^i$		$\bar{b}_{\mathcal{S}}$	$\bar{b}_{\mathcal{N}\setminus\mathcal{S}}$	$e_{n(j)y^j}$	e_j
10	$j \in \mathcal{I} \setminus i : y^j = y^i$, $n(j) \neq n(i)$		$\bar{b}_{\mathcal{S}}$	$\bar{b}_{\mathcal{N}\setminus\mathcal{S}}$	$e_{n(j)y^j}$	e_j
11	$j \in \mathcal{I} \setminus i : y^j = y^i$, $n(j) = n(i)$		$\bar{b}_{\mathcal{S}}$	$\bar{b}_{\mathcal{N}\setminus\mathcal{S}}$	$e_{n(i)y^i}$	$e_i + e_j$

1 One point that is a "baseline" point; all other points are variants of it. It is given by $b_{\mathcal{S}} = \bar{b}_{\mathcal{S}}$, $b_{\mathcal{N}\setminus\mathcal{S}} = 0$, $w = 0$ and $g = 0$ and corresponds to selecting the features to branch on according to $\bar{b}$ for nodes in $\mathcal{S}$ and setting all remaining variables to 0. The baseline point belongs to $\mathcal{H}_{\leq}$ and constraint (11) is active at this point.

2-4 $|\mathcal{L}| \times |\mathcal{K}|$ points obtained from the baseline point by varying the w coordinates and adjusting g as necessary to ensure (11) remains active: **2:** $|\mathcal{L}| \times (|\mathcal{K}| - 1)$ points, each associated with a leaf $n \in \mathcal{L}$ and class $k \in \mathcal{K} : k \neq y^i$, where the label of leaf n is changed to k. **3:** $|\mathcal{L}| - 1$ points, each associated with a leaf $n \in \mathcal{L} : n \neq n(i)$, where the class label of n is changed to y^i. **4:** One point where the class label of leaf $n(i)$ is set to y^i, allowing for correct classification of datapoint i; in this case, the value of the rhs of (11) is 1, and we set $g^i = 1$ to ensure the cut (11) remains active.

5 $|\mathcal{N} \setminus \mathcal{S}| \times |\mathcal{F}|$ points obtained from the baseline point by varying the $b_{\mathcal{N}\setminus\mathcal{S}}$ coordinates. Each point is associated with a node $n \in \mathcal{N}\setminus\mathcal{S}$ and feature $f \in \mathcal{F}$ and is obtained by changing the decision to branch on feature f and node n to 1. As those branching decisions do not impact the routing of datapoint i the value of the rhs of inequality (11) remains unchanged and the inequality stays active.

6-8 $|\mathcal{S}| \times |\mathcal{F}|$ points, obtained from the baseline point by varying the $b_{\mathcal{S}}$ coordinates and adjusting w and g as necessary to guarantee feasibility of the resulting point and to ensure that (11) stays active. **6:** $|\mathcal{S}|$ points, each associated with a node $n \in \mathcal{S}$ obtained by not branching on feature $f(n)$ at node n (nor on any other feature), resulting in a "dead-end" node. The value of the rhs of (11) is unchanged in this case and the inequality remains active. **7-8:** $|\mathcal{S}|$ points, each associated with a node $n \in \mathcal{S}$ and feature $f \neq f(n)$. **7:** If the branching decision $f(n)$ at node n is replaced with a branching decision that results in the same path for datapoint i, i.e., if $x^i_f = x^i_{f(n)}$, it is possible to swap those decisions without affecting the value of the rhs in inequality (11). **8:** If a feature that causes i to change paths is chosen for branching, i.e., if $x^i_f \neq x^i_{f(n)}$, then the value of the rhs of (11) is increased by 1, and we set $g^i = 1$ to ensure the inequality remains active; to guarantee feasibility of the resulting point, we label each leaf node except for $n(i)$ with the class y^i, which does not affect inequality (11).

9-11 $|I| - 1$ points, obtained from the baseline point by letting $b_{\mathcal{N}\setminus\mathcal{S}} = \bar{b}_{\mathcal{N}\setminus\mathcal{S}}$ and adjusting w and g as necessary. Each point is associated with a datapoint $j \in \mathcal{I} \setminus i$ which we allow to be correctly classified. **9:** If datapoint j has a different class than datapoint i ($y^j \neq y^i$), we label the leaf node where j is routed to with the class of j, i.e., $w_{n(j)y^j} = 1$. The value of the rhs of (11) is unaffected the inequality remains active. **10:** If datapoint j has the same class as datapoint i but is routed to a different leaf than i, an argument paralleling that in **9** can be made. **11:** If datapoint j has the same class as datapoint i and is routed to the same leaf $n(i)$, we label $n(i)$ with the class of $y^i = y^j$ and set $g^j = 1$; the value of the rhs of (11) increases by 1. Thus, we set also correctly classify datapoint i by setting $g^i = 1$ to ensure that (11) is active.

The $|\mathcal{N} \times \mathcal{F}| + |\mathcal{L} \times \mathcal{K}| + |\mathcal{I}|$ points constructed above, see also Table 3, are affinely independent. Indeed, each differs from the previously introduced points in at least one coordinate. All these points also belong to $\mathcal{H}_{\leq}$. This concludes the proof.

B. OCT

In this section, we provide a simplified version of the formulation of Bertsimas and Dunn (2017) specialized to the case of binary data.

We start with introducing the notation that is used for the formulation. Let $\mathcal{N}$ and $\mathcal{L}$ denote the sets of all internal and leaf nodes in the tree structure. For each node $n \in \mathcal{N} \cup \mathcal{L} \setminus \{1\}$, $a(n)$ refers to the direct ancestor of node n. $\mathcal{AL}(n)$ is the set of ancestors of n whose left branch has been followed on the path from the root node to n, and similarly $\mathcal{AR}(n)$ is the set of right-branch ancestors, such that $\mathcal{A}(n) = \mathcal{AL}(n) \cup \mathcal{AR}(n)$

Let b_{nf} be a binary decision variable where $b_{nf} = 1$ iff at node n, feature f is branched upon. For each datapoint i at node $n \in \mathcal{N}$ a test $\sum_{f \in \mathcal{F}} b_{nf} x^i_f < v_n$ is performed where $v_n \in \mathbb{R}$ is a decision

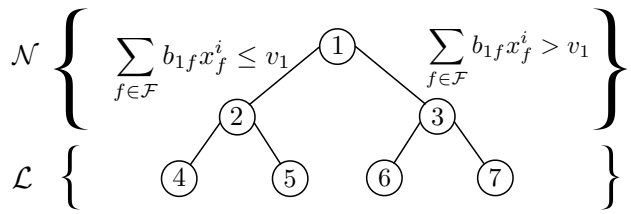


Figure 4: A classification tree of depth 2

variable represeinting the cut-off value of the test. If datapoint i passes the test it follows the left branch otherwise it follows the right one. Let $d_n = 1$ iff node n applies a split, to allow having this option not to split at a node. To track each datapoint i through the tree, the decision variable $\zeta^i_{a(n),n}$ is introduced where $\zeta^i_{a(n),n} = 1$ iff datapoint i is in node n.

Let P_{nk} to be the number of datapoints of class k assigned to leaf node n and P_n to be the total number of datapoints in leaf node n. Let w_{nk} denote the prediction of each leaf node n, where $w_{nk} = 1$ iff the predicted label of node n is $k \in \mathcal{K}$. At the end, let L_n denote the number of

missclassified datapoints at node n.

$$
\begin{aligned}
\max \ & (1-\lambda)\left(|\mathcal{I}| - \sum_{n\in\mathcal{L}} L_n\right) - \lambda \sum_{n\in\mathcal{N}} d_n && && (12.1)\\
\text{s.t.} \ & L_n \geq P_n - P_{nk} - |\mathcal{I}|(1-w_{nk}) && \forall k\in\mathcal{K}, n\in\mathcal{L} && (12.2)\\
& L_n \leq P_n - P_{nk} + |\mathcal{I}| w_{nk} && \forall k\in\mathcal{K}, n\in\mathcal{L} && (12.3)\\
& P_{nk} = \sum_{\substack{i\in\mathcal{I}:\\ y^i=k}} \zeta^i_{a(n),n} && \forall k\in\mathcal{K}, n\in\mathcal{L} && (12.4)\\
& P_n = \sum_{i\in\mathcal{I}} \zeta^i_{a(n),n} && \forall n\in\mathcal{L} && (12.5)\\
& l_n = \sum_{k\in\mathcal{K}} w_{nk} && \forall n\in\mathcal{L} && (12.6)\\
& \zeta^i_{a(n),n} \leq l_n && \forall n\in\mathcal{L} && (12.7)\\
& \sum_{n\in\mathcal{L}} \zeta^i_{a(n),n} = 1 && \forall i\in\mathcal{I} && (12.8)\\
& \sum_{f\in\mathcal{F}} b_{mf} x^i_f \geq v_m + \zeta^i_{a(n),n} - 1 && \forall i\in\mathcal{I}, n\in\mathcal{L}, m\in\mathcal{AR}(n) && (12.9)\\
& \sum_{f\in\mathcal{F}} b_{mf} x^i_f \leq v_m - 2\zeta^i_{a(n),n} + 1 && \forall i\in\mathcal{I}, n\in\mathcal{L}, m\in\mathcal{AL}(n) && (12.10)\\
& \sum_{f\in\mathcal{F}} b_{nf} = d_n && \forall n\in\mathcal{N} && (12.11)\\
& 0 \leq v_n \leq d_n && \forall n\in\mathcal{N} && (12.12)\\
& d_n \leq d_{a(n)} && \forall n\in\mathcal{N}\backslash\{1\} && (12.13)\\
& z^i_n, l_n \in \{0,1\} && \forall i\in\mathcal{I}, n\in\mathcal{L} && (12.14)\\
& b_{nf}, d_n \in \{0,1\} && \forall f\in\mathcal{F}, n\in\mathcal{N}, && (12.15)
\end{aligned}
$$

where $\lambda \in [0,1]$ is a regularization term. The objective (12.1) maximizes the total number of correctly classified datapoints $|\mathcal{I}| - \sum_{n\in\mathcal{L}} L_n$ while minimizing the number of splits $\sum_{n\in\mathcal{N}} d_n$. Constraints (12.2) and (12.3) defines the number of missclassified datapoints at each node n. Constraints (12.4) and (12.5) give the definitions of P_{nk} and P_n respectively. constraints (12.6)-(12.7), enforce that if a leaf n does not have an assigned class label, no datapoint should end up at that leaf. Constraint (12.8) makes sure that each datapoint i is assigned to exactly one of the leaf nodes. Constraint (12.9) implies that if datapoint i is assigned to node n, it should take the right branch for all ancestors of n belonging to $\mathcal{AR}(n)$. Respectively, constraint (12.10) implies that if datapoint i is assigned to node n, it should take the left branch for all ancestors of n belonging to $\mathcal{AL}(n)$. Constraint (12.11) enforces that if node n splits, it should split on exatcly one of the

features $f \in \mathcal{F}$. Constraint (12.12) implies that if a node does not apply a split, all datapoints going through this node would take the right branch. At the end constraint (12.13) makes sure that if node n does not split, none of its descendants cannot split.

In the main formulation of Bertsimas and Dunn (2017), they have parameter $N_{\min}$ which denotes the minimum number of points at each leaf. We set this parameter to zero as we do not have a similar notion in our formulation.

C. Comparison with OCT

In formulation (12), v_n can be fixed to d_n for all nodes (for the case of binary data) leading to the simplified formulation

$$
\begin{aligned}
\max \quad & (1-\lambda)\left(|\mathcal{I}| - \sum_{n\in\mathcal{L}} L_n\right) - \lambda \sum_{n\in\mathcal{N}} d_n \\
\text{s.t.} \quad & L_n \geq P_n - P_{nk} - |\mathcal{I}|(1-w_{nk}) && \forall k \in \mathcal{K}, n \in \mathcal{L} \\
& L_n \leq P_n - P_{nk} + |\mathcal{I}| w_{nk} && \forall k \in \mathcal{K}, n \in \mathcal{L} \\
& P_{nk} = \sum_{\substack{i\in\mathcal{I}:\\ y^i=k}} \zeta^i_{a(n),n} && \forall k \in \mathcal{K}, n \in \mathcal{L} \\
& P_n = \sum_{i\in\mathcal{I}} \zeta^i_{a(n),n} && \forall n \in \mathcal{L} \\
& l_n = \sum_{k\in\mathcal{K}} w_{nk} && \forall n \in \mathcal{L} \\
& \zeta^i_{a(n),n} \leq l_n && \forall n \in \mathcal{L} \\
& \sum_{n\in\mathcal{L}} \zeta^i_{a(n),n} = 1 && \forall i \in \mathcal{I} \\
& \sum_{f\in\mathcal{F}:x^i_f=1} b_{mf} \geq d_m + \zeta^i_{a(n),n} - 1 && \forall i \in \mathcal{I}, n \in \mathcal{L}, m \in \mathcal{AR}(n) \\
& \sum_{f\in\mathcal{F}:x^i_f=1} b_{mf} \leq d_m - 2\zeta^i_{a(n),n} + 1 && \forall i \in \mathcal{I}, n \in \mathcal{L}, m \in \mathcal{AL}(n) \\
& \sum_{f\in\mathcal{F}} b_{nf} = d_n && \forall n \in \mathcal{N} \\
& d_n \leq d_{a(n)} && \forall n \in \mathcal{N}\backslash\{1\} \\
& z^i_n, l_n \in \{0,1\} && \forall i \in \mathcal{I}, n \in \mathcal{L} \\
& b_{nf}, d_n \in \{0,1\} && \forall f \in \mathcal{F}, n \in \mathcal{N},
\end{aligned}
$$

Note that fixing v_n is a simplification due to the assumption of binary data, rather than an actual strengthening. Moreover, note that OCT and FlowOCT have different conventions for nodes where

branching is not performed: in FlowOCT, a label (encoded by w_{nk}) is directly assigned to that node, while in OCT all points go right by convention. This different convention creates a slight change in the feasible region of both formulations. To be able to directly compare the formulations, we consider the case of "full" trees where branching is performed at all internal nodes $\mathcal{N}$. For FlowOCT formulation, this corresponds to setting $w_{nk} = 0$ for all $n \in \mathcal{N}$, $k \in \mathcal{K}$, while for OCT it corresponds to setting $d_n = 1$ for all $n \in \mathcal{S}$. Moreover, using the identity $\sum_{f \in \mathcal{F}: x_f^i = 1} b_{mf} = 1 - \sum_{f \in \mathcal{F}: x_f^i = 0} b_{mf}$ and and noting that l_n can be fixed to 1 in the formulation, we obtain the simplified OCT formulation

$$
\begin{aligned}
\max \quad & \left(|\mathcal{I}| - \sum_{n \in \mathcal{L}} L_n \right) && && (13.1)\\
\text{s.t.} \quad & L_n \geq P_n - P_{nk} - |\mathcal{I}|(1 - w_{nk}) && \forall k \in \mathcal{K}, n \in \mathcal{L} && (13.2)\\
& L_n \leq P_n - P_{nk} + |\mathcal{I}| w_{nk} && \forall k \in \mathcal{K}, n \in \mathcal{L} && (13.3)\\
& P_{nk} = \sum_{\substack{i \in \mathcal{I}: \\ y^i = k}} \zeta^i_{a(n),n} && \forall k \in \mathcal{K}, n \in \mathcal{L} && (13.4)\\
& P_n = \sum_{i \in \mathcal{I}} \zeta^i_{a(n),n} && \forall n \in \mathcal{L} && (13.5)\\
& \sum_{k \in \mathcal{K}} w_{nk} = 1 && \forall n \in \mathcal{L} && (13.6)\\
& \sum_{n \in \mathcal{L}} \zeta^i_{a(n),n} = 1 && \forall i \in \mathcal{I} && (13.7)\\
& \sum_{f \in \mathcal{F}: x_f^i = 1} b_{mf} \geq \zeta^i_{a(n),n} && \forall i \in \mathcal{I}, n \in \mathcal{L}, m \in \mathcal{AR}(n) && (13.8)\\
& \sum_{f \in \mathcal{F}: x_f^i = 0} b_{mf} \geq 2\zeta^i_{a(n),n} - 1 && \forall i \in \mathcal{I}, n \in \mathcal{L}, m \in \mathcal{AL}(n) && (13.9)\\
& \sum_{f \in \mathcal{F}} b_{nf} = 1 && \forall n \in \mathcal{N} && (13.10)\\
& \zeta^i_{a(n),n} \in \{0, 1\} && \forall i \in \mathcal{I}, n \in \mathcal{L} && (13.11)\\
& b_{nf} \in \{0, 1\} && \forall f \in \mathcal{F}, n \in \mathcal{N}. && (13.12)
\end{aligned}
$$

C.1. Strengthening

We now show how formulation (13) can be strengthened. Observe that the validity of the steps below is guaranteed by the validity of FlowOCT, thus we do not focus on validity below.

Bound tightening for (13.9) Adding the quantity $1 - \zeta^i_{a(n),n} \geq 0$ to the right hand side of (13.9), we obtain the stronger constraints

$$\sum_{f \in \mathcal{F}: x^i_f = 0} b_{mf} \geq \zeta^i_{a(n),n} \; \forall i \in \mathcal{I}, n \in \mathcal{L}, m \in \mathcal{AL}(n). \tag{14}$$

Improved branching constraints Constraints (13.8) can be strengthened to

$$\sum_{f \in \mathcal{F}: x^i_f = 1} b_{mf} \geq \sum_{n \in \mathcal{L}: m \in \mathcal{AR}(n)} \zeta^i_{a(n),n} \; \forall i \in \mathcal{I}, m \in \mathcal{N}. \tag{15}$$

Observe that constraints (15), in addition to being stronger than (13.8), also reduce the number of constraints require to represent the LP relaxation. Similarly, constraint (14) can be further improved to

$$\sum_{f \in \mathcal{F}: x^i_f = 0} b_{mf} \geq \sum_{n \in \mathcal{L}: m \in \mathcal{AL}(n)} \zeta^i_{a(n),n} \; \forall i \in \mathcal{I}, m \in \mathcal{N}. \tag{16}$$

Improved missclassification formulation Define for all $i \in \mathcal{I}$ and $n \in \mathcal{L}$ additional variables $z^i_{a(n),n} \leq \zeta_{a(n),n} w_{n,y^i}$. Note that $z^i_{a(n),n} = 1$ implies that datapoint i is routed to leaf n ($\zeta^i_{a(n),n} = 1$) and the class of i is assigned to n ($w_{ny^i} = 1$), hence $z^i_{a(n),n} = 1$ only if i is correctly classified at leaf n. Upper bounds of $z^i_{a(n),n} = 1$ can be imposed via the linear constraints

$$z^i_{a(n),n} \leq \zeta^i_{a(n),n}, \; z^i_{a(n),n} \leq w_{ny^i} \quad \forall n \in \mathcal{L}, i \in \mathcal{I}. \tag{17}$$

In addition, since L_n corresponds to the number of missclassified points at leaf $n \in \mathcal{L}$ and $\sum_{n \in \mathcal{L}} L_n$, we find that constraints

$$L_n \geq \sum_{i \in \mathcal{I}} (\zeta^i_{n,a(n)} - z^i_{n,a(n)}). \tag{18}$$

Note that constraints (18) and (13.7) imply that

$$\sum_{n \in \mathcal{L}} L_n \geq |\mathcal{I}| - \sum_{i \in \mathcal{I}} \sum_{n \in \mathcal{L}} z^i_{n,a(n)}. \tag{19}$$

C.2. Simplification

The linear programming relaxation of the formulation obtained in §C.1, given by constraints (13.2)-(13.7), (13.10)-(13.12), (15), (16), (17) and (18), is certainly stronger than the relaxation of OCT, as either constraints where tightened or additional constraints were added. We now show how the resulting formulation can be simplified without loss of relaxation quality, ultimately obtaining FlowOCT.

Upper bound on missclassification Variable L_n has a negative objective coefficient and only appears on constraints (13.2)-(13.3) and (18), it will always be set to a lower bound. Therefore, constraint (13.3) is redundant and can be dropped without affecting the relaxation of the problem.

Lower bound on missclassification Substituting variables according to (13.4) and (13.5), we find that for a given $k \in \mathcal{K}$ and $n \in \mathcal{L}$, (13.2) is equivalent to

$$\begin{aligned} L_n &\geq \sum_{i \in \mathcal{I}} \zeta^i_{a(n),n} - \sum_{\substack{i \in \mathcal{I}: \\ y^i = k}} \zeta^i_{a(n),n} - |\mathcal{I}|(1 - w_{nk}) \\ \Leftrightarrow L_n &\geq \sum_{\substack{i \in \mathcal{I} \\ y_i = k}} (w_{nk} - 1) + \sum_{\substack{i \in \mathcal{I} \\ y^i \neq k}} (\zeta^i_{a(n),n} - 1 + w_{nk}). \end{aligned}$$

Observe that $w_{nk} - 1 \leq 0 \leq \zeta^i_{a(n),n} - z^i_{a(n),n}$. Moreover, we also have that for any $i \in \mathcal{I}$ and $k \in \mathcal{K} \setminus \{y^i\}$,

$$z^i_{a(n),n} \leq w_{ny^i} \leq 1 - w_{nk}, \tag{20}$$

where the first inequality follows from (17) and the second inequality follows from (13.6). Therefore from (20) we conclude that $\zeta^i_{a(n),n} - 1 + w_{nk} \leq \zeta^i_{a(n),n} - z^i_{a(n),n}$ and inequalities (18) dominate inequalities (13.2). Since inequalities (13.4)-(13.5) only appeared in inequalities (13.2)-(13.3), which where shown to be redundant, they can be dropped as well. Finally, as inequalities (18) define the unique the lower bounds of L_n in the simplified formulation, they can be changed to equalities without loss of generalities, and the objective can be updated according to (19). After all the

changes outlined so far, the formulation reduces to

$$\max \sum_{i\in\mathcal{I}}\sum_{n\in\mathcal{L}} z^i_{n,a(n)} \tag{21.1}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}} w_{nk} = 1 \qquad \forall n \in \mathcal{L} \tag{21.2}$$

$$\sum_{n\in\mathcal{L}} \zeta^i_{a(n),n} = 1 \qquad \forall i \in \mathcal{I} \tag{21.3}$$

$$\sum_{f\in\mathcal{F}:x^i_f=1} b_{mf} \geq \sum_{n\in\mathcal{L}:m\in\mathcal{AR}(n)} \zeta^i_{a(n),n} \qquad \forall i \in \mathcal{I}, m \in \mathcal{N} \tag{21.4}$$

$$\sum_{f\in\mathcal{F}:x^i_f=0} b_{mf} \geq \sum_{n\in\mathcal{L}:m\in\mathcal{AL}(n)} \zeta^i_{a(n),n} \qquad \forall i \in \mathcal{I}, m \in \mathcal{N} \tag{21.5}$$

$$\sum_{f\in\mathcal{F}} b_{nf} = 1 \qquad \forall n \in \mathcal{N} \tag{21.6}$$

$$z^i_{a(n),n} \leq \zeta^i_{a(n),n} \qquad \forall n \in \mathcal{L}, i \in \mathcal{I} \tag{21.7}$$

$$z^i_{a(n),n} \leq w_{ny^i} \qquad \forall n \in \mathcal{L}, i \in \mathcal{I} \tag{21.8}$$

$$\zeta^i_{a(n),n} \in \{0,1\} \qquad \forall i \in \mathcal{I}, n \in \mathcal{L} \tag{21.9}$$

$$b_{nf} \in \{0,1\} \qquad \forall f \in \mathcal{F}, n \in \mathcal{N}. \tag{21.10}$$

C.3. Projection

We now project out the ζ variables, obtaining a more compact formulation with the same LP relaxation. Specifically, consider the formulation

$$\max \sum_{i\in\mathcal{I}}\sum_{n\in\mathcal{L}} z^i_{n,a(n)} \tag{22.1}$$

$$\text{s.t.} \sum_{k\in\mathcal{K}} w_{nk} = 1 \qquad \forall n \in \mathcal{L} \tag{22.2}$$

$$\sum_{f\in\mathcal{F}:x^i_f=1} b_{mf} \geq \sum_{n\in\mathcal{L}:m\in\mathcal{AR}(n)} z^i_{a(n),n} \qquad \forall i \in \mathcal{I}, m \in \mathcal{N} \tag{22.3}$$

$$\sum_{f\in\mathcal{F}:x^i_f=0} b_{mf} \geq \sum_{n\in\mathcal{L}:m\in\mathcal{AL}(n)} z^i_{a(n),n} \qquad \forall i \in \mathcal{I}, m \in \mathcal{N} \tag{22.4}$$

$$\sum_{f\in\mathcal{F}} b_{nf} = 1 \qquad \forall n \in \mathcal{N} \tag{22.5}$$

$$z^i_{a(n),n} \leq w_{ny^i} \qquad \forall n \in \mathcal{L}, i \in \mathcal{I} \tag{22.6}$$

$$z^i_{a(n),n} \in \{0,1\} \qquad \forall i \in \mathcal{I}, n \in \mathcal{L} \tag{22.7}$$

$$b_{nf} \in \{0,1\} \qquad \forall f \in \mathcal{F}, n \in \mathcal{N}.. \tag{22.8}$$

Proposition 3. *Formulations* (21) *and* (22) *are equivalent, i.e., their LP relaxations have the same*

optimal objective value.

Proof. Let ν_1 and ν_2 be the optimal objective value of the LP relaxations of (21) and (22). Note that (22) is a relaxation of (21), obtained by dropping constraint (21.3) and replacing ζ with a lower bound in constraints (21.4)-(21.5). Therefore, it follows that $\nu_2 \geq \nu_1$. We now show that $\nu_2 \leq \nu_1$.

Let (b^*, w^*, z^*) be an optimal solution of (22) and let $i \in \mathcal{I}$. For any given $i \in \mathcal{I}$, by summing constraints (22.3) and (22.4) for the root node $m = 1$, we find that

$$1 = \sum_{f \in \mathcal{F}: x_f^i = 1} b_{1f}^* + \sum_{f \in \mathcal{F}: x_f^i = 0} b_{1f}^* \geq \sum_{n \in \mathcal{L}} (z_{a(n),n}^i)^*. \tag{23}$$

Now let $\zeta = z^*$. If the inequality in (23) holds at equality, then (b^*, w^*, z^*, ζ) satisfies all constraints in (21) and the proof is complete. Otherwise, it follows that either (22.3) or (22.4) is strict at node $m = 1$, and without loss of generality assume (22.3) is strict. Summing up inequalities (22.3) and (22.4) for node $m = r(1)$, we find that

$$1 = \sum_{f \in \mathcal{F}} b_{r(1)f}^* > \sum_{n \in \mathcal{L}: r(1) \in \mathcal{AR}(n) \cup \mathcal{AL}(n)} (z_{a(n),n}^i)^*, \tag{24}$$

where the strict inequality holds since the right hand side of (24) is no greater than the right hand side of (23). By applying this process recursively, we obtain a path from node 1 to a leaf $h \in \mathcal{L}$ such that all inequalities (22.3)-(22.4) corresponding to this path are strict. The value $\zeta_{a(h),h}^i$ can be then increased by the minimum slack in the constraints, and the overall process can be repeated until inequality (21.3) is tight. ■

C.4. Substitution

Finally, to recover the FlowOCT formulation, for all $m \in \mathcal{L}$, substitute variables

$$z_{m,r(m)}^i := \sum_{n \in \mathcal{L}: m \in \mathcal{AR}(n)} z_{a(n),n}^i, \text{ and}$$
$$z_{m,\ell(m)}^i := \sum_{n \in \mathcal{L}: m \in \mathcal{AL}(n)} z_{a(n),n}^i.$$

and for all $n \in \mathcal{L}$ introduce variables $z_{n,t} = z_{a(n),n}$. Constraints (22.3)-(22.4) reduce to $\sum_{f \in \mathcal{F}: x_f^i = 1} b_{mf} \geq z_{m,r(m)}^i$ and $\sum_{f \in \mathcal{F}: x_f^i = 0} b_{mf} \geq z_{m,\ell(m)}^i$. Finally, since

$$\begin{aligned} z_{a(m),m} &= \sum_{n \in \mathcal{L}: m \in \mathcal{AR}(n) \cup \mathcal{AL}(n)} z_{a(n),n} \\ &= \sum_{n \in \mathcal{L}: m \in \mathcal{AR}(n)} z_{a(n),n} + \sum_{n \in \mathcal{L}: m \in \mathcal{AL}(n)} z_{a(n),n} \\ &= z_{m,r(m)} + z_{m,\ell(m)}, \end{aligned}$$

we recover the flow conservation constraints.

D. Extended Results

In Table 4, for each dataset and depth, we show the in sample results for each approach. In this table, for $\lambda = 0$, we average the in sample results including the training accuracy, optimality gap and solving time across five different samples trained over 50% of the data when λ is fixed to be *zero*. Out of 32 instances, `OCT` has the best training accuracy in 0 instances (excluding ties) and `BinOCT` in 7 instances while `FlowOCT` and `Benders` have the best accuracy in 11 instances. In terms of solving time, `BinOCT` achieves a smaller solving time in 7 instances while `Benders` achieves a smaller solving time in 13 instances (excluding ties). In terms of optimality gap, `OCT` achieves a smaller gap time only in one of the instances while `Benders` achieves a smaller gap time in 15 instances (excluding ties).

Similarly for $\lambda > 0$ we show similar results but this time for a given instance and a $\lambda \in [0.1, 0.9]$ with step size of 0.1 we solve 5 different samples and report the average results across all 45 samples. As `BinOCT` does not have any regularization term, we have excluded it from this section. We observe that `Benders` outperform `OCT` in both optimality gap and solving time for all instances.

Table 4: In sample results

Dataset	Depth	$\lambda = 0$												$\lambda > 0$					
		OCT			BinOCT			FlowOCT			Benders			OCT		FlowOCT		Benders	
		Train-acc	Gap	Time	Train-acc	Gap	Time	Train-acc	Gap	Time	Train-acc	Gap	Time	Gap	Time	Gap	Time	Gap	Time
monk3	2	**94.0**	**0.0**	2.1	**94.0**	**0.0**	**0.3**	**94.0**	**0.0**	6.7	**94.0**	**0.0**	0.9	**0.0**	2.8	**0.0**	5.9	**0.0**	**0.8**
monk3	3	**98.0**	**0.0**	283.6	**98.0**	**0.0**	32.2	**98.0**	**0.0**	89.1	**98.0**	**0.0**	**16.1**	**0.0**	313.7	**0.0**	23.7	**0.0**	**4.0**
monk3	4	**100.0**	**0.0**	391.2	**100.0**	**0.0**	654.3	**100.0**	**0.0**	141.6	**100.0**	**0.0**	**73.3**	0.5	1197.3	**0.0**	244.1	**0.0**	**31.0**
monk3	5	**100.0**	**0.0**	203.8	**100.0**	**0.0**	156.6	**100.0**	**0.0**	117.7	**100.0**	**0.0**	**5.9**	0.7	1661.3	**0.0**	262.5	**0.0**	**35.2**
monk1	2	**85.8**	**0.0**	2.9	**85.8**	**0.0**	1.5	**85.8**	**0.0**	8.0	**85.8**	**0.0**	**1.2**	**0.0**	4.0	**0.0**	8.6	**0.0**	**1.0**
monk1	3	**95.5**	**0.0**	672.1	**95.5**	**0.0**	44.8	**95.5**	**0.0**	45.7	**95.5**	**0.0**	**4.8**	0.3	1096.3	**0.0**	25.6	**0.0**	**4.3**
monk1	4	**100.0**	**0.0**	49.5	**100.0**	**0.0**	61.9	**100.0**	**0.0**	69.0	**100.0**	**0.0**	**5.3**	0.3	1042.1	**0.0**	63.1	**0.0**	**6.2**
monk1	5	**100.0**	**0.0**	116.9	**100.0**	**0.0**	5.3	**100.0**	**0.0**	214.3	**100.0**	**0.0**	**2.7**	3.3	2607.3	**0.0**	154.5	**0.0**	**8.4**
monk2	2	**71.4**	**0.0**	13.3	**71.4**	**0.0**	**5.7**	**71.4**	**0.0**	12.8	**71.4**	**0.0**	22.1	**0.0**	15.6	**0.0**	15.4	**0.0**	**3.5**
monk2	3	81.0	22.1	3602.9	80.5	99.8	3600.0	**81.2**	**0.0**	1106.6	**81.2**	**0.0**	**693.0**	18.5	3076.1	**0.0**	619.7	**0.0**	**373.7**
monk2	4	88.3	13.4	3607.3	86.7	100.0	**3600.0**	**89.5**	8.4	3602.5	**89.5**	**8.0**	**3600.0**	27.4	3211.1	11.0	2886.9	**9.2**	**2818.2**
monk2	5	92.6	8.1	3617.9	94.8	100.0	3600.0	93.6	7.0	3605.5	**96.9**	**3.2**	**3505.4**	31.0	3224.2	14.4	2987.8	**13.1**	**2829.3**
house	2	**97.1**	**0.0**	5.8	**97.1**	**0.0**	**0.6**	**97.1**	**0.0**	12.9	**97.1**	**0.0**	1.5	**0.0**	4.3	**0.0**	8.0	**0.0**	**1.0**
house	3	**99.0**	**0.0**	298.1	**99.0**	**0.0**	180.4	**99.0**	**0.0**	192.9	**99.0**	**0.0**	**67.1**	**0.0**	407.5	**0.0**	76.5	**0.0**	**10.9**
house	4	**100.0**	**0.0**	92.6	99.8	20.0	1105.6	**100.0**	**0.0**	276.9	**100.0**	**0.0**	**13.4**	**0.0**	788.3	**0.0**	164.6	**0.0**	**22.2**
house	5	**100.0**	**0.0**	83.1	**100.0**	**0.0**	454.6	**100.0**	**0.0**	206.9	**100.0**	**0.0**	**18.8**	0.2	1334.0	**0.0**	310.7	**0.0**	**25.7**
balance	2	70.2	**0.0**	159.4	70.2	**0.0**	**6.8**	**70.3**	**0.0**	49.5	70.2	**0.0**	7.0	**0.0**	142.2	**0.0**	92.1	**0.0**	**6.4**
balance	3	75.0	33.6	3613.4	76.5	95.3	3600.0	**76.6**	0.5	3320.8	**76.6**	**0.0**	**548.4**	37.8	3613.4	0.1	2342.4	**0.0**	**359.1**
balance	4	75.5	32.7	3635.1	78.8	100.0	**3600.0**	76.4	30.5	3611.3	**79.7**	**9.7**	**3600.0**	43.2	3635.0	22.1	3611.3	**10.1**	**3340.2**
balance	5	71.9	39.1	3686.4	81.7	100.0	**3600.0**	78.8	27.0	3623.0	**82.9**	**19.9**	**3600.0**	52.3	3689.8	29.3	3622.8	**24.4**	**3600.1**
tic-tac-toe	2	72.7	1.4	2629.8	72.7	**0.0**	**89.7**	**72.8**	**0.0**	1198.7	72.7	**0.0**	118.3	2.8	2365.7	**0.0**	1193.7	**0.0**	**115.8**
tic-tac-toe	3	77.5	29.1	3626.6	**78.8**	100.0	**3600.0**	76.0	31.5	3610.6	78.7	**18.8**	**3600.0**	33.6	3626.5	29.1	3610.6	**18.4**	**3600.2**
tic-tac-toe	4	74.9	33.6	3670.0	**85.0**	100.0	**3600.0**	79.5	26.0	3623.1	83.0	**20.5**	**3600.0**	38.9	3670.0	27.2	3621.7	**24.8**	**3600.2**
tic-tac-toe	5	71.4	40.1	3774.4	**88.2**	100.0	**3600.0**	70.4	42.2	3769.1	**88.2**	**13.4**	**3600.0**	41.4	3778.4	35.1	3650.6	**24.9**	**3600.2**
car_eval	2	**78.5**	**0.0**	1047.8	**78.5**	0.0	**27.7**	76.9	7.7	1214.8	**78.5**	**0.0**	66.4	**0.0**	1610.7	0.8	679.6	**0.0**	**55.1**
car_eval	3	77.2	29.9	3634.7	**81.9**	100.0	**3600.0**	79.4	26.2	3614.9	81.7	**9.3**	**3600.0**	36.0	3634.8	22.2	3614.9	**6.7**	**3571.2**
car_eval	4	76.5	31.0	3692.3	**83.9**	100.0	**3600.0**	71.9	39.1	3630.8	83.6	**19.6**	**3600.0**	37.9	3691.9	28.2	3630.6	**20.1**	**3600.1**
car_eval	5	72.1	39.1	3828.6	85.0	100.0	**3600.0**	71.6	39.9	3742.6	**85.2**	**17.3**	**3600.0**	37.3	3828.4	162072.3	3753.2	**18.9**	**3600.1**
kr-vs-kp	2	84.7	18.0	3641.1	**86.9**	**0.0**	**102.6**	82.1	14.3	3481.7	**86.9**	**0.0**	671.2	24.4	3641.2	18.4	3544.5	**0.0**	**652.1**
kr-vs-kp	3	73.8	35.6	3725.7	**92.6**	100.0	**3600.0**	61.1	65.0	3701.0	90.4	**10.7**	**3600.0**	40.1	3721.9	49.2	3671.3	**11.7**	**3600.2**
kr-vs-kp	4	68.1	47.2	3923.1	**92.5**	100.0	**3600.0**	63.8	58.1	4560.5	89.8	**11.8**	**3600.0**	49.6	3923.7	419947.6	4479.5	**9.7**	**3600.2**
kr-vs-kp	5	66.6	50.9	4398.2	**93.6**	100.0	**3600.0**	51.5	10000	3790.7	90.5	**10.7**	**3600.0**	50.5	4400.7	677022.8	4789.0	**12.5**	**3600.3**