Learning to Branch[*]

Maria-Florina Balcan   Travis Dick   Tuomas Sandholm   Ellen Vitercik

May 18, 2018

Abstract

Tree search algorithms, such as branch-and-bound, are the most widely used tools for solving combinatorial and nonconvex problems. For example, they are the foremost method for solving (mixed) integer programs and constraint satisfaction problems. Tree search algorithms recursively partition the search space to find an optimal solution. In order to keep the tree size small, it is crucial to carefully decide, when expanding a tree node, which question (typically variable) to branch on at that node in order to partition the remaining space. Numerous partitioning techniques (e.g., variable selection) have been proposed, but there is no theory describing which technique is optimal. We show how to use machine learning to determine an optimal weighting of any set of partitioning procedures for the instance distribution at hand using samples from the distribution. We provide the first sample complexity guarantees for tree search algorithm configuration. These guarantees bound the number of samples sufficient to ensure that the empirical performance of an algorithm over the samples nearly matches its expected performance on the unknown instance distribution. This thorough theoretical investigation naturally gives rise to our learning algorithm. Via experiments, we show that learning an optimal weighting of partitioning procedures can dramatically reduce tree size, and we prove that this reduction can even be exponential. Through theory and experiments, we show that learning to branch is both practical and hugely beneficial.

1 Introduction

Many widely-used algorithms are customizable: they have tunable parameters that have an enormous effect on runtime, solution quality, or both. Tuning parameters by hand is notoriously tedious and time-consuming. In this work, we study algorithm configuration via machine learning, where the goal is to design algorithms that learn the optimal parameter setting for the problem instance distribution at hand.

We study configuration of tree search algorithms. These algorithms are the most widely used tools for solving combinatorial and nonconvex problems throughout artificial intelligence, operations research, and beyond (e.g., [Russell and Norvig, 2010, Williams, 2013]). For example, branch-and-bound (B&B) algorithms [Land and Doig, 1960] solve mixed integer linear programs (MILPs), and thus have diverse applications, including ones in machine learning such as MAP estimation [Kappes et al., 2013], object recognition [Kokkinos, 2011], clustering [Komodakis et al., 2009], and semi-supervised SVMs [Chapelle et al., 2007].

A tree search algorithm systematically partitions the search space to find an optimal solution. The algorithm organizes this partition via a tree: the original problem is at the root and the children of a given node represent the subproblems formed by partitioning the feasible set of the

---

[*]Authors' addresses: {`ninamf, tdick, sandholm, vitercik`}`@cs.cmu.edu`.

parent node. A branch is pruned if it is infeasible or it cannot produce a better solution than the best one found so far by the algorithm. Typically the search space is partitioned by adding an additional constraint on some variable. For example, suppose the feasible set is defined by the constraint $A\boldsymbol{x} \leq \boldsymbol{b}$, with $\boldsymbol{x} \in \{0,1\}^n$. A tree search algorithm might partition this feasible set into two sets, one where $A\boldsymbol{x} \leq \boldsymbol{b}$, $x_1 = 0$, and $x_2, \ldots, x_n \in \{0,1\}$, and another where $A\boldsymbol{x} \leq \boldsymbol{b}$, $x_1 = 1$, and $x_2, \ldots, x_n \in \{0,1\}$, in which case the algorithm has *branched on* x_1. A crucial question in tree search algorithm design is determining which variable to branch on at each step. An effective variable selection policy can have a tremendous effect on the size of the tree. Currently, there is no known optimal strategy and the vast majority of existing techniques are backed only by empirical comparisons. In the worst-case, finding an approximately optimal branching variable, even at the root of the tree alone, is NP-hard. This is true even in the case of satisfiability, which is a special case of constraint satisfaction and of MILP [Liberatore, 2000].

In this work, rather than attempt to characterize a branching strategy that is universally optimal, we show empirically and theoretically that it is possible to learn high-performing branching strategies for a given application domain. We model an application domain as a distribution over problem instances, such as a distribution over scheduling problems that an airline solves on a day-to-day basis. This model is standard throughout the algorithm configuration literature (e.g., [Hutter et al., 2009, 2011, Dai et al., 2017, Kleinberg et al., 2017]). The approach has also been used on large-scale problems in industry to configure and select winner determination algorithms for clearing tens of billions of dollars of combinatorial auctions [Sandholm, 2013]. The algorithm designer does not know the underlying distribution over problem instances, but has sample access to the distribution. We show how to use samples from the distribution to learn a variable selection policy that will result in as small a search tree as possible in expectation over the underlying distribution.

Our learning algorithm adaptively partitions the parameter space of the variable selection policy into regions where for any parameter in a given region, the resulting tree sizes across the training set are invariant. The learning algorithm returns the empirically optimal parameter over the training set, and thus performs empirical risk minimization (ERM). We prove that the adaptive nature of our algorithm is necessary: performing ERM over a data-independent discretization of the parameter space can be disastrous. In particular, for any discretization of the parameter space, we provide an infinite family of distributions over MILP instances such that every point in the discretization results in a B&B tree with exponential size in expectation, but there exist infinitely-many parameters outside of the discretized points that result in a tree with constant size with probability 1. A small change in parameters can thus cause a drastic change in the algorithm's behavior. This fact contradicts conventional wisdom. For example, SCIP, the best open-source MILP solver, sets one of the parameters we investigate to 5/6, regardless of the input MILP's structure. Achterberg [2009] wrote that 5/6 was empirically optimal when compared against four other data-independent values. In contrast, our analysis shows that a data-driven approach to parameter tuning can have an enormous benefit.

The sensitivity of tree search algorithms to small changes in their parameters is a key challenge that differentiates our sample complexity analysis from those typically found in machine learning. For many well-understood function classes in machine learning, there is a close connection between the distance in parameter space between two parameter vectors and the distance in function space between the two corresponding functions. Understanding this connection is a necessary prerequisite to analyzing how many significantly different functions there are in the class, and thereby quantifying the class's intrinsic complexity. Intrinsic complexity typically translates to VC dimension, Rademacher complexity, or some other metric which allows us to derive learnability guarantees. Since the tree size of a search algorithm as a function of its parameters does not exhibit this predictable behavior, we must carefully analyze the way in which the parameters influence each step of

the procedure in order to derive learning algorithms with strong guarantees. In doing so, we present the first sample complexity guarantees for automated configuration of tree search algorithms. We provide worst-case bounds proving that a surprisingly small number of samples are sufficient for strong learnability guarantees: the sample complexity bound grows quadratically in the size of the problem instance, despite the complexity of the algorithms we study.

In our experiments section, we show that on many datasets based on real-world NP-hard problems, different parameters can result in B&B trees of vastly different sizes. Using an optimal parameter for one distribution on problems from a different distribution can lead to a dramatic tree size blowup. We also provide data-dependent generalization guarantees that allow the algorithm designer to use far fewer samples than in the worst case if the data is well-structured.

1.1 Related work

Several works have studied the use of machine learning techniques in the context of B&B; for an overview, see the summary by Lodi and Zarpellon [2017].

As in this work, Khalil et al. [2016] study variable selection policies. Their goal is to find a variable selection strategy that mimics the behavior of the classic branching strategy known as *strong branching* while running faster than strong branching. Alvarez et al. [2017] study a similar problem, although in their work, the feature vectors in the training set describe nodes from multiple MILP instances. Neither or these works come with any theoretical guarantees, unlike our work.

Several other works study data-driven variable selection from a purely experimental perspective. Di Liberto et al. [2016] devise an algorithm that learns how to dynamically switch between different branching heuristics along the branching tree. Karzan et al. [2009] propose techniques for choosing problem-specific branching rules based on a partial B&B tree. Ideally, these branching rules will choose variables that will lead to fast fathoming. They do not rely on any techniques from machine learning. In the context of CSP tree search, Xia and Yap [2018] apply existing multi-armed bandit algorithms to learning variable selection policies during tree search and Balafrej et al. [2015] use a bandit approach to select different levels of propagation during search.

Other works have explored the use of machine learning techniques in the context of other aspects of B&B beyond variable selection. For example, He et al. [2014] use machine learning to speed up branch-and-bound, focusing on speeding up the node selection policy. Their work does not provide any learning-theoretic guarantees. Other works that have studied machine learning techniques for branch-and-bound problems other than variable selection include Sabharwal et al. [2017], who also study how to devise node selection policies, Hutter et al. [2009], who study how to set CPLEX parameters, Kruber et al. [2017], who study how to detect decomposable model structure, and Khalil et al. [2017], who study how to determine when to run heuristics.

From a theoretical perspective, Le Bodic and Nemhauser [2017] present a theoretical model for the selection of branching variables. It is based upon an abstraction of MIPs to a simpler setting in which it is possible to analytically evaluate the dual bound improvement of choosing a given variable. Based on this model, they present a new variable selection policy which has strong performance on many MIPLIB instances. Unlike our work, this paper is unrelated to machine learning.

The learning-theoretic model of algorithm configuration that we study in this paper was introduced to the theoretical computer science community by Gupta and Roughgarden [2017]. Under this model, an application domain is modeled as a distribution over problem instances and the goal is to PAC-learn an algorithm that is nearly optimal over the distribution. This model was later studied by Balcan et al. [2017] as well. These papers were purely theoretical. In contrast, we show that the techniques proposed in this paper are practical as well, and provide significant benefit.

We provide a more detailed description of several of these papers in Appendix A.

2 Tree search

Tree search is a broad family of algorithms with diverse applications. To exemplify the specifics of tree search, we present a vast family of NP-hard problems — (mixed) integer linear programs — and describe how tree search finds optimal solutions to problems from this family. Later on in Section 5, we provide another example of tree search for constraint satisfaction problems. In Appendix E, we provide a formal, more abstract definition of tree search and generalize our results to this more general algorithm.

2.1 Mixed integer linear programs

We study *mixed integer linear programs* (MILPs) where the objective is to maximize $\boldsymbol{c}^\top \boldsymbol{x}$ subject to $A\boldsymbol{x} \leq \boldsymbol{b}$ and where some of the entries of $\boldsymbol{x}$ are constrained to be in $\{0, 1\}$. Given a MILP Q, we denote an optimal solution to the LP relaxation of Q as $\check{\boldsymbol{x}}_Q = (\check{x}_Q[1], \ldots \check{x}_Q[n])$. Throughout this work, given a vector $\boldsymbol{a}$, we use the notation $a[i]$ to denote the ith component of $\boldsymbol{a}$. We also use the notation $\check{c}_Q$ to denote the optimal objective value of the LP relaxation of Q. In other words, $\check{c}_Q = \boldsymbol{c}^\top \check{\boldsymbol{x}}_Q$.

Example 2.1 (Winner determination)**.** Suppose there is a set $\{1, \ldots, m\}$ of items for sale and a set $\{1, \ldots, n\}$ of buyers. In a combinatorial auction, each buyer i submits bids $v_i(b)$ for any number of bundles $b \subseteq \{1, \ldots, m\}$. The goal of the winner determination problem is to allocate the goods among the bidders so as to maximize *social welfare*, which is the sum of the buyers' values for the bundles they are allocated. We can model this problem as a MILP by assigning a binary variable $x_{i,b}$ for every buyer i and every bundle b they submit a bid $v_i(b)$ on. The variable $x_{i,b}$ is equal to 1 if and only if buyer i receives the bundle b. Let B_i be the set of all bundles b that buyer i submits a bid on. An allocation is feasible if it allocates no item more than once ($\sum_{i=1}^n \sum_{b \in B_i, j \ni b} x_{i,b} \leq 1$ for all $j \in \{1, \ldots, m\}$) and if each bidder receives at most one bundle ($\sum_{b \in B_i} x_{i,b} \leq 1$ for all $i \in \{1, \ldots, n\}$). Therefore, the MILP is:

$$
\begin{array}{ll}
\text{maximize} & \sum_{i=1}^n \sum_{b \in B_i} v_i(b) x_{i,b} \\
\text{s.t.} & \sum_{i=1}^n \sum_{b \in B_i, j \ni b} x_{i,b} \leq 1 \quad \forall j \in [m] \\
& \sum_{b \in B_i} x_{i,b} \leq 1 \quad\quad\quad\ \forall i \in [n] \\
& x_{i,b} \in \{0, 1\} \quad\quad\quad\ \forall i \in [n], b \in B_i.
\end{array}
$$

2.1.1 MILP tree search

MILPs are typically solved using a tree search algorithm called branch-and-bound (B&B). Given a MILP problem instance, B&B relies on two subroutines that efficiently compute upper and lower bounds on the optimal value within a given region of the search space. The lower bound can be found by choosing any feasible point in the region. An upper bound can be found via a linear programming relaxation. The basic idea of B&B is to partition the search space into convex sets and find upper and lower bounds on the optimal solution within each. The algorithm uses these bounds to form global upper and lower bounds, and if these are equal, the algorithm terminates, since the feasible solution corresponding to the global lower bound must be optimal. If the global upper and lower bounds are not equal, the algorithm refines the partition and repeats.

In more detail, suppose we want to use B&B to solve a MILP Q'. B&B iteratively builds a search tree $\mathcal{T}$ with the original MILP Q' at the root. In the first iteration, $\mathcal{T}$ consists of a single

---
Algorithm 1 Branch and bound

Input: A MILP instance Q'.

1: Let $\mathcal{T}$ be a tree that consists of a single node containing the MILP Q'.
2: Let $c^* = -\infty$ be the objective value of the best-known feasible solution.
3: **while** there remains an unfathomed leaf in $\mathcal{T}$ **do**
4:     Use a *node selection policy* to select a leaf of the tree $\mathcal{T}$, which corresponds to a MILP Q.
5:     Use a *variable selection policy* to choose a variable x_i of the MILP Q to branch on.
6:     Let Q_i^+ (resp., Q_i^-) be the MILP Q except with the constraint that $x_i = 1$ (resp., $x_i = 0$).
7:     Set the right (resp., left) child of Q in $\mathcal{T}$ to be a node containing the MILP Q_i^+ (resp., Q_i^-).
8:     **for** $\tilde{Q} \in \{Q_i^+, Q_i^-\}$ **do**
9:         **if** the LP relaxation of $\tilde{Q}$ is feasible **then**
10:             Let $\breve{\boldsymbol{x}}_{\tilde{Q}}$ be an optimal solution to the LP and let $\breve{c}_{\tilde{Q}}$ be its objective value.
11:             **if** the vector $\breve{\boldsymbol{x}}_{\tilde{Q}}$ satisfies the constraints of the original MILP Q' **then**
12:                 Fathom the leaf containing $\tilde{Q}$.
13:                 **if** $c^* < \breve{c}_{\tilde{Q}}$ **then**
14:                     Set $c^* = \breve{c}_{\tilde{Q}}$.
15:             **else if** $\breve{\boldsymbol{x}}_{\tilde{Q}}$ is no better than the best known feasible solution, i.e., $c^* \geq \breve{c}_{\tilde{Q}}$ **then**
16:                 Fathom the leaf containing $\tilde{Q}$.
17:         **else**
18:             Fathom the leaf containing $\tilde{Q}$.
---

node containing the MILP Q'. At each iteration, B&B uses a *node selection policy* (which we expand on later) to select a leaf node of the tree $\mathcal{T}$, which corresponds to a MILP Q. B&B then uses a *variable selection policy* (which we expand on in Section 2.1.2) to choose a variable x_i of the MILP Q to branch on. Specifically, let Q_i^+ (resp., Q_i^-) be the MILP Q except with the additional constraint that $x_i = 1$ (resp., $x_i = 0$). B&B sets the right (resp., left) child of Q in $\mathcal{T}$ to be a node containing the MILP Q_i^+ (resp., Q_i^-). B&B then tries to "fathom" these leafs: the leaf containing Q_i^+ (resp., Q_i^-) is *fathomed* if:

1. The optimal solution to the LP relaxation of Q_i^+ (resp., Q_i^-) satisfies the constraints of the original MILP Q'.

2. The relaxation of Q_i^+ (resp., Q_i^-) is infeasible, so Q_i^+ (resp., Q_i^-) must be infeasible as well.

3. The objective value of the LP relaxation of Q_i^+ (resp., Q_i^-) is smaller than the objective value of the best known feasible solution, so the optimal solution to Q_i^+ (resp., Q_i^-) is no better than the best known feasible solution.

B&B terminates when every leaf has been fathomed. It returns the best known feasible solution, which is optimal. See Algorithm 1 for the pseudocode.

The most common node selection policy is the *best bound policy*. Given a B&B tree, it selects the unfathomed leaf containing the MILP Q with the maximum LP relaxation objective value. Another common policy is the *depth-first policy*, which selects the next unfathomed leaf in the tree in depth-first order.

Example 2.2. In Figure 1, we show the search tree built by B&B given as input the following
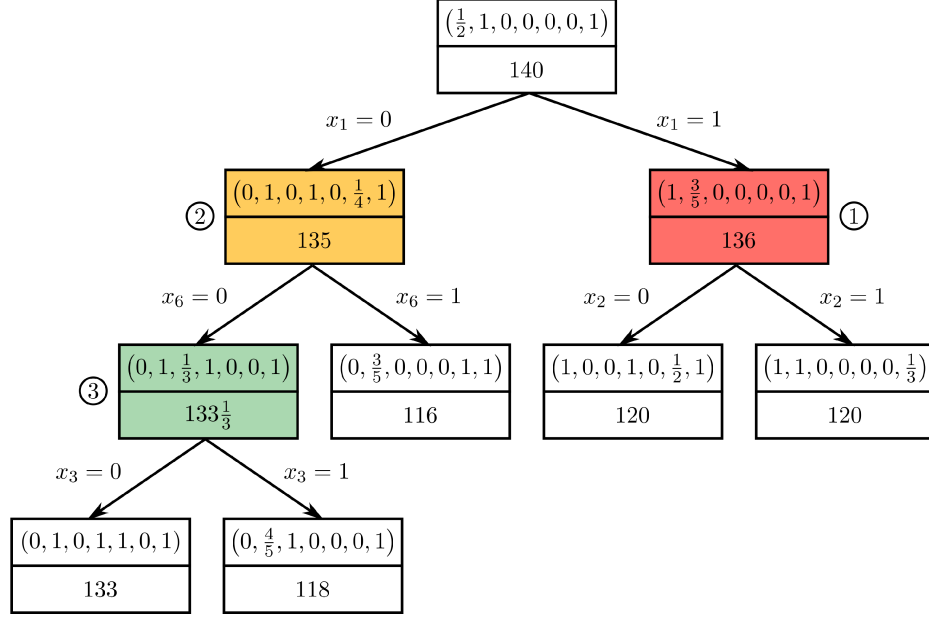
Figure 1: Illustration of Example 2.2.

MILP [Kolesar, 1967]:

$$
\begin{aligned}
\text{maximize} \quad & 40x_1 + 60x_2 + 10x_3 + 10x_4 + 3x_5 + 20x_6 + 60x_7 \\
\text{subject to} \quad & 40x_1 + 50x_2 + 30x_3 + 10x_4 + 10x_5 + 40x_6 + 30x_7 \leq 100 \quad\quad (1) \\
& x_1, \ldots, x_7 \in \{0, 1\}.
\end{aligned}
$$

Each rectangle denotes a node in the B&B tree. Given a node Q, the top portion of its rectangle displays the optimal solution $\breve{\boldsymbol{x}}_Q$ to the LP relaxation of Q, which is the MILP (1) with the additional constraints labeling the edges from the root to Q. The bottom portion of the rectangle corresponding to Q displays the objective value $\breve{c}_Q$ of the optimal solution to this LP relaxation, i.e., $\breve{c}_Q = (40, 60, 10, 10, 3, 20, 60) \cdot \breve{\boldsymbol{x}}_Q$. In this example, the node selection policy is the best bound policy and the variable selection policy selects the "most fractional" variable: the variable x_i such that $\breve{x}_Q[i]$ is closest to $\frac{1}{2}$, i.e., $i = \operatorname{argmax} \{\min \{1 - \breve{x}_Q[i], \breve{x}_Q[i]\}\}$.

In Figure 1, the algorithm first explores the root. At this point, it has the option of exploring either the left or the right child. Since the optimal objective value of the right child (136) is greater than the optimal objective value of the left child (135), B&B will next explore the pink node (marked ①). Next, B&B can either explore either of the pink node's children or the orange node (marked ②). Since the optimal objective value of the orange node (135) is greater than the optimal objective values of the pink node's children (120), B&B will next explore the orange node. After that B&B can explore either of the orange node's children or either of the pink node's children. The optimal objective value of the green node (marked ③) is higher than the optimal objective values of the orange node's right child (116) and the pink node's children (120), so B&B will next explore the green node. At this point, it finds an integral solution, which satisfies all of the constraints of the original MILP (1). This integral solution has an objective value of 133. Since all of the other leafs have smaller objective values, the algorithm cannot find a better solution by exploring those leafs. Therefore, the algorithm fathoms all of the leafs and terminates.

2.1.2 Variable selection in MILP tree search

Variable selection policies typically depend on a real-valued *score* per variable x_i.

Definition 2.1 (Score-based variable selection policy). Let `score` be a deterministic function that takes as input a partial search tree $\mathcal{T}$, a leaf Q of that tree, and an index i and returns a real value $(\texttt{score}(\mathcal{T}, Q, i) \in \mathbb{R})$. For a leaf Q of a tree $\mathcal{T}$, let $N_{\mathcal{T},Q}$ be the set of variables that have not yet been branched on along the path from the root of $\mathcal{T}$ to Q. A score-based variable selection policy selects the variable $\operatorname{argmax}_{x_j \in N_{\mathcal{T},Q}} \{\texttt{score}(\mathcal{T}, Q, j)\}$ to branch on at the node Q.

We list several common definitions of the function `score` below. Recall that for a MILP Q with objective function $\boldsymbol{c} \cdot \boldsymbol{x}$, we denote an optimal solution to the LP relaxation of Q as $\breve{\boldsymbol{x}}_Q = (\breve{x}_Q[1], \dots \breve{x}_Q[n])$. We also use the notation $\breve{c}_Q$ to denote the objective value of the optimal solution to the LP relaxation of Q, i.e., $\breve{c}_Q = \boldsymbol{c}^\top \breve{\boldsymbol{x}}_Q$. Finally, we use the notation Q_i^+ (resp., Q_i^-) to denote the MILP Q with the additional constraint that $x_i = 1$ (resp., $x_i = 0$). If Q_i^+ (resp., Q_i^-) is infeasible, then we set $\breve{c}_Q - \breve{c}_{Q_i^+}$ (resp., $\breve{c}_Q - \breve{c}_{Q_i^-}$) to be some large number greater than $||\boldsymbol{c}||_1$.

Most fractional. In this case, $\texttt{score}(\mathcal{T}, Q, i) = \min\left\{1 - \breve{x}_Q[i], \breve{x}_Q[i]\right\}$. The variable that maximizes $\texttt{score}(\mathcal{T}, Q, i)$ is the "most fractional" variable, since it is the variable such that $\breve{x}_Q[i]$ is closest to $\frac{1}{2}$.

Linear scoring rule [Linderoth and Savelsbergh, 1999]. In this case, $\texttt{score}(\mathcal{T}, Q, i) = (1-\mu) \cdot \max\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\right\} + \mu \cdot \min\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\right\}$ where $\mu \in [0, 1]$ is a user-specified parameter. This parameter balances an "optimistic" and a "pessimistic" approach to branching: An optimistic approach would choose the variable that maximizes $\max\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\right\}$, which corresponds to $\mu = 0$, and a pessimistic approach would choose the variable that maximizes $\min\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\right\}$, which corresponds to $\mu = 1$.

Product scoring rule [Achterberg, 2009]. In this case, $\texttt{score}(\mathcal{T}, Q, i) = \max\left\{\breve{c}_Q - \breve{c}_{Q_i^-}, \gamma\right\} \cdot \max\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \gamma\right\}$ where $\gamma = 10^{-6}$. Comparing $\breve{c}_Q - \breve{c}_{Q_i^-}$ and $\breve{c}_Q - \breve{c}_{Q_i^+}$ to γ allows the algorithm to compare two variables even if $\breve{c}_Q - \breve{c}_{Q_i^-} = 0$ or $\breve{c}_Q - \breve{c}_{Q_i^+} = 0$. After all, suppose the scoring rule simply calculated the product $\left(\breve{c}_Q - \breve{c}_{Q_i^-}\right) \cdot \left(\breve{c}_Q - \breve{c}_{Q_i^+}\right)$ without comparing to γ. If $\breve{c}_Q - \breve{c}_{Q_i^-} = 0$, then the score equals 0, canceling out the value of $\breve{c}_Q - \breve{c}_{Q_i^+}$ and thus losing the information encoded by this difference.

Entropic lookahead scoring rule [Gilpin and Sandholm, 2011]. Let

$$e(x) = \begin{cases} -x \log_2(x) - (1-x)\log_2(1-x) & \text{if } x \in (0,1) \\ 0 & \text{if } x \in \{0, 1\}. \end{cases}$$

Set $\texttt{score}(\mathcal{T}, Q, i) = -\sum_{j=1}^{n} \left(1 - \breve{x}_Q[i]\right) \cdot e\left(\breve{x}_{Q_i^-}[j]\right) + \breve{x}_Q[i] \cdot e\left(\breve{x}_{Q_i^+}[j]\right)$.

Alternative definitions of the linear and product scoring rules. In practice, it is often too slow to compute the differences $\breve{c}_Q - \breve{c}_{Q_i^-}$ and $\breve{c}_Q - \breve{c}_{Q_i^+}$ for every variable, since it requires solving as many as $2n$ LPs. A faster option is to partially solve the LP relaxations of Q_i^- and

7

Q_i^+, starting at $\breve{\boldsymbol{x}}_Q$ and running a small number of simplex iterations. Denoting the new objective values as $\tilde{c}_{Q_i^-}$ and $\tilde{c}_{Q_i^+}$, we can revise the linear scoring rule to be $\texttt{score}(\mathcal{T}, Q, i) = (1 - \mu) \cdot \max\left\{\breve{c}_Q - \tilde{c}_{Q_i^+}, \breve{c}_Q - \tilde{c}_{Q_i^-}\right\} + \mu \cdot \min\left\{\breve{c}_Q - \tilde{c}_{Q_i^+}, \breve{c}_Q - \tilde{c}_{Q_i^-}\right\}$ and we can revise the product scoring rule to be $\texttt{score}(\mathcal{T}, Q, i) = \max\left\{\breve{c}_Q - \tilde{c}_{Q_i^-}, \gamma\right\} \cdot \max\left\{\breve{c}_Q - \tilde{c}_{Q_i^+}, \gamma\right\}$. Other popular alternatives to computing $\breve{c}_{Q_i^-}$ and $\breve{c}_{Q_i^+}$ that fit within our framework are *pseudo-cost branching* [Bénichou et al., 1971, Gauthier and Ribière, 1977, Linderoth and Savelsbergh, 1999] and *reliability branching* [Achterberg et al., 2005].

3 Guarantees for data-driven learning to branch

In this section, we begin with our formal problem statement. We then present worst-case distributions over MILP instances demonstrating that learning over any data-independent discretization of the parameter space can be inadequate. Finally, we present sample complexity guarantees and a learning algorithm. Throughout the remainder of this paper, we assume that all aspects of the tree search algorithm except the variable selection policy, such as the node selection policy, are fixed.

3.1 Problem statement

Let $\mathcal{D}$ be a distribution over MILPs Q. For example, $\mathcal{D}$ could be a distribution over clustering problems a biology lab solves day to day, formulated as MILPs. Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be a set of variable selection scoring rules, such as those in Section 2.1.2. Our goal is to learn a convex combination $\mu_1 \texttt{score}_1 + \cdots + \mu_d \texttt{score}_d$ of the scoring rules that is nearly optimal in expectation over $\mathcal{D}$. More formally, let $\texttt{cost}$ be an abstract cost function that takes as input a problem instance Q and a scoring rule $\texttt{score}$ and returns some measure of the quality of B&B using $\texttt{score}$ on input Q. For example, $\texttt{cost}(Q, \texttt{score})$ might be the number of nodes produced by running B&B using $\texttt{score}$ on input Q. We say that an algorithm (ϵ, δ)-learns a convex combination of the d scoring rules $\texttt{score}_1, \ldots, \texttt{score}_d$ if for any distribution $\mathcal{D}$, with probability at least $1 - \delta$ over the draw of a sample $\{Q_1, \ldots, Q_m\} \sim \mathcal{D}^m$, the algorithm returns a convex combination $\texttt{score} = \hat{\mu}_1 \texttt{score}_1 + \cdots + \hat{\mu}_d \texttt{score}_d$ such that $\mathbb{E}_{Q \sim \mathcal{D}}\left[\texttt{cost}(Q, \texttt{score})\right] - \mathbb{E}_{Q \sim \mathcal{D}}\left[\texttt{cost}(Q, \texttt{score}^*)\right] \leq \epsilon$, where $\texttt{score}^*$ is the convex combination of $\texttt{score}_1, \ldots, \texttt{score}_d$ with minimal expected cost. In this work, we prove that only a small number of samples is sufficient to ensure (ϵ, δ)-learnability.

Following prior work (e.g., [Hutter et al., 2009, Kleinberg et al., 2017]), we assume that there is some cap κ on the range of the cost function $\texttt{cost}$. For example, if $\texttt{cost}$ is the size of the search tree, we may choose to terminate the algorithm when the tree size grows beyond some bound κ. We also assume that the problem instances in the support of $\mathcal{D}$ are over n binary variables, for some $n \in \mathbb{N}$.

Our results hold for cost functions that are *tree-constant*, which means that for any problem instance Q, so long as the scoring rules $\texttt{score}_1$ and $\texttt{score}_2$ result in the same search tree, $\texttt{cost}(Q, \texttt{score}_1) = \texttt{cost}(Q, \texttt{score}_2)$. For example, the size of the search tree is tree-constant.

3.2 Impossibility results for data-independent approaches

In this section, we focus on MILP tree search and prove that it is *impossible* to find a nearly optimal B&B configuration using a data-independent discretization of the parameters. Specifically, suppose $\texttt{score}_1(\mathcal{T}, Q, i) = \min\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\right\}$ and $\texttt{score}_2(\mathcal{T}, Q, i) = \max\left\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\right\}$ and suppose the cost function $\texttt{cost}(Q, \mu\texttt{score}_1 + (1 - \mu)\texttt{score}_2)$ measures the size of the tree

(a) The tree size plot for the instance Q_a as a function of μ.

(b) The tree size plot for the instance Q_b as a function of μ.

(c) The expected tree size plot under the distribution $\mathcal{D}$ as a function of μ.
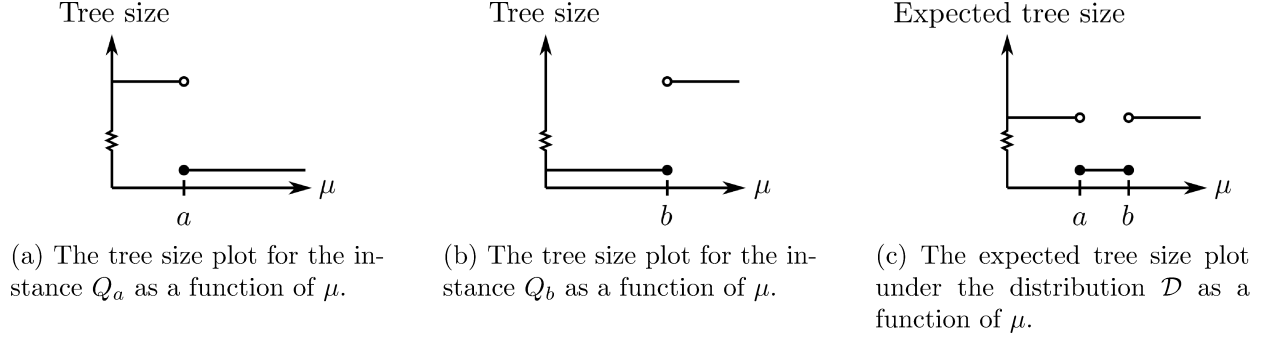
Figure 2: Illustrations of the proof of Theorem 3.1.

produced by B&B when using a fixed but arbitrary node selection policy. We would like to learn a nearly optimal convex combination $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ of these two rules with respect to $\mathtt{cost}$. Gauthier and Ribière [1977] proposed setting $\mu = 1/2$, Bénichou et al. [1971] and Beale [1979] suggested setting $\mu = 1$, and Linderoth and Savelsbergh [1999] found that $\mu = 2/3$ performs well. Achterberg [2009] found that experimentally, $\mu = 5/6$ performed best when comparing among $\mu \in \{0, 1/2, 2/3, 5/6, 1\}$.

We show that for *any* discretization of the parameter space $[0, 1]$, there exists an infinite family of distributions over MILP problem instances such that for any parameter in the discretization, the expected tree size is exponential in n. Yet, there exists an infinite number of parameters such that the tree size is just a constant (with probability 1). The proof is in Appendix C.

Theorem 3.1. *Let*

$$\mathtt{score}_1(\mathcal{T}, Q, i) = \min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, \ \mathtt{score}_2(\mathcal{T}, Q, i) = \max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\},$$

and $\mathtt{cost}(Q, \mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2)$ *be the size of the tree produced by B&B. For every a, b such that $\frac{1}{3} < a < b < \frac{1}{2}$ and for all even $n \geq 6$, there exists an infinite family of distributions $\mathcal{D}$ over MILP instances with n variables such that if $\mu \in [0, 1] \setminus (a, b)$, then*

$$\mathbb{E}_{Q \sim \mathcal{D}}\left[\mathtt{cost}\left(Q, \mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2\right)\right] = \Omega\left(2^{(n-9)/4}\right)$$

and if $\mu \in (a, b)$, then with probability 1, $\mathtt{cost}\left(Q, \mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2\right) = O(1)$. This holds no matter which node selection policy B&B uses.

Proof sketch. We populate the support of the distribution $\mathcal{D}$ by relying on two helpful theorems: Theorem 3.2 and C.4. In Theorem 3.2, we prove that for all $\mu^* \in (0, 1)$, there exists an infinite family $\mathcal{F}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{F}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ results in a B&B tree with $O(1)$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results a tree with $2^{(n-4)/2}$ nodes. Conversely, in Theorem C.4, we prove that there exists an infinite family $\mathcal{G}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{G}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ results in a B&B tree with $2^{(n-5)/4}$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results a tree with $O(1)$ nodes. Now, let Q_a be an arbitrary instance in $\mathcal{G}_{n,a}$ and let Q_b be an arbitrary instance in $\mathcal{F}_{n,b}$. The theorem follows by letting $\mathcal{D}$ be a distribution such that $\Pr_{Q \sim \mathcal{D}}[Q = Q_a] = \Pr_{Q \sim \mathcal{D}}[Q = Q_b] = 1/2$. See Figure 2 for an illustration.

Throughout the proof of this theorem, we assume the node selection policy is depth-first search. We then prove that for any infeasible MILP, if NSP and NSP' are two node selection policies and $\mathtt{score} = \mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ for any $\mu \in [0, 1]$, then tree $\mathcal{T}$ B&B builds using NSP and $\mathtt{score}$

equals the tree $\mathcal{T}'$ it builds using NSP' and `score` (see Theorem C.9). Thus, the theorem holds for any node selection policy. □

We now provide a proof sketch of Theorem 3.2, which helps us populate the support of the worst-case distributions in Theorem 3.1. The full proof is in Appendix C.

Theorem 3.2. *Let*

$$\mathit{score}_1(\mathcal{T}, Q, i) = \min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, \ \mathit{score}_2(\mathcal{T}, Q, i) = \max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\},$$

and $\mathit{cost}(Q, \mu\,\mathit{score}_1 + (1-\mu)\mathit{score}_2)$ *be the size of the tree produced by B&B. For all even* $n \geq 6$ *and all* $\mu^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, *there exists an infinite family* $\mathcal{F}_{n,\mu^*}$ *of MILP instances such that for any* $Q \in \mathcal{F}_{n,\mu^*}$, *if* $\mu \in [0, \mu^*)$, *then the scoring rule* $\mu\,\mathit{score}_1 + (1-\mu)\mathit{score}_2$ *results in a B&B tree with* $O(1)$ *nodes and if* $\mu \in (\mu^*, 1]$, *the scoring rule results a tree with* $2^{(n-4)/2}$ *nodes.*

Proof sketch. The MILP instances in $\mathcal{F}_{n,\mu^*}$ are inspired by a worst-case B&B instance introduced by Jeroslow [1974]. He proved that for any odd n', every B&B algorithm will build a tree with $2^{(n'-1)/2}$ nodes before it determines that for any $\boldsymbol{c} \in \mathbb{R}^{n'}$, the following MILP is infeasible:

$$\begin{aligned}
\text{maximize} \quad & \boldsymbol{c} \cdot \boldsymbol{x} \\
\text{subject to} \quad & 2\sum_{i=1}^{n'} x_i = n' \\
& \boldsymbol{x} \in \{0,1\}^{n'}.
\end{aligned}$$

We build off of this MILP to create the infinite family $\mathcal{F}_{n,\mu^*}$. Each MILP in $\mathcal{F}_{n,\mu^*}$ combines a hard version of Jeroslow's instance on $n-3$ variables $\{x_1, \ldots, x_{n-3}\}$ and an easy version on 3 variables $\{x_{n-2}, x_{n-1}, x_n\}$. Branch-and-bound only needs to determine that one of these problems is infeasible in order to terminate. The key idea of this proof is that if B&B branches on all variables in $\{x_{n-2}, x_{n-1}, x_n\}$ first, it will terminate upon making a small tree. However, if B&B branches on all variables in $\{x_1, \ldots, x_{n-3}\}$ first, it will create a tree with exponential size before it terminates. The challenge is to design an objective function that enforces the first behavior when $\mu < \mu^*$ and the second behavior when $\mu > \mu^*$. Proving this is the bulk of the work.

In a bit more detail, every instance in $\mathcal{F}_{n,\mu^*}$ is defined as follows. For any constant $\gamma \geq 1$, let $\boldsymbol{c}_1 = \gamma(1, 2, \ldots, n-3)$ and let $\boldsymbol{c}_2 = \gamma\left(0, \frac{3}{2}, 3 - \frac{1}{2\mu^*}\right)$. Let $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2) \in \mathbb{R}^n$ be the concatenation of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Let $Q_{\gamma,n}$ be the MILP

$$\begin{aligned}
\text{maximize} \quad & \boldsymbol{c} \cdot \boldsymbol{x} \\
\text{subject to} \quad & 2\sum_{i=1}^{n-3} x_i = n-3 \\
& 2\left(x_{n-2} + x_{n-1} + x_n\right) = 3 \\
& \boldsymbol{x} \in \{0,1\}^n.
\end{aligned}$$

We define $\mathcal{F}_{n,\mu^*} = \{Q_{n,\gamma} : \gamma \geq 1\}$.

For example, if $\gamma = 1$ and $n = 8$, then $Q_{\gamma,n}$ is

$$\begin{aligned}
\text{maximize} \quad & \left(1, 2, 3, 4, 5, 0, \tfrac{3}{2}, 3 - \tfrac{1}{2\mu^*}\right) \cdot \boldsymbol{x} \\
\text{subject to} \quad & \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 5 \\ 3 \end{pmatrix} \\
& \boldsymbol{x} \in \{0,1\}^8.
\end{aligned}$$

As is illustrated in Figure 3, we have essentially "glued together" two disjoint versions of Jeroslow's instance: the first five variables of $Q_{1,8}$ correspond to a "big" version of Jeroslow's instance and the

$$\text{maximize} \quad \left(1,2,3,4,5,0,\tfrac{3}{2},3-\tfrac{1}{2\mu^*}\right)\cdot \boldsymbol{x}$$

$$\text{subject to} \quad \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \end{pmatrix}\boldsymbol{x} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

$$\boldsymbol{x}\in\{0,1\}^8.$$

(a) A big version of Jeroslow's instance on five variables.

$$\text{maximize} \quad \left(1,2,3,4,5,0,\tfrac{3}{2},3-\tfrac{1}{2\mu^*}\right)\cdot \boldsymbol{x}$$

$$\text{subject to} \quad \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \end{pmatrix}\boldsymbol{x} = \begin{pmatrix} 5 \\ 3 \end{pmatrix}$$

$$\boldsymbol{x}\in\{0,1\}^8.$$

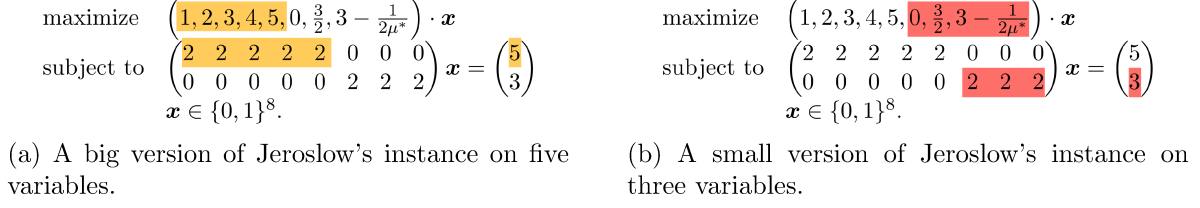(b) A small version of Jeroslow's instance on three variables.

Figure 3: Illustrations of the construction from Theorem 3.2.

last three variables correspond to a small version. Since the goal is maximization, the solution to the LP relaxation of $Q_{1,8}$ will try to obtain as much value from the first five variables $\{x_1,\dots,x_5\}$ as it can, but it is constrained to ensure that $2(x_1+\cdots+x_5)=5$. Therefore, the first five variables will be set to $\left(0,0,\tfrac{1}{2},1,1\right)$. Similarly, the solution to the LP relaxation of $Q_{1,8}$ will set $(x_6,x_7,x_8)=\left(0,\tfrac{1}{2},1\right)$ because $\tfrac{3}{2}<3-\tfrac{1}{2\mu^*}$ under our assumption that $\mu^*>\tfrac{1}{3}$. Thus, the solution to the LP relaxation of $Q_{1,8}$ is $\left(0,0,\tfrac{1}{2},1,1,0,\tfrac{1}{2},1\right)$. There are only two fractional variables that B&B might branch on: x_3 and x_7. Straightforward calculations show that if $\mathcal{T}$ is the B&B tree so far, which just consists of the root node, $\mu\mathtt{score}_1\left(\mathcal{T},Q_{\gamma,n},3\right)+(1-\mu)\mathtt{score}_2\left(\mathcal{T},Q_{\gamma,n},3\right)=\tfrac{\gamma}{2}$ and $\mu\mathtt{score}_1\left(\mathcal{T},Q_{\gamma,n},7\right)+(1-\mu)\mathtt{score}_2\left(\mathcal{T},Q_{\gamma,n},7\right)=\tfrac{3\gamma}{4}-\tfrac{\mu\gamma}{4\mu^*}$. This means that B&B will branch first on variable x_7, which corresponds to the small version of Jeroslow's instance (see Figure 3b) if and only if $\tfrac{\gamma}{2}<\tfrac{3\gamma}{4}-\tfrac{\mu\gamma}{4\mu^*}$, which occurs if and only if $\mu<\mu^*$. We show that this first branch sets off a cascade: if B&B branches first on variable x_7, then it will proceed to branch on all variables in $\{x_6,x_7,x_8\}$, thus terminating upon making a small tree. Meanwhile, if it branches on variable x_3 first, it will then only branch on variables in $\{x_1,\dots,x_5\}$, creating a larger tree.

In the full proof, we generalize beyond eight variables to n, and expand the large version of Jeroslow's instance (as depicted in Figure 3a) from five variables to $n-3$. When $\mu<\mu^*$, we simply track B&B's progress to make sure it only branches on variables from the small version of Jeroslow's instance (x_{n-2},x_{n-1},x_n) before figuring out the MILP is infeasible. Therefore, the tree will have constant size. When $\mu>\mu^*$, we prove by induction that if B&B has only branched on variables from the big version of Jeroslow's instance $(x_1,\dots,x_{n-3})$, it will continue to only branch on those variables. We also prove it will branch on about half of these variables along each path of the B&B tree. The tree will thus have exponential size. $\qquad\square$

3.3 Sample complexity guarantees

We now provide worst-case guarantees on the number of samples sufficient to (ϵ,δ)-learn a convex combination of scoring rules. These results bound the number of samples sufficient to ensure that for any convex combination $\mathtt{score}=\mu_1\mathtt{score}_1+\cdots+\mu_d\mathtt{score}_d$ of scoring rules, the empirical cost of tree search using $\mathtt{score}$ is close to its expected cost. Therefore, the algorithm designer can optimize $(\mu_1,\dots,\mu_d)$ over the samples without any further knowledge of the distribution. Moreover these sample complexity guarantees apply for any procedure the algorithm designer uses to tune $(\mu_1,\dots,\mu_d)$, be it an approximation, heuristic, or optimal algorithm. She can use our guarantees to bound the number of samples she needs to ensure that performance on the sample generalizes to the distribution.

In Section 3.3.1 we provide generalization guarantees for a family of scoring rules we call *path-wise*, which includes many well-known scoring rules as special cases. In this case, the number of samples is surprisingly small given the complexity of these problems: it grows only quadratically with the number of variables. In Section 3.3.2, we provide guarantees that apply to any scoring rule, path-wise or otherwise.

(a) A B&B search tree $\mathcal{T}$.

(b) The path $\mathcal{T}_Q$ from the root of the tree $\mathcal{T}$ in Figure 4a to the node labeled Q.

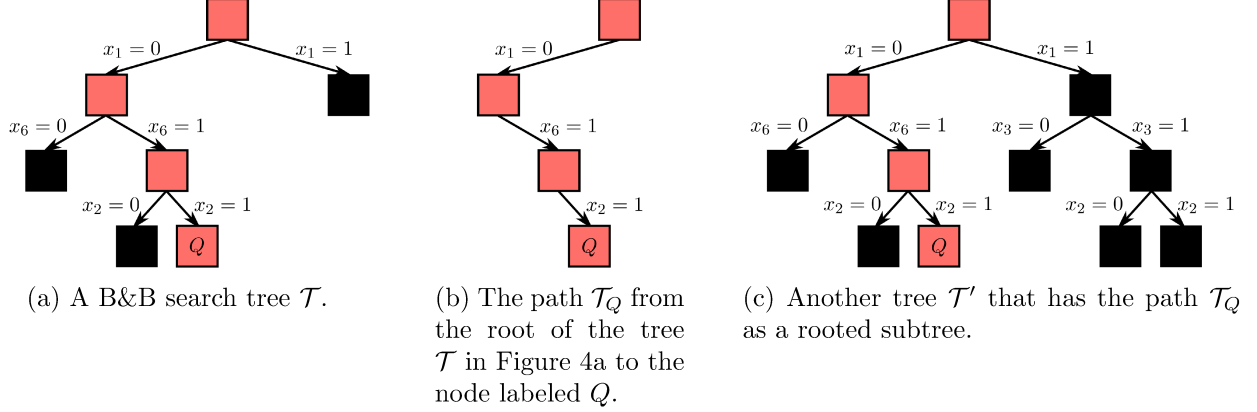(c) Another tree $\mathcal{T}'$ that has the path $\mathcal{T}_Q$ as a rooted subtree.

Figure 4: Illustrations to accompany the definition of a path-wise scoring rule (Definition 3.1). If the scoring rule `score` is path-wise, then for any variable x_i, `score`$(\mathcal{T}, Q, i) =$ `score`$(\mathcal{T}', Q, i) =$ `score`$(\mathcal{T}_Q, Q, i)$.

3.3.1   Path-wise scoring rules

The guarantees in this section apply broadly to a class of scoring rules we call *path-wise* scoring rules. Given a node Q in a search tree $\mathcal{T}$, we denote the path from the root of $\mathcal{T}$ to the node Q as $\mathcal{T}_Q$. The path $\mathcal{T}_Q$ includes all nodes and edge labels from the root of $\mathcal{T}$ to Q. For example, Figure 4b illustrates the path $\mathcal{T}_Q$ from the root of the tree $\mathcal{T}$ in Figure 4a to the node labeled Q. We now state the definition of path-wise scoring rules.

Definition 3.1 (Path-wise scoring rule). The function `score` is a path-wise scoring rule if for all search trees $\mathcal{T}$, all nodes Q in $\mathcal{T}$, and all variables x_i,

$$\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}_Q, Q, i) \tag{2}$$

where $\mathcal{T}_Q$ is the path from the root of $\mathcal{T}$ to Q.[1] See Figure 4 for an illustration.

Definition 3.1 requires that if the node Q appears at the end of the same path in two different B&B trees, then any path-wise scoring rule must assign every variable the same score with respect to Q in both trees.

Path-wise scoring rules include many well-studied rules as special cases, such as the *most fractional*, *product*, and *linear* scoring rules, as defined in Section 2.1.2. The same is true when B&B only partially solves the LP relaxations of Q_i^- and Q_i^+ for every variable x_i by running a small number of simplex iterations, as we describe in Section 2.1.2 and as is our approach in our experiments. In fact, these scoring rules depend only on the node in question, rather than the path from the root to the node. We present our sample complexity bound for the more general class of path-wise scoring rules because this class captures the level of generality the proof holds for. On the other hand, *pseudo-cost branching* [Bénichou et al., 1971, Gauthier and Ribière, 1977, Linderoth and Savelsbergh, 1999] and *reliability branching* [Achterberg et al., 2005], two widely-used branching strategies, are not path-wise, but our more general results from Section 3.3.2 do apply to those strategies.

In order to prove our generalization guarantees, we make use of the following key structure which bounds the number of search trees branch-and-bound will build on a given instance over the

---

[1]Under this definition, the scoring rule can simulate B&B for any number of steps starting at any point in the tree and use that information to calculate the score, so long as Equality (2) always holds.

entire range of parameters. In essence, this is a bound on the intrinsic complexity of the algorithm class defined by the range of parameters, and this bound on algorithm class's intrinsic complexity implies strong generalization guarantees.

Lemma 3.3. *Let* `cost` *be a tree-constant cost function, let* `score`$_1$ *and* `score`$_2$ *be two path-wise scoring rules, and let Q be an arbitrary problem instance over n binary variables. There are $T \leq 2^{n(n-1)/2}n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, across all $\mu \in I_j$, the scoring rule μ`score`$_1 + (1 - \mu)$`score`$_2$ results in the same search tree.*

Proof. We prove this lemma first by considering the actions of an alternative algorithm A' which runs exactly like B&B, except it only fathoms nodes if they are integral or infeasible. We then relate the behavior of A' to the behavior of B&B to prove the lemma.

First, we prove the following bound on the number of search trees A' will build on a given instance over the entire range of parameters. This bound matches that in the lemma statement.

Claim 3.4. *There are $T \leq 2^{n(n-1)/2}n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, the search tree A' builds using the scoring rule μ`score`$_1 + (1 - \mu)$`score`$_2$ is invariant across all $\mu \in I_j$.[2]*

Proof sketch of Claim 3.4. We prove this claim by induction. For a tree $\mathcal{T}$, let $\mathcal{T}[i]$ be the nodes of depth at most i. We prove that for $i \in \{1, \ldots, n\}$, there are $T \leq 2^{i(i-1)/2}n^i$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j and any two parameters $\mu, \mu' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees A' builds using the scoring rules μ`score`$_1 + (1 - \mu)$`score`$_2$ and μ'`score`$_1 + (1 - \mu')$`score`$_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. Suppose that this is indeed the case for some $i \in \{1, \ldots, n\}$ and consider an arbitrary interval I_j and any two parameters $\mu, \mu' \in I_j$. Consider an arbitrary node Q in $\mathcal{T}[i-1]$ (or equivalently, $\mathcal{T}'[i-1]$) at depth $i-1$. If Q is integral or infeasible, then it will be fathomed no matter which parameter $\mu \in I_j$ the algorithm A' uses. Otherwise, for all $\mu \in I_j$, let $\mathcal{T}_\mu$ be the state of the search tree A' builds using the scoring rule μ`score`$_1 + (1-\mu)$`score`$_2$ at the point when it branches on Q. By the inductive hypothesis, we know that across all $\mu \in I_j$, the path from the root to Q in $\mathcal{T}_\mu$ is invariant, and we refer to this path as $\mathcal{T}_Q$. Given a parameter $\mu \in I_j$, the variable x_k will be branched on at node Q so long as $k = \text{argmax}_\ell \{\mu$`score`$_1(\mathcal{T}_\mu, Q, \ell) + (1 - \mu)$`score`$_2(\mathcal{T}_\mu, Q, \ell)\}$, or equivalently, so long as $k = \text{argmax}_\ell \{\mu$`score`$_1(\mathcal{T}_Q, Q, \ell) + (1 - \mu)$`score`$_2(\mathcal{T}_Q, Q, \ell)\}$. In other words, the decision of which variable to branch on is determined by a convex combination of the constant values `score`$_1(\mathcal{T}_Q, Q, \ell)$ and `score`$_2(\mathcal{T}_Q, Q, \ell)$ no matter which parameter $\mu \in I_j$ the algorithm A' uses. Here, we critically use the fact that the scoring rule is path-wise.

Since μ`score`$_1(\mathcal{T}_Q, Q, \ell) + (1 - \mu)$`score`$_2(\mathcal{T}_Q, Q, \ell)$ is a linear function of μ for all ℓ, there are at most n intervals subdividing the interval I_j such that the variable branched on at node Q is fixed. Moreover, there are at most 2^{i-1} nodes at depth $i-1$, and each node similarly contributes a subpartition of I_j of size n. If we merge all 2^{i-1} partitions, we have $T' \leq 2^{i-1}(n-1) + 1$ intervals $I_1', \ldots, I_{T'}'$ partitioning I_j where for any interval I_p' and any two parameters $\mu, \mu' \in I_p'$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees A' builds using the scoring rules μ`score`$_1 + (1-\mu)$`score`$_2$ and μ'`score`$_1 + (1-\mu')$`score`$_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. We can similarly subdivide each interval $I_1, \ldots, I_T$. The claim then follows from counting the number of subdivisions. $\square$

Next, we explicitly relate the behavior of B&B to A', proving that the search tree B&B builds is a rooted subtree of the search tree A' builds.

---

[2]This claim holds even when `score`$_1$ and `score`$_2$ are members of the more general class of *depth-wise* scoring rules, which we define as follows. For any search tree $\mathcal{T}$ of depth depth$(\mathcal{T})$ and any $j \in [n]$, let $\mathcal{T}[j]$ be the subtree of $\mathcal{T}$ consisting of all nodes in $\mathcal{T}$ of depth at most j. We say that `score` is a *depth-wise* scoring rule if for all search trees $\mathcal{T}$, all $j \in [\text{depth}(\mathcal{T})]$, all nodes Q of depth j, and all variables x_i, `score`$(\mathcal{T}, Q, i) = $ `score`$(\mathcal{T}[j], Q, i)$.
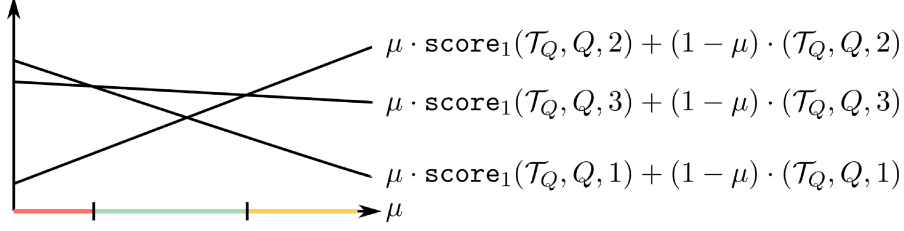
Figure 5: Illustrations of the proof of Claim 3.4 for a hypothetical MIP Q where the algorithm can either branch on x_1, x_2, or x_3 next. In the left-most interval (colored pink), x_1 will be branched on next, in the central interval (colored green), x_3 will be branched on next, and in the right-most interval (colored orange), x_2 next will be branched on next.

Claim 3.5. *Given a parameter $\mu \in [0,1]$, let $\mathcal{T}$ and $\mathcal{T}'$ be the trees B&B and A' build respectively using the scoring rule $\mu \mathtt{score}_1 + (1-\mu)\mathtt{score}_2$. For any node Q of $\mathcal{T}$, let $\mathcal{T}_Q$ be the path from the root of $\mathcal{T}$ to Q. Then $\mathcal{T}_Q$ is a rooted subtree of $\mathcal{T}'$.*

Proof of Claim 3.5. The path $\mathcal{T}_Q$ can be labeled by a sequence of indices from $\{0,1\}$ and a sequence of variables from $\{x_1, \ldots, x_n\}$ describing which variable is branched on and which value it takes on along the path $\mathcal{T}_Q$. Let $((j_1, x_{i_1}), \ldots, (j_t, x_{i_t}))$ be this sequence of labels, where t is the number of edges in $\mathcal{T}_Q$. We can similarly label every edge in $\mathcal{T}'$. We claim that there exists a path beginning at the root of $\mathcal{T}'$ with the labels $((j_1, x_{i_1}), \ldots, (j_t, x_{i_t}))$.

For a contradiction, suppose no such path exists. Let (j_τ, x_{i_τ}) be the earliest label in the sequence $((j_1, x_{i_1}), \ldots, (j_t, x_{i_t}))$ where there is a path beginning at the root of $\mathcal{T}'$ with the labels $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$, but there is no way to continue the path using an edge labeled (j_τ, x_{i_τ}). There are exactly two reasons why this could be the case:

1. The node at the end of the path with labels $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$ was fathomed by A'.

2. The algorithm A' branched on a variable other than x_{i_τ} at the end of the path labeled $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$.

In the first case, since A' only fathoms a node if it is integral or infeasible, we know that B&B will also fathom the node at the end of the path with labels $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$. However, this is not the case since B&B next branches on the variable x_{i_τ}.

The second case is also not possible since the scoring rules are both path-wise. In a bit more detail, let Q' be the node at the end of the path with labels $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$. We refer to this path as $\mathcal{T}_{Q'}$. Let $\bar{\mathcal{T}}$ (respectively, $\bar{\mathcal{T}}'$) be the state of the search tree B&B (respectively, A') has built at the point it branches on Q'. We know that $\mathcal{T}_{Q'}$ is the path from the root to Q' in both of the trees $\bar{\mathcal{T}}$ and $\bar{\mathcal{T}}'$. Therefore, for all variables x_k, $\mu \mathtt{score}_1(\bar{\mathcal{T}}, Q', k) + (1-\mu)\mathtt{score}_2(\bar{\mathcal{T}}, Q', k) = \mu \mathtt{score}_1(\mathcal{T}_{Q'}, Q', k) + (1-\mu)\mathtt{score}_2(\mathcal{T}_{Q'}, Q', k) = \mu \mathtt{score}_1(\bar{\mathcal{T}}', Q', k) + (1-\mu)\mathtt{score}_2(\bar{\mathcal{T}}', Q', k)$. This means that B&B and A' will choose the same variable to branch on at the node Q'.

Therefore, we have reached a contradiction, so the claim holds. □

Next, we use Claims 3.4 and 3.5 to prove Lemma 3.3. Let $I_1, \ldots, I_T$ be the intervals guaranteed to exist by Claim 3.4 and let I_t be an arbitrary one of the intervals. Let μ' and μ'' be two arbitrary parameters from I_t. We will prove that the scoring rules $\mu'\mathtt{score}_1 + (1-\mu')\mathtt{score}_2$ and $\mu''\mathtt{score}_1 + (1-\mu'')\mathtt{score}_2$ result in the same B&B search tree. For a contradiction, suppose that this is not the case. Consider the first iteration where B&B using the scoring rule $\mu'\mathtt{score}_1 + (1-\mu')\mathtt{score}_2$ differs from B&B using the scoring rule $\mu''\mathtt{score}_1 + (1-\mu'')\mathtt{score}_2$. By iteration, we mean lines 4

14

through 18 of Algorithm 1. Up until this iteration, B&B has built the same partial search tree $\mathcal{T}$. Since the node selection policy does not depend on μ' or μ'', B&B will choose the same leaf Q of the B&B search tree to branch on no matter which scoring rule it uses.

Suppose B&B chooses different variables to branch on in Step 5 of Algorithm 1 depending on whether it uses the scoring rule $\mu'\mathtt{score}_1 + (1 - \mu')\mathtt{score}_2$ or $\mu''\mathtt{score}_1 + (1 - \mu'')\mathtt{score}_2$. Let $\mathcal{T}_Q$ be the path from the root of $\mathcal{T}$ to Q. By Claim 3.4, we know that the algorithm A' builds the same search tree using the two scoring rules. Let $\bar{\mathcal{T}}'$ (respectively, $\bar{\mathcal{T}}''$) be the state of the search tree A' has built using the scoring rule $\mu'\mathtt{score}_1 + (1 - \mu')\mathtt{score}_2$ (respectively, $\mu''\mathtt{score}_1 + (1 - \mu'')\mathtt{score}_2$) by the time it branches on the node Q. By Claims 3.4 and 3.5, we know that $\mathcal{T}_Q$ is the path of from the root to Q of both $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$. By Claim 3.4, we know that A' will branch on the same variable x_i at the node Q in both the trees $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$, so $i = \mathrm{argmax}_j \left\{\mu'\mathtt{score}_1(\bar{\mathcal{T}}', Q, j) + (1 - \mu')\mathtt{score}_2(\bar{\mathcal{T}}', Q, j)\right\}$, or equivalently,

$$i = \mathrm{argmax}_j \left\{\mu'\mathtt{score}_1(\mathcal{T}_Q, Q, j) + (1 - \mu')\mathtt{score}_2(\mathcal{T}_Q, Q, j)\right\}, \tag{3}$$

and $i = \mathrm{argmax}_j \left\{\mu''\mathtt{score}_1(\bar{\mathcal{T}}'', Q, j) + (1 - \mu'')\mathtt{score}_2(\bar{\mathcal{T}}'', Q, j)\right\}$, or equivalently,

$$i = \mathrm{argmax}_j \left\{\mu''\mathtt{score}_1(\mathcal{T}_Q, Q, j) + (1 - \mu'')\mathtt{score}_2(\mathcal{T}_Q, Q, j)\right\}. \tag{4}$$

Returning to the search tree $\mathcal{T}$ that B&B is building, Equation (3) implies that

$$i = \mathrm{argmax}_j \left\{\mu'\mathtt{score}_1(\mathcal{T}, Q, j) + (1 - \mu')\mathtt{score}_2(\mathcal{T}, Q, j)\right\}$$

and Equation (4) implies that $i = \mathrm{argmax}_j \left\{\mu''\mathtt{score}_1(\mathcal{T}, Q, j) + (1 - \mu'')\mathtt{score}_2(\mathcal{T}, Q, j)\right\}$. Therefore, B&B will branch on x_i at the node Q no matter which scoring rule it uses.

Finally, since the trees B&B has built so far are identical, the choice of whether or not to fathom the children Q_i^+ and Q_i^- does not depend on the scoring rule, so B&B will fathom the same nodes no matter whether it uses the scoring rule $\mu'\mathtt{score}_1 + (1 - \mu')\mathtt{score}_2$ or $\mu''\mathtt{score}_1 + (1 - \mu'')\mathtt{score}_2$. Therefore, we have reached a contradiction: the iterations were identical. We conclude that the lemma holds. $\qquad \square$

We now show how this structure implies a generalization guarantee. We formulate our guarantees in terms of *pseudo-dimension*. By classic results from learning theory, pseudo-dimension immediately implies generalization guarantees. Pseudo-dimension is defined as follows.

Definition 3.2 (Pseudo-dimension [Pollard, 1984]). Let $\mathcal{F}$ be a class of functions mapping an abstract domain $\mathcal{Z}$ to the set $[-\kappa, \kappa]$. Let $\mathcal{S} = \{z_1, \dots, z_m\}$ be a subset of $\mathcal{Z}$ and let $r_1, \dots, r_m \in \mathbb{R}$ be a set of *targets*. We say that $r_1, \dots, r_m$ *witness the shattering* of $\mathcal{S}$ by $\mathcal{F}$ if for all $\mathcal{S}' \subseteq \mathcal{S}$, there exists some function $f_{\mathcal{S}'} \in \mathcal{F}$ such that for all $z_i \in \mathcal{S}'$, $f_{\mathcal{S}'}(z_i) \leq r_i$ and for all $z_i \notin \mathcal{S}'$, $f_{\mathcal{S}'}(z_i) > r_i$. If there exists some $\boldsymbol{r} \in \mathbb{R}^m$ that witnesses the shattering of $\mathcal{S}$ by $\mathcal{F}$, then we say that $\mathcal{S}$ is *shatterable* by $\mathcal{F}$. Finally, the pseudo-dimension of $\mathcal{F}$, denoted $\mathrm{Pdim}(\mathcal{F})$, is the size of the largest set that is shatterable by $\mathcal{F}$.

Theorem 3.6 provides generalization bounds in terms of pseudo-dimension.

Theorem 3.6 (Pollard [1984]). *For any distribution $\mathcal{D}$ over $\mathcal{Z}$, with probability at least $1 - \delta$ over the draw of $\mathcal{S} = \{z_1, \dots, z_m\} \sim \mathcal{D}^m$, for all $f \in \mathcal{F}$,*

$$\left|\mathbb{E}_{z\sim\mathcal{D}}[f(z)] - \frac{1}{m}\sum_{i=1}^{m} f(z_i)\right| = O\left(\kappa\sqrt{\frac{\mathrm{Pdim}(\mathcal{F})}{m}} + \kappa\sqrt{\frac{\ln(1/\delta)}{m}}\right).$$

Theorem 3.7. *Let* `cost` *be a tree-constant cost function, let* `score`$_1$ *and* `score`$_2$ *be two path-wise scoring rules, and let* $\mathcal{C}$ *be the set of functions* $\{$`cost`$(\cdot, \mu$`score`$_1 + (1 - \mu)$`score`$_2) : \mu \in [0, 1]\}$. *Then* $\mathrm{Pdim}(\mathcal{C}) = O\left(n^2\right)$.

Proof. Suppose that $\mathrm{Pdim}(\mathcal{C}) = m$ and let $\mathcal{S} = \{Q_1, \ldots, Q_m\}$ be a shatterable set of problem instances. We know there exists a set of targets $r_1, \ldots, r_m \in \mathbb{R}$ that witness the shattering of $\mathcal{S}$ by $\mathcal{C}$. This means that for every $\mathcal{S}' \subseteq \mathcal{S}$, there exists a parameter $\mu_{\mathcal{S}'}$ such that if $Q_i \in \mathcal{S}$, then `cost`$(Q_i, \mu_{\mathcal{S}'}$`score`$_1 + (1 - \mu_{\mathcal{S}'})$`score`$_2) \leq r_i$. Otherwise `cost`$(Q_i, \mu_{\mathcal{S}'}$`score`$_1 + (1 - \mu_{\mathcal{S}'})$`score`$_2) > r_i$. Let $M = \{\mu_{\mathcal{S}'} : \mathcal{S}' \subseteq \mathcal{S}\}$. We will prove that $|M| \leq m2^{n(n-1)/2}n^n + 1$, and since $2^m = |M|$, this means that $\mathrm{Pdim}(\mathcal{C}) = m = O\left(\log\left(2^{n(n-1)/2}n^n\right)\right) = O\left(n^2\right)$ (see Lemma C.10 in Appendix C).

To prove that $|M| \leq m2^{n(n-1)/2}n^n + 1$, we rely on Lemma 3.3, which tells us that for any problem instance Q, there are $T \leq 2^{n(n-1)/2}n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, across all $\mu \in I_j$, the scoring rule μ`score`$_1 + (1 - \mu)$`score`$_2$ results in the same search tree. If we merge all T intervals for all samples in $\mathcal{S}$, we are left with $T' \leq m2^{n(n-1)/2}n^n + 1$ intervals $I'_1, \ldots, I'_{T'}$ where for any interval I'_j and any $Q_i \in \mathcal{S}$, `cost`$(Q_i, \mu$`score`$_1 + (1 - \mu)$`score`$_2)$ is constant for all $\mu \in I'_j$. Therefore, at most one element of M can come from each interval, meaning that $|M| \leq T' \leq m2^{n(n-1)/2}n^n + 1$, as claimed. $\square$

3.3.2 General scoring rules

In this section, we provide generalization guarantees that apply to learning convex combinations of any set of scoring rules. Unlike the guarantees in Section 3.3.1, they depend on the size of the search trees B&B is allowed to build. For example, we may choose to terminate the algorithm when the tree size grows beyond some bound $\bar{\kappa}$. The following lemma corresponds to Lemma 3.3 for this setting. For the full proof, see Lemma E.4 in Appendix E.2.1 which proves the lemma for a more general tree search algorithm.

Lemma 3.8. *Let* `cost` *be a tree-constant cost function, let* `score`$_1, \ldots,$ `score`$_d$ *be d arbitrary scoring rules, and let Q be an arbitrary MILP over n binary variables. Suppose we limit B&B to producing search trees of size $\bar{\kappa}$. There is a set $\mathcal{H}$ of at most $n^{2(\bar{\kappa}+1)}$ hyperplanes such that for any connected component R of $[0, 1]^d \setminus \mathcal{H}$, the search tree B&B builds using the scoring rule μ_1`score`$_1 + \cdots + \mu_d$`score`$_d$ is invariant across all $(\mu_1, \ldots, \mu_d) \in R$.*

Proof sketch. The proof has two steps. In Claim E.5, we show that there are at most $n^{\bar{\kappa}}$ different search trees that B&B might produce for the instance Q as we vary the mixing parameter vector $(\mu_1, \ldots, \mu_d)$. In Claim E.6, for each of the possible search trees $\mathcal{T}$ that might be produced, we show that the set of parameter values $(\mu_1, \ldots, \mu_d)$ which give rise to that tree lie in the intersection of $n^{\bar{\kappa}+2}$ halfspaces. Of course, each of these halfspaces is defined by a hyperplane. Let $\mathcal{H}$ be the union of all $n^{\bar{\kappa}+2}$ hyperplanes over all $n^{\bar{\kappa}}$ trees. We know that for any connected component R of $[0, 1]^d \setminus \mathcal{H}$, the search tree B&B builds using the scoring rule μ_1`score`$_1 + \cdots + \mu_d$`score`$_d$ is invariant across all $(\mu_1, \ldots, \mu_d) \in R$, so the lemma statement holds. $\square$

In the same way Lemma 3.3 implies the pseudo-dimension bound in Theorem 3.7, Lemma 3.8 also implies a pseudo-dimension bound. The proof is similar to that of Theorem 3.7. See Theorem E.7 in Appendix E.2.1 which proves the theorem for a more general tree search algorithm.

Theorem 3.9. *Let* `cost` *be a tree-constant cost function and let* `score`$_1, \ldots,$ `score`$_d$ *be d arbitrary scoring rules. Suppose we limit B&B to producing search trees of size $\bar{\kappa}$. Let $\mathcal{C}$ be the set of functions $\{$`cost`$(\cdot, \mu$`score`$_1 + \cdots + \mu_d$`score`$_d) : (\mu_1, \ldots, \mu_d) \in [0, 1]^d\}$. Then $\mathrm{Pdim}(\mathcal{C}) = O\left(d\bar{\kappa} \log n + d \log d\right)$.*

16

Naturally, if the function `cost` measures the size of the search tree B&B returns capped at some value κ as described in Section 3.1, we obtain the following corollary.

Corollary 3.10. *Let `cost` be a tree-constant cost function, let $score_1, \ldots, score_d$ be d arbitrary scoring rules, and let $\kappa \in \mathbb{N}$ be an arbitrary tree size bound. Suppose that for any problem instance Q, $cost(Q, \mu score_1 + \cdots + \mu_d score_d)$ equals the minimum of the following two values: 1) The number of nodes in the search tree B&B builds using the scoring rule $\mu score_1 + \cdots + \mu_d score_d$ on input Q; and 2) κ. For any distribution $\mathcal{D}$ over problem instances Q with at most n variables, with probability at least $1 - \delta$ over the draw $\{Q_1, \ldots, Q_m\} \sim \mathcal{D}^m$, for any $\mu \in [0, 1]$,*

$$\left| \mathbb{E}_{Q \sim \mathcal{D}}[cost(Q, \mu score_1 + \cdots + \mu_d score_d)] - \frac{1}{m} \sum_{i=1}^{m} cost(Q_i, \mu score_1 + \cdots + \mu_d score_d) \right|$$

$$= O\left(\sqrt{\frac{d\kappa^2(\kappa \log n + \log d)}{m}} + \kappa\sqrt{\frac{\ln(1/\delta)}{m}} \right).$$

3.3.3 Learning algorithm

For the case where we wish to learn the optimal tradeoff between two scoring rules, we provide an algorithm in Appendix C that finds the empirically optimal parameter $\hat{\mu}$ given a sample of m problem instances. By Theorems 3.7, 3.9, and 3.6, we know that so long as m is sufficiently large, $\hat{\mu}$ is nearly optimal in expectation. Our algorithm stems from the observation that for any tree-constant cost function `cost` and any problem instance Q, $\texttt{cost}(Q, \mu score_1 + (1 - \mu)score_2)$ is simple: it is a piecewise-constant function of μ with a finite number of pieces. This is the same observation that we prove in Lemma 3.8. Given a sample of problem instances $\mathcal{S} = \{Q_1, \ldots, Q_m\}$, our ERM algorithm constructs all m piecewise-constant functions, takes their average, and finds the minimizer of that function. In practice, we find that the number of pieces making up this piecewise-constant function is small, so our algorithm can learn over a training set of many problem instances.

4 Experiments

In this section, we show that the parameter of the variable selection rule in B&B algorithms for MILP can have a dramatic effect on the average tree size generated for several domains, and no parameter value is effective across the multiple natural distributions.

Experimental setup. We use the C API of IBM ILOG CPLEX 12.8.0.0 to override the default variable selection rule using a branch callback. Additionally, our callback performs extra book-keeping to determine a finite set of values for the parameter that give rise to *all* possible B&B trees for a given instance (for the given choice of branching rules that our algorithm is learning to weight). This ensures that there are no good or bad values for the parameter that get skipped; such skipping could be problematic according to our theory in Section 3.2. We run CPLEX exactly once for each possible B&B tree on each instance. Following Khalil et al. [2016], Fischetti and Monaci [2012], and Karzan et al. [2009], we disable CPLEX's cuts and primal heuristics, and we also disable its root-node preprocessing. The CPLEX node selection policy is set to "best bound" (aka. A^* in AI), which is the most typical choice in MILP. All experiments are run on a cluster of 64 c3.large Amazon AWS instances.
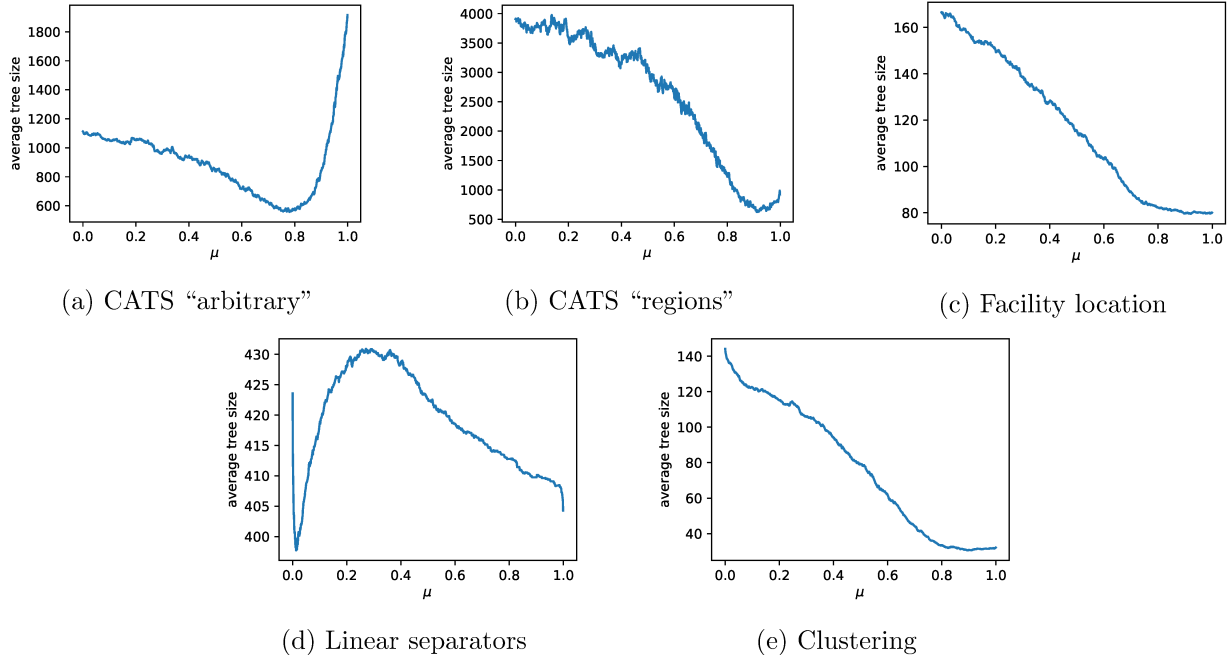
(a) CATS "arbitrary"

(b) CATS "regions"

(c) Facility location

(d) Linear separators

(e) Clustering

Figure 6: The average tree size produced by B&B when run with the linear scoring rule with parameter μ.

For each of the following application domains, Figure 6 shows the average B&B tree size produced for each possible value of the μ parameter for the linear scoring rule, averaged over m independent samples from the distribution.

Combinatorial auctions. We generate $m = 100$ instances of the combinatorial auction winner determination problem under the OR-bidding language [Sandholm, 2002], which makes this problem equivalent to weighted set packing. The problem is NP-complete. We encode each instance as a binary MILP (see Example 2.1). We use the Combinatorial Auction Test Suite (CATS) [Leyton-Brown et al., 2000] to generate these instances. We use the "arbitrary" generator with 200 bids and 100 goods and "regions" generator with 400 bids and 200 goods.

Facility location. Suppose there is a set I of customers and a set J of facilities that have not yet been built. The facilities each produce the same good, and each consumer demands one unit of that good. Consumer i can obtain some fraction y_{ij} of the good from facility j, which costs them $d_{ij}y_{ij}$. Moreover, it costs f_j to construct facility j. The goal is to choose a subset of facilities to construct while minimizing total cost. In Appendix D we show how to formulate facility location as a MILP. We generate $m = 500$ instances with 70 facilities and 70 customers each. Each cost d_{ij} is uniformly sampled from $[0, 10^4]$ and each cost f_j is uniformly sampled from $[0, 3 \cdot 10^3]$. This distribution has regularly been used to generate benchmark sets for facility location [Gilpin and Sandholm, 2011, Alekseeva et al., 2015, Goldengorin et al., 2011, Kochetov and Ivanenko, 2005, Homberger and Gehring, 2008].

Clustering. Given n points $P = \{p_1, \ldots, p_n\}$ and pairwise distances $d(p_i, p_j)$ between each pair of points p_i and p_j, the goal of k-means clustering is to find k *centers* $C = \{c_1, \ldots, c_k\} \subseteq P$ such that the following objective function is minimized: $\sum_{i=1}^{n} \min_{j \in [k]} d(p_i, c_j)^2$. In Appendix D,

we show how to formulate this problem as a MILP. We generate $m = 500$ instances with 35 points each and $k = 5$. We set $d(i, i) = 0$ for all i and choose $d(i, j)$ uniformly at random from $[0, 1]$ for $i \neq j$. These distances do not satisfy the triangle inequality and they are not symmetric (i.e., $d(i, j) \neq d(j, i)$), which tends to lead to harder MILP instances than using Euclidean distances between randomly chosen points in $\mathbb{R}^d$.

Agnostically learning linear separators. Let $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N$ be N points in $\mathbb{R}^d$ labeled by $z_1, \ldots, z_N \in \{-1, 1\}$. Suppose we wish to learn a linear separator $\boldsymbol{w} \in \mathbb{R}^d$ that minimizes 0-1 loss, i.e., $\sum_{i=1}^{N} \mathbf{1}_{\{z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle < 0\}}$. In Appendix D, we show how to formulate this problem as a MILP. We generate $m = 500$ problem instances with 50 points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{50}$ from the 2-dimensional standard normal distribution. We sample the true linear separator $\boldsymbol{w}^*$ from the 2-dimensional standard Gaussian distribution and label point $\boldsymbol{p}_i$ by $z_i = \text{sign}(\langle \boldsymbol{w}^*, \boldsymbol{p}_i \rangle)$. We then choose 10 random points and flip their labels so that there is no consistent linear separator.

Experimental results. The relationship between the variable selection parameter and the average tree size varies greatly from application to application. This implies that the parameters should be tuned on a per-application basis, and that no parameter value is universally effective. In particular, the optimal parameter for the "regions" combinatorial auction problem, facility location, and clustering is close to 1. However, that value is severely suboptimal for the "arbitrary" combinatorial auction domain, resulting in trees that are three times the size of the trees obtained under the optimal parameter value.

Data-dependent guarantees. We now explore data-dependent generalization guarantees. To prove our worst-case guarantee Theorem 3.7, we show in Lemma 3.3 that for any MILP instance over n binary variables, there are $T \leq 2^{n(n-1)/2} n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, across all $\mu \in I_j$, the scoring rule $\mu \texttt{score}_1 + (1 - \mu) \texttt{score}_2$ results in the same B&B tree. Our generalization guarantees grow logarithmically in the number of intervals. In practice, we find that the number of intervals partitioning $[0, 1]$ is much smaller than $2^{n(n-1)/2} n^n$. In this section, we take advantage of this data-dependent simplicity to derive stronger generalization guarantees when the number of intervals partitioning $[0, 1]$ is small. To do so, we move from pseudo-dimension to Rademacher complexity [Bartlett and Mendelson, 2002, Koltchinskii, 2001] since Rademacher complexity implies distribution-dependent guarantees whereas pseudo-dimension implies generalization guarantees that are worst-case over the distribution. We now define empirical Rademacher complexity.

Definition 4.1 (Empirical Rademacher complexity). Let $\mathcal{F}$ be a class of functions mapping an abstract domain $\mathcal{Z}$ to the set $[-\kappa, \kappa]$. The *empirical Rademacher complexity* of $\mathcal{F}$ with respect to a sample $\mathcal{S} = \{z_1, \ldots, z_m\} \subseteq \mathcal{Z}$ of size m is defined as

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma} \sim \{-1, 1\}^m} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma[i] f(z_i) \right].$$

The following *generalization guarantee* based on Rademacher complexity is well-known.

Theorem 4.1. *For any distribution $\mathcal{D}$ over $\mathcal{Z}$, with probability at least $1 - \delta$ over the draw of $\mathcal{S} = \{z_1, \ldots, z_m\} \sim \mathcal{D}^m$, for all $f \in \mathcal{F}$, $\left| \mathbb{E}_{z \sim \mathcal{D}}[f(z)] - \frac{1}{m} \sum_{i=1}^{m} f(z_i) \right| \leq 2\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) + 4\kappa \sqrt{\frac{2}{m} \ln \frac{4}{\delta}}$.*

To obtain data-dependent generalization guarantees, we rely on the following theorem, which follows from truncating the proof of Massart's finite lemma [Massart, 2000], as observed by Riondato

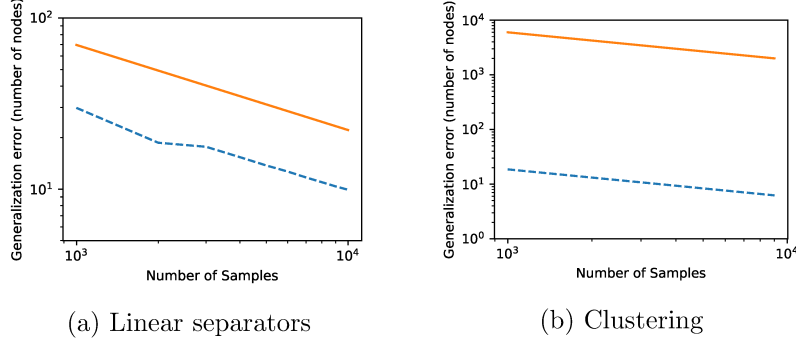(a) Linear separators   (b) Clustering

Figure 7: Data-dependent generalization guarantees (the dotted lines) and worst-case generalization guarantees (the solid lines). Figure 7a displays generalization guarantees for linear separators when there are 10 points drawn from the 2-dimensional standard normal distribution, the true classifier is drawn from the 2-dimensional standard normal distribution, and 4 random points have their label flipped. Figure 7b shows generalization guarantees for k-means clustering MILPs when there are 35 points drawn from the 5-dimensional standard normal distribution and $k = 3$. In both plots, we cap the tree size at 150 nodes.

and Upfal [2015]. For a function class $\mathcal{F}$ and a set $\mathcal{S} = \{z_1, \ldots, z_m\} \subseteq \mathcal{Z}$, we use the notation $\mathcal{F}(\mathcal{S})$ to denote the set of vectors $\{(f(z_1), \ldots, f(z_m)) : f \in \mathcal{F}\}$.

Theorem 4.2. *For a sample $\mathcal{S}$ of size m, suppose that $\mathcal{F}(\mathcal{S})$ is finite. Then*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) \leq \inf_{\lambda > 0} \frac{1}{\lambda} \log \left(\sum_{\boldsymbol{a} \in \mathcal{F}(\mathcal{S})} \exp \left(\frac{1}{2} \left(\frac{\lambda \|\boldsymbol{a}\|_2}{m} \right)^2 \right) \right).$$

Figure 7 shows a comparison of the worst-case versus data-dependent guarantees. The latter are significantly better. For ease of comparison, we compare the data-dependent bound provided by Theorem 4.2 with a variation of Theorem 3.7 in terms of Rademacher complexity. See Theorem D.1 in Appendix D for details.

5   Constraint satisfaction problems

In this section, we describe tree search for constraint satisfaction problems. The generalization guarantee from Section 3.3 also applies to tree search in this domain, as we describe in Appendix E.

A constraint satisfaction problem (CSP) is a tuple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of domains where D_i is the set of values variable x_i can take on, and C is a set of constraints between variables. Each constraint in C is a pair $((x_{i_1}, \ldots, x_{i_r}), \psi)$ where ψ is a function mapping $D_{i_1} \times \cdots \times D_{i_r}$ to $\{0, 1\}$ for some $r \in [n]$ and some $i_1, \ldots, i_r \in [n]$. Given an assignment $(y_1, \ldots, y_n) \in D_1 \times \cdots \times D_n$ of the variables in X, a constraint $((x_{i_1}, \ldots, x_{i_r}), \psi)$ is satisfied if $\psi(y_{i_1}, \ldots, y_{i_r}) = 1$. The goal is to find an assignment that maximizes the number of satisfied constraints.

The *degree* of a variable x, denoted $\deg(x)$, is the number of constraints involving x. The *dynamic degree* of (an unassigned variable) x given a partial assignment $\boldsymbol{y}$, denoted $\operatorname{ddeg}(x, \boldsymbol{y})$ is the number of constraints involving x and at least one other unassigned variable.

20

Figure 8: Illustrations of Example 5.1.

Example 5.1 (Graph k-coloring)**.** Given a graph, the goal of this problem is to color its vertices using at most k colors such that no two adjacent vertices share the same color. This problem can be formulated as a CSP, as illustrated by the following example. Suppose we want to 3-color the graph in Figure 8a using pink, green, and orange. The four vertices correspond to the four variables $X = \{x_1, \ldots, x_4\}$. The domain $D_1 = \cdots = D_4 = \{\text{pink, green, orange}\}$. The only constraints on this problem are that no two adjacent vertices share the same color. Therefore we define ψ to be the "not equal" relation mapping $\{\text{pink, green, orange}\} \times \{\text{pink, green, orange}\} \to \{0, 1\}$ such that $\psi(\omega_1, \omega_2) = \mathbf{1}_{\{\omega_1 \neq \omega_2\}}$. Finally, we define the set of constraints to be

$$C = \{((x_1, x_2), \psi), ((x_1, x_3), \psi), ((x_2, x_3), \psi), ((x_3, x_4), \psi))\,.$$

See Figure 8b for a coloring that satisfies all constraints ($y_1 = y_4 = $ green, $y_2 = $ orange, and $y_3 = $ pink).

5.0.1 CSP tree search

CSP tree search begins by choosing a variable x_i with domain D_i and building $|D_i|$ branches, each one corresponding to one of the $|D_i|$ possible value assignments of x. Next, a node Q of the tree is chosen, another variable x_j is chosen, and $|D_j|$ branches from Q are built, each corresponding to the possible assignments of x_j. The search continues and a branch is pruned if any of the constraints are not feasible given the partial assignment of the variables from the root to the leaf of that branch.

5.0.2 Variable selection in CSP tree search

As in MILP tree search, there are many variable selection policies researchers have suggested for choosing which variable to branch on at a given node. Typically, algorithms associate a score for branching on a given variable x_i at node Q in the tree $\mathcal{T}$, as in B&B. The algorithm then branches on the variable with the highest score. We provide several examples of common variable selection policies below.

deg/dom and ddeg/dom [Bessiere and Régin, 1996]: deg/dom corresponds to the scoring rule $\texttt{score}(\mathcal{T}, Q, i) = \deg(x_i)/|D_i|$ and ddeg/dom corresponds to the scoring rule $\texttt{score}(\mathcal{T}, Q, i) = \text{ddeg}(x_i, \boldsymbol{y})/|D_i|$, where $\boldsymbol{y}$ is the assignment of variables from the root of $\mathcal{T}$ to Q.

Smallest domain [Haralick and Elliott, 1980]: In this case, $\texttt{score}(\mathcal{T}, Q, i) = 1/|D_i|$.

Our theory is for tree search and applies to both MILPs and CSPs. It applies both to lookahead approaches that require learning the weighting of the two children (the more promising and less promising child) and to approaches that require learning the weighting of several different scoring rules.

21

6 Conclusions and broader applicability

In this work, we studied machine learning for tree search algorithm configuration. We showed how to learn a nearly optimal mixture of branching (e.g., variable selection) rules. Through experiments, we showed that using the optimal parameter for one application domain when solving problem instances from a different application domain can lead to a substantial tree size blow up. We proved that this blowup can even be exponential. We provided the first sample complexity guarantees for tree search algorithm configuration. With only a small number of samples, the empirical cost incurred by using any mixture of two scoring rules will be close to its expected cost, where cost is an abstract measure such as tree size. We showed that using empirical Rademacher complexity, these bounds can be significantly tightened further. Through theory and experiments, we showed that learning to branch is practical and hugely beneficial.

While we presented the theory in the context of tree search, it also applies to other tree-growing applications. For example, it could be used for learning rules for selecting variables to branch on in order to construct small *decision trees* that correctly classify training examples. Similarly, it could be used for learning to branch in order to construct a desirable *taxonomy* of items represented as a tree—for example for representing customer segments in advertising day to day.

References

Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, January 2005.

Ekaterina Alekseeva, Yury Kochetov, and Alexandr Plyasunov. An exact method for the discrete $(r|p)$-centroid problem. *Journal of Global Optimization*, 63(3):445–460, 2015.

Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

Amine Balafrej, Christian Bessiere, and Anastasia Paparrizou. Multi-armed bandits for adaptive constraint propagation. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. *Conference on Learning Theory (COLT)*, 2017.

Peter Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

Evelyn Beale. Branch and bound methods for mathematical programming systems. *Annals of Discrete Mathematics*, 5:201–219, 1979.

Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

Christian Bessiere and Jean-Charles Régin. Mac and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 61–75. Springer, 1996.

Olivier Chapelle, Vikas Sindhwani, and S Sathiya Keerthi. Branch and bound for semi-supervised support vector machines. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 217–224, 2007.

Hanjun Dai, Elias Boutros Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

Giovanni Di Liberto, Serdar Kadioglu, Kevin Leo, and Yuri Malitsky. Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 248(3):943–953, 2016.

Matteo Fischetti and Michele Monaci. Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164, 2012.

J-M Gauthier and Gerard Ribière. Experiments in mixed-integer linear programming using pseudo-costs. *Mathematical Programming*, 12(1):26–47, 1977.

Andrew Gilpin and Tuomas Sandholm. Information-theoretic approaches to branching in search. *Discrete Optimization*, 8(2):147–159, 2011. Early version in IJCAI-07.

Boris Goldengorin, John Keane, Viktor Kuzmenko, and Michael Tso. Optimal supplier choice with discounting. *Journal of the Operational Research Society*, 62(4):690–699, 2011.

Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

Robert Haralick and Gordon Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2014.

Jorg Homberger and Hermann Gehring. A two-level parallel genetic algorithm for the uncapacitated warehouse location problem. In *Hawaii International Conference on System Sciences*, pages 67–67. IEEE, 2008.

Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009. ISSN 1076-9757.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, pages 507–523, 2011.

Robert G Jeroslow. Trivial integer programs unsolvable by branch-and-bound. *Mathematical Programming*, 6(1):105–109, 1974.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142. ACM, 2002.

Jörg Hendrik Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Towards efficient and exact map-inference for large scale discrete computer vision problems via combinatorial optimization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1752–1758. IEEE, 2013.

Fatma Kılınç Karzan, George L Nemhauser, and Martin Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1(4):249–293, 2009.

Elias Boutros Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Elias Boutros Khalil, Bistra Dilkina, George Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Yuri Kochetov and Dmitry Ivanenko. Computationally difficult instances for the uncapacitated facility location problem. In *Metaheuristics: Progress as real problem solvers*, pages 351–367. Springer, 2005.

Iasonas Kokkinos. Rapid deformable object detection using dual-tree branch-and-bound. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pages 2681–2689, 2011.

Peter J Kolesar. A branch and bound algorithm for the knapsack problem. *Management science*, 13(9):723–735, 1967.

Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. Clustering via LP-based stabilities. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*. 2009.

Markus Kruber, Marco E Lübbecke, and Axel Parmentier. Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Springer, 2017.

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

Pierre Le Bodic and George Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, pages 1–37, 2017.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 66–76, Minneapolis, MN, 2000.

Paolo Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1-2):315–326, 2000.

Jeff Linderoth and Martin Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal of Computing*, 11:173–187, 1999.

Andrea Lodi and Giulia Zarpellon. On learning and branching: a survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 25(2):207–236, 2017.

Pascal Massart. Some applications of concentration inequalities to statistics. *Annales de la Faculté des Sciences de Toulouse*, 9:245–303, 2000.

David Pollard. *Convergence of Stochastic Processes*. Springer, 1984.

Matteo Riondato and Eli Upfal. Mining frequent itemsets through progressive sampling with Rademacher averages. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with UCT. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2017.

Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, January 2002.

Tuomas Sandholm. Very-large-scale generalized combinatorial multi-attribute auctions: Lessons from conducting \$60 billion of sourcing. In Zvika Neeman, Alvin Roth, and Nir Vulkan, editors, *Handbook of Market Design*. Oxford University Press, 2013.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.

Wei Xia and Roland Yap. Learning robust search strategies using a bandit-based approach. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

A Additional related work

As in this work, Khalil et al. [2016] study variable selection policies. Their goal is to find a variable selection strategy that mimics the behavior of the classic branching strategy known as *strong branching* while running faster than strong branching. They split a single MILP instance into a training and test set. The beginning of the B&B algorithm is the training phase: their algorithm assigns features to each node and observes the variable selected to branch on by strong branching using the product scoring rule (see the definition in Section 2.1.2). Collecting these features and variable observations for a pre-specified number of nodes, they thus amass a training set. Their algorithm then uses this training set together with a classic ranking algorithm from the machine learning literature [Joachims, 2002] to determine a function f that, ideally, will imitate strong branching. In a bit more detail, on each node the algorithm encounters as it explores the rest of the tree, the algorithm assigns features ϕ to that node, evaluates the function $f(\phi)$ which returns a variable to branch on, and the algorithm branches on that variable. The hope is that $f(\phi)$ matches the variable strong branching would choose to branch on at that node.

Other works have explored the use of machine learning techniques in the context of other aspects of B&B beyond variable selection. For example, He et al. [2014] use machine learning to speed up branch-and-bound, focusing on speeding up the node selection policy. Their work does not provide any learning-theoretic guarantees. Beyond node selection, He et al. [2014] also aim to learn a *pruning policy*, which determines whether to fathom a given node. This policy may occasionally fathom a node even when the branch contains the optimal solution, but ideally, this will happen infrequently or the resulting solution will be nearly optimal. Their learning algorithm takes as input a set of MIP instances and their optimal solutions. For each MIP in this set, the algorithm constructs a simple *oracle* which returns, given a set of B&B leaf nodes as input, the leaf whose feasible set contains the optimal solution. The learning algorithm uses the oracle to train the node selection and pruning policies using *imitation learning*. Ideally, the resulting node selection policy should choose the nodes the oracle selects, and the pruning policy should prune any node that does not contain the optimal solution in its feasible set.

B Variable selection policies

Algorithm 2 Generic variable selection

Input: Current subproblem Q.
  1: Let $F = \{j \in I \mid \breve{x}_Q[j] \notin \mathbb{Z}\}$ be the set of candidate variables.
  2: For all candidates $i \in F$, calculate a score value $\texttt{score}(Q, i)$.
Output: $\operatorname{argmax}_i\{\texttt{score}(Q, i)\}$.

C Proofs from Section 3

Notation. In the proofs in this section, we will use the following notation. Let Q be a MILP instance. Suppose that we branch on x_i and x_j, setting $x_i = 0$ and $x_j = 1$. We use the notation $Q_{i,j}^{-;+}$ to denote the resulting MILP. Similar, if we set $x_i = 1$ and $x_j = 0$, we denote the resulting MILP as $Q_{i,j}^{+;-}$.

Theorem 3.1. *Let*

$$\textbf{\textit{score}}_1(\mathcal{T}, Q, i) = \min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, \; \textbf{\textit{score}}_2(\mathcal{T}, Q, i) = \max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\},$$

and $\textbf{\textit{cost}}(Q, \mu \textbf{\textit{score}}_1 + (1-\mu)\textbf{\textit{score}}_2)$ be the size of the tree produced by B&B. For every a, b such that $\frac{1}{3} < a < b < \frac{1}{2}$ and for all even $n \geq 6$, there exists an infinite family of distributions $\mathcal{D}$ over MILP instances with n variables such that if $\mu \in [0, 1] \setminus (a, b)$, then

$$\mathbb{E}_{Q \sim \mathcal{D}}\left[\textbf{\textit{cost}}\left(Q, \mu \textbf{\textit{score}}_1 + (1-\mu)\textbf{\textit{score}}_2\right)\right] = \Omega\left(2^{(n-9)/4}\right)$$

and if $\mu \in (a, b)$, then with probability 1, $\textbf{\textit{cost}}\left(Q, \mu \textbf{\textit{score}}_1 + (1-\mu)\textbf{\textit{score}}_2\right) = O(1)$. This holds no matter which node selection policy B&B uses.

Proof. We populate the support of the distribution $\mathcal{D}$ by relying on two helpful theorems: Theorem 3.2 and C.4. In Theorem 3.2, we prove that for all $\mu^* \in \left(\frac{1}{3}, \frac{2}{3}\right)$, there exists an infinite family $\mathcal{F}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{F}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\textsf{score}_1 + (1-\mu)\textsf{score}_2$ results in a B&B tree with $O(1)$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results a tree with $2^{(n-4)/2}$ nodes. Conversely, in Theorem C.4, we prove that there exists an infinite family $\mathcal{G}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{G}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\textsf{score}_1 + (1-\mu)\textsf{score}_2$ results in a B&B tree with $2^{(n-5)/4}$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results a tree with $O(1)$ nodes.

Now, let Q_a be an arbitrary instance in $\mathcal{G}_{n,a}$ and let Q_b be an arbitrary instance in $\mathcal{F}_{n,b}$. The theorem follows by letting $\mathcal{D}$ be a distribution such that $\Pr_{Q \sim \mathcal{D}}[Q = Q_a] = \Pr_{Q \sim \mathcal{D}}[Q = Q_b] = 1/2$. We know that if $\mu \in [0, 1] \setminus (a, b)$, then the expected value of $\textsf{cost}\left(Q, \mu\textsf{score}_1 + (1-\mu)\textsf{score}_2\right)$ is $\frac{1}{2}\left(O(1) + 2^{(n-5)/4}\right) \geq 2^{(n-9)/4}$. Meanwhile, if $\mu \in (a, b)$, then with probability 1,

$$\textsf{cost}\left(Q, \mu\textsf{score}_1 + (1-\mu)\textsf{score}_2\right) = O(1).$$

Throughout the proof of this theorem, we assume the node selection policy is depth-first search. We then prove that for any infeasible MILP, if NSP and NSP' are two node selection policies and $\textsf{score} = \mu\textsf{score}_1 + (1-\mu)\textsf{score}_2$ for any $\mu \in [0, 1]$, then tree $\mathcal{T}$ B&B builds using NSP and $\textsf{score}$ equals the tree $\mathcal{T}'$ it builds using NSP' and $\textsf{score}$ (see Theorem C.9). Thus, the theorem holds for any node selection policy. $\square$

Theorem 3.2. *Let*

$$\textbf{\textit{score}}_1(\mathcal{T}, Q, i) = \min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, \; \textbf{\textit{score}}_2(\mathcal{T}, Q, i) = \max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\},$$

and $\textbf{\textit{cost}}(Q, \mu\textbf{\textit{score}}_1 + (1-\mu)\textbf{\textit{score}}_2)$ be the size of the tree produced by B&B. For all even $n \geq 6$ and all $\mu^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, there exists an infinite family $\mathcal{F}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{F}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\textbf{\textit{score}}_1 + (1-\mu)\textbf{\textit{score}}_2$ results in a B&B tree with $O(1)$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results a tree with $2^{(n-4)/2}$ nodes.

Proof. For ease of notation in this proof, we will drop $\mathcal{T}$ from the input of the functions $\textsf{score}_1$ and $\textsf{score}_2$ since the scoring rules do not depend on $\mathcal{T}$, they only depend on the input MILP instance and variable.

For any constant $\gamma \geq 1$, let $\boldsymbol{c}_1 = \gamma(1, 2, \ldots, n-3)$ and let $\boldsymbol{c}_2 = \gamma\left(0, 1.5, 3 - \frac{1}{2\mu^*}\right)$. Let $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2) \in \mathbb{R}^n$ be the concatenation of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Next, define the n-dimensional vectors

27

$a_1 = 2\sum_{i=1}^{n-3} e_i$ and $a_2 = 2\sum_{i=1}^{3} e_{n-3+i}$, and let A be a matrix whose first row is a_1 and second row is a_2. Let $Q_{\gamma,n}$ be the MILP

$$
\begin{aligned}
\text{maximize} \quad & c \cdot x \\
\text{subject to} \quad & Ax = (n-3,3)^\top \\
& x \in \{0,1\}^n.
\end{aligned}
$$

We define $\mathcal{F}_{n,\mu^*} = \{Q_{n,\gamma} : \gamma \geq 1\}$.

Example C.1. If $\gamma = 1$ and $n = 8$, then $Q_{\gamma,n}$ is

$$
\begin{aligned}
\text{maximize} \quad & \left(1,2,3,4,5,0,1.5,3-\tfrac{1}{2\mu^*}\right) \cdot x \\
\text{subject to} \quad & \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \end{pmatrix} x = \begin{pmatrix} 5 \\ 3 \end{pmatrix} \\
& x \in \{0,1\}^8.
\end{aligned}
$$

For every even $n \geq 6$, both of the constraints $a_1 \cdot x = 2\sum_{i=1}^{n-3} x_i = n-3$ and $a_2 \cdot x = 2\sum_{i=1}^{3} x_{n-3+i} = 3$ are infeasible for $x \in \{0,1\}^n$ since $2\sum_{i=1}^{n-3} x_i$ and $2\sum_{i=1}^{3} x_{n-3+i}$ are even numbers but 3 and $n-3$ are odd numbers. The key idea of this proof is that if branch-and-bound branches on all variables in $\{x_{n-2}, x_{n-1}, x_n\}$ first, it will terminate upon making a tree of size at most $2^3 = 8$, since at most three branches are necessary to determine that $a_1 \cdot x = 2\sum_{i=1}^{3} x_{n-3+i} = 3$ is infeasible. However, if branch-and-bound branches on all variables in $\{x_1, \ldots, x_{n-3}\}$ first, it will create a tree with exponential size before it terminates.

Lemma C.1. *Suppose $\mu < \mu^*$. Then for any MILP $Q_{\gamma,n} \in \mathcal{F}_{n,\mu^*}$, $\mu score_1 + (1-\mu) score_2$ branches on all variables in $\{x_{n-2}, x_{n-1}, x_n\}$ before branching on variables in $\{x_1, \ldots, x_{n-3}\}$.*

Proof of Lemma C.1. For ease of notation, for the remainder of this proof, we drop the subscript (γ, n) from $Q_{\gamma,n}$ and denote this MILP as Q. We first need to determine the form of $\breve{x}_Q$, which is the optimal solution to the LP relaxation of Q. It is easiest to see how the LP relaxation will set the variables x_{n-2}, x_{n-1}, and x_n. The only constraints on these variables are that $2(x_{n-2}+x_{n-1}+x_n) = 3$ and that $x_{n-2}, x_{n-1}, x_n \in [0,1]$. Recall that $c = \left(1,2,\ldots,n-3,0,1.5,3-\tfrac{1}{2\mu^*}\right)$ is the vector defining the objective value of Q. Since $\mu^* > 1/3$, we know that $1.5 < 3 - \tfrac{1}{2\mu^*}$, which means that $c[n-2] < c[n-1] < c[n]$. Since the goal is to maximize $c \cdot x$, the LP relaxation will set $x_{n-2} = 0$, $x_{n-1} = \tfrac{1}{2}$, and $x_n = 1$. The logic for the first $n-3$ variables is similar. In this case, $c[1] < c[2] < \cdots < c[n-3]$, so the LP relaxation's solution will put as much weight as possible on the variable x_{n-3}, then as much weight as possible on the variable x_{n-4}, and so on, putting as little weight as possible on the variable x_1 since it has the smallest corresponding objective coefficient $c[1]$. Since the only constraints on these variables are that $2\sum_{i=1}^{n-3} x_i = n-3$ and $x_1, \ldots, x_{n-3} \in [0,1]$, the LP objective value can set $\left\lfloor \frac{n-3}{2} \right\rfloor$ of the variables to 1, it can set one variable to $\tfrac{1}{2}$, and it has to set the rest of the variables to 0. Letting $i = \left\lceil \frac{n-3}{2} \right\rceil$, this means the LP relaxation will set the first $i-1$ variables $x_1, \cdots x_{i-1}$ to zero, it will set $x_i = \tfrac{1}{2}$, and it will set $x_{i+1}, \ldots, x_{n-3}$ to 1. In other words,

$$
\breve{x}_Q[j] = \begin{cases} 0 & \text{if } j \leq \lfloor (n-3)/2 \rfloor \text{ or } j = n-2 \\ \tfrac{1}{2} & \text{if } j = \lceil (n-3)/2 \rceil \text{ or } j = n-1 \\ 1 & \text{if } \lceil (n-3)/2 \rceil \leq j \leq n-3 \text{ or } j = n. \end{cases}
$$

For example, if $n = 8$, then $\breve{x}_Q = \left(0,0,\tfrac{1}{2},1,1,0,\tfrac{1}{2},1\right)$. (See Figure 9a.) Therefore, the only candidate variables to branch on are x_{n-1} and x_i where again, $i = \lceil (n-3)/2 \rceil$.
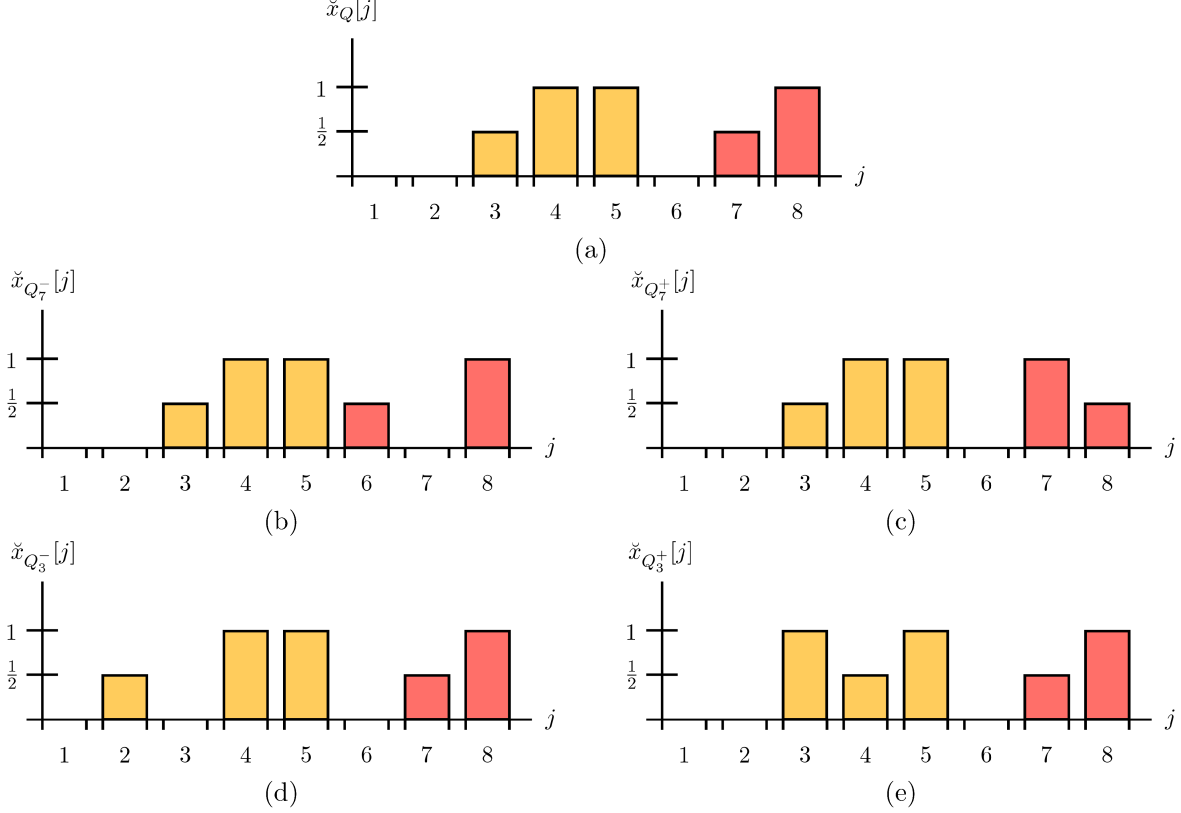
28

Figure 9: Illustrations to accompany the proof of Lemma C.1 when $n = 8$. For each j on the x-axis, the histogram gives the value of either $\breve{x}_Q[j]$ (Figure 9a), $\breve{x}_{Q_7^-}[j]$ (Figure 9b), $\breve{x}_{Q_7^+}[j]$ (Figure 9c), $\breve{x}_{Q_3^-}[j]$ (Figure 9d), or $\breve{x}_{Q_3^+}[j]$ (Figure 9e). In this case, $i = 3$.

To determine when variable B&B will branch on, we need to calculate $\breve{\boldsymbol{x}}_{Q_i^-}$ which is the solution to the LP relaxation of Q with the additional constraint that $x_i = 0$, as well as $x_{Q_i^+}$ which is the solution to the LP relaxation of Q with the additional constraint that $x_i = 1$, and $x_{Q_{n-1}^-}$ and $x_{Q_{n-1}^+}$. First, we will determine the form of the vectors $\breve{\boldsymbol{x}}_{Q_{n-1}^-}$ and $\breve{\boldsymbol{x}}_{Q_{n-1}^+}$. Suppose we set $x_{n-1} = 0$. We now need to ensure that $2(x_{n-2} + 0 + x_n) = 3$ and that $x_{n-2}, x_n \in [0, 1]$. Since $c[n-2] = 0 < 3 - \frac{1}{2\mu^*} = c[n]$, the solution to the LP relaxation of Q_{n-1}^- will set $x_{n-2} = \frac{1}{2}$ and $x_n = 1$. (See Figure 9b.) Similarly, if we set $x_{n-1} = 1$, we now need to ensure that $2(x_{n-2} + 1 + x_n) = 3$ and that $x_{n-2}, x_n \in [0, 1]$. Therefore, the solution to the LP relaxation of Q_{n-1}^- will set $x_{n-2} = 0$ and $x_n = \frac{1}{2}$. (See Figure 9c.) In other words, $\breve{\boldsymbol{x}}_{Q_{n-1}^-} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\breve{\boldsymbol{x}}_{Q_{n-1}^+} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$.

The argument for $\breve{\boldsymbol{x}}_{Q_i^-}$ and $\breve{\boldsymbol{x}}_{Q_i^-}$ is similar. Recall that in $\breve{\boldsymbol{x}}_Q$, the solution to the LP relaxation of the original MIP Q, we have that $\breve{x}_Q[i] = \frac{1}{2}$ since it is the median of the variables $x_1, \ldots, x_{n-3}$. For all $j > i$, we have that $\breve{x}_Q[j] = 1$ and for all $j < i$, we have that $\breve{x}_Q[j] = 0$. Suppose we set $x_i = 0$. As before, the LP relaxation's solution will put as much weight as possible on the variable x_{n-3}, then as much weight as possible on the variable x_{n-4}, and so on, putting as little weight as possible on the variable x_1 since it has the smallest corresponding objective coefficient $c[1]$. Since it cannot set $x_i = \frac{1}{2}$, it will set the next-best variable to $\frac{1}{2}$, which is x_{i-1}. (See Figure 9d.) In other words, $\breve{\boldsymbol{x}}_{Q_i^-} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_i + \frac{1}{2}\boldsymbol{e}_{i-1}$. If we set $x_i = 1$, the LP relaxation's solution will have to take some weight away from the variables $x_{i+1}, \ldots, x_{n-3}$ since it needs to ensure that $2\sum_{i=1}^{n-3} x_i = n-3$.

29

Therefore, it will set x_{i+1} to $\frac{1}{2}$ and x_j to 1 for all $j > i+1$. (See Figure 9e.) In other words, $\breve{x}_{Q_i^+} = \breve{x}_Q + \frac{1}{2}e_i - \frac{1}{2}e_{i+1}$.

Therefore,

$$\breve{c}_{Q_i^-} = \breve{c}_Q - \frac{1}{2}\left\lceil \frac{n-3}{2} \right\rceil + \frac{1}{2}\left\lfloor \frac{n-3}{2} \right\rfloor = \breve{c}_Q - \frac{\gamma}{2},$$

$$\breve{c}_{Q_i^+} = \breve{c}_Q + \frac{1}{2}\left\lceil \frac{n-3}{2} \right\rceil - \frac{1}{2}\left(\left\lceil \frac{n-3}{2} \right\rceil + 1\right) = \breve{c}_Q - \frac{\gamma}{2},$$

$$\breve{c}_{Q_{n-1}^-} = \breve{c}_Q - \frac{3\gamma}{4}, \text{ and}$$

$$\breve{c}_{Q_{n-1}^+} = \breve{c}_Q + \frac{3}{4} - \frac{1}{2}\left(3 - \frac{1}{2\mu^*}\right) = \breve{c}_Q - \frac{\gamma}{4}\left(3 - \frac{1}{\mu^*}\right).$$

This means that $\breve{c}_Q - \breve{c}_{Q_i^-} = \breve{c}_Q - \breve{c}_{Q_i^+} = \gamma/2$, $\breve{c}_Q - \breve{c}_{Q_{n-1}^-} = 3\gamma/4$, and $\breve{c}_Q - \breve{c}_{Q_{n-1}^+} = \frac{\gamma}{4}\left(3 - \frac{1}{\mu^*}\right)$. Therefore, $\mu\mathtt{score}_1(Q,i) + (1-\mu)\mathtt{score}_2(Q,i) = \gamma/2$ and $\mu\mathtt{score}_1(Q,n-1) + (1-\mu)\mathtt{score}_2(Q,n-1) = \frac{\mu\gamma}{4}\left(3 - \frac{1}{\mu^*}\right) + \frac{3\gamma(1-\mu)}{4} = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$. This means that $\mu\mathtt{score}_1(Q,i) + (1-\mu)\mathtt{score}_2(Q,i) = \gamma/2 < \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} = \mu\mathtt{score}_1(Q,n-1) + (1-\mu)\mathtt{score}_2(Q,n-1)$ so long as $\mu < \mu^*$.

The next node branch-and-bound will explore is Q_{n-1}^-. The vector $\breve{x}_{Q_{n-1}^-}$ has fractional values only in positions i and $n-2$. Branching on i, we again have that $\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,i}^{-,-}} = \breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,i}^{-,+}} = \gamma/2$. Branching on $n-2$, $Q_{n-1,n-2}^{-,-}$ is infeasible, so $\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,n-2}^{-,-}}$ equals some large number $B \geq ||c||_1$. Next, $\breve{x}_{Q_{n-1,n-2}^{-,+}} = \breve{x}_{Q_{n-1}^-} + \frac{1}{2}e_{n-2} - \frac{1}{2}e_n$, so $\breve{c}_{Q_{n-1,n-2}^{-,+}} = \breve{c}_{Q_{n-1}^-} - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$. Therefore, $\mu\mathtt{score}_1\left(Q_{n-1}^-, i\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1}^-, i\right) = \gamma/2$ and

$$\mu\mathtt{score}_1\left(Q_{n-1}^-, n-2\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1}^-, n-2\right) = \frac{\mu\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right) + (1-\mu)B$$

$$= B + \mu\left(\frac{3\gamma}{2} - \frac{\gamma}{4\mu^*} - B\right)$$

$$\geq B + \mu^*\left(\frac{3\gamma}{2} - \frac{\gamma}{4\mu^*} - B\right)$$

$$= B - \frac{\gamma}{4} + \mu^*\left(\frac{3\gamma}{2} - B\right)$$

$$> B - \frac{\gamma}{4} + \frac{3\gamma/4 - B}{3\gamma/2 - B}\left(\frac{3\gamma}{2} - B\right)$$

$$= B - \frac{\gamma}{4} + \frac{3\gamma}{4} - B$$

$$= \frac{\gamma}{2},$$

where the final inequality holds because $\mu^* < 1 < \frac{B - 3\gamma/4}{B - 3\gamma/2}$. Therefore, x_{n-2} will be branched on next.

Since $Q_{n-1,n-2}^{-,-}$ is infeasible, the next node branch-and-bound will explore is $Q_{n-1,n-2}^{-,+}$. The vector $\breve{x}_{Q_{n-1,n-2}^{-,+}}$ has fractional values only in positions i and n. Both MILP instances $Q_{n-1,n-2,n}^{-,+,-}$ and $Q_{n-1,n-2,n}^{-,+,+}$ are infeasible, so $\mu\mathtt{score}_1\left(Q_{n-1,n-2}^{-,+}, n\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1,n-2}^{-,+}, n\right) = B$ whereas

30

$\mu \mathrm{score}_1\left(Q_{n-1,n-2}^{-,+}, i\right) + (1-\mu)\mathrm{score}_2\left(Q_{n-1,n-2}^{-,+}, i\right) = \gamma/2$, as before. Therefore, branch-and-bound will branch on x_n and fathom both children.

The next node branch-and-bound will explore is Q_{n-1}^+. The vector $\breve{\boldsymbol{x}}_{Q_{n-1}^+}$ has fractional values only in positions i and n. Branching on i, we again have that $\breve{c}_{Q_{n-1}^+} - \breve{c}_{Q_{n-1,i}^{+,-}} = \breve{c}_{Q_{n-1}^+} - \breve{c}_{Q_{n-1,i}^{+,+}} = \gamma/2$. Branching on x_n, $\breve{\boldsymbol{x}}_{Q_{n-1,n}^{+,-}} = \breve{\boldsymbol{x}}_{Q_{n-1}^+} - \frac{1}{2}\boldsymbol{e}_n + \frac{1}{2}\boldsymbol{e}_{n-2}$, so $\breve{c}_{Q_{n-1,n}^{+,-}} = \breve{c}_{Q_{n-1}^+} - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$. Meanwhile, $Q_{n-1,n}^{+,+}$ is infeasible, so $\breve{c}_{Q_{n-1}^+} - \breve{c}_{Q_{n-1,n-2}^{+,+}} = B$. Therefore, $\mu \mathrm{score}_1\left(Q_{n-1}^+, i\right) + (1-\mu)\mathrm{score}_2\left(Q_{n-1}^+, i\right) = \gamma/2$ and $\mu \mathrm{score}_1\left(Q_{n-1}^+, n\right) + (1-\mu)\mathrm{score}_2\left(Q_{n-1}^+, n\right) = \frac{\mu\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right) + (1-\mu)B > \gamma/2$. Therefore, x_n will be branched on next.

The next node branch-and-bound will explore is $Q_{n-1,n}^{+,-}$. The vector $\breve{\boldsymbol{x}}_{Q_{n-1,n}^{+,-}}$ has fractional values only in positions i and $n-2$. Both MILP instances $Q_{n-1,n,n-2}^{+,-,-}$ and $Q_{n-1,n,n-2}^{+,-,-}$ are infeasible, so $\mu \mathrm{score}_1\left(Q_{n-1,n}^{-,+}, n-2\right) + (1-\mu)\mathrm{score}_2\left(Q_{n-1,n}^{-,+}, n-2\right) = B$ whereas $\mu \mathrm{score}_1\left(Q_{n-1,n-2}^{-,+}, i\right) + (1-\mu)\mathrm{score}_2\left(Q_{n-1,n-2}^{-,+}, i\right) = \gamma/2$, as before. Therefore, branch-and-bound will branch on x_{n-2} and fathom both children.

At this point, all children have been fathomed, so branch-and-bound will terminate. $\qquad\square$

Lemma C.2. *Suppose $\mu > \mu^*$. Then for any MILP $Q_{\gamma,n} \in \mathcal{F}_{n,\mu^*}$, branch-and-bound with the scoring rule $\mu \mathrm{score}_1 + (1-\mu)\mathrm{score}_2$ will create a tree of depth at least $2^{(n-5)/4}$.*

Proof of Lemma C.2. To prove this lemma, we use induction to show that on any path from the root of the branch-and-bound tree to a node of depth $i = \lfloor(n-3)/2\rfloor$, if J are the set of indices branched on along that path, then $J \subseteq \{x_1, \ldots, x_{n-3}\}$. Even after branching on i nodes from $\{x_1, \ldots, x_{n-3}\}$, the MILP will still be feasible, so the branch will not yet have been fathomed (since the original MILP is infeasible, a node will be fathomed only when it is infeasible). Therefore, branch-and-bound will continue down every branch to depth $\lfloor(n-3)/2\rfloor$, thus creating a tree with $2^{(n-4)/2}$ nodes.

Claim C.3. *On any path from the root of the branch-and-bound tree to a node of depth $i = \lfloor(n-3)/2\rfloor$, if J are the set of indices branched on along that path, then $J \subseteq \{x_1, \ldots, x_{n-3}\}$.*

Proof of Claim C.3. We prove this claim by induction.

Inductive hypothesis. For $j \leq \lfloor(n-3)/2\rfloor$, let J be the set of indices branched on along an arbitrary path of the branch-and-bound tree from the root to a node of depth j. Then $J \subseteq \{x_1, \ldots, x_{n-3}\}$.

Base case ($j = 0$). As we saw in the proof of Lemma C.1, if $\mu > \mu^*$, then branch-and-bound will first branch on x_i where $i = \lceil(n-3)/2\rceil$.

Inductive step. Let j be an arbitrary index such that $0 \leq j \leq i-1$. Let J be the set of indices branched on along an arbitrary path of the branch-and-bound tree from the root to a node of depth j. We know from the inductive hypothesis that $J \subseteq \{x_1, \ldots, x_{n-3}\}$. Let Q' be the MILP at that node. Since $j \leq \lfloor(n-3)/2\rfloor - 1$, we know that the LP relaxation of Q' is feasible. Let z be the number of variables set to zero in J and let $x_{p_1}, x_{p_2}, \ldots, x_{p_t}$ be $\{x_1, \ldots, x_{n-3}\} \setminus J$ ordered such that $p_k < p_{k'}$ for $k < k'$. We know that the solution to the LP relaxation of Q' will have the first $i' := \lfloor(n-3)/2\rfloor - z$ variables $x_{p_1}, \ldots, x_{p_{i'}}$ set to 0, it will set $x_{p_{i'+1}}$ to 1/2, and it will set the

31

remaining variables in $\{x_1, \ldots, x_{n-3}\} \setminus J$ to 1. Thus, the fractional variables are $x_{p_{i'}+1}$ and x_{n-1}. Note that since $z \leq |J| \leq \lfloor (n-3)/2 \rfloor - 1$, $i' = \lfloor (n-3)/2 \rfloor - z \geq 1$.

Suppose we branch on $x_{p_{i'}+1}$. If we set $x_{p_{i'}+1} = 0$, then the LP relaxation of $(Q')^-_{p_{i'}+1}$ will set $x_{p_{i'}}$ to be $1/2$ and otherwise the optimal solution will remain unchanged. Thus, $\check{c}_{Q'} - \check{c}_{(Q')^-_{p_{i'}+1}} = \check{c}_{Q'} - \left(\check{c}_{Q'} - \gamma p_{i'+1}/2 + \gamma p_{i'}/2\right) = \frac{\gamma(p_{i'+1}-p_{i'})}{2}$. Meanwhile, if we set $x_{p_{i'}+1} = 1$, then the LP relaxation of $(Q')^-_{p_{i'}+1}$ will set $x_{p_{i'}+2}$ to be 0 and otherwise the optimal solution will remain unchanged. Thus, $\check{c}_{Q'} - \check{c}_{(Q')^+_{p_{i'}+1}} = \check{c}_{Q'} - \left(\check{c}_{Q'} + \gamma p_{i'+1}/2 - \gamma p_{i'+2}/2\right) = \frac{\gamma(p_{i'+2}-p_{i'+1})}{2}$. Suppose that $\check{c}_{Q'} - \check{c}_{(Q')^+_{p_{i'}+1}} > \check{c}_{Q'} - \check{c}_{(Q')^-_{p_{i'}+1}}$. Then

$$\mu\text{score}_1\left(Q', p_{i'+1}\right) + (1-\mu)\text{score}_2\left(Q', p_{i'+1}\right) = \frac{\gamma}{2}\left(\mu\left(p_{i'+1} - p_{i'}\right) + (1-\mu)\left(p_{i'+2} - p_{i'+1}\right)\right)$$
$$\geq \frac{\gamma}{2}\left(\mu + (1-\mu)\right)$$
$$= \frac{\gamma}{2}.$$

Meanwhile, suppose that $\check{c}_{Q'} - \check{c}_{(Q')^+_{p_{i'}+1}} \leq \check{c}_{Q'} - \check{c}_{(Q')^-_{p_{i'}+1}}$. Then

$$\mu\text{score}_1\left(Q', p_{i'+1}\right) + (1-\mu)\text{score}_2\left(Q', p_{i'+1}\right) = \frac{\gamma}{2}\left(\mu\left(p_{i'+2} - p_{i'+1}\right) + (1-\mu)\left(p_{i'+1} - p_{i'}\right)\right)$$
$$\geq \frac{\gamma}{2}\left(\mu + (1-\mu)\right)$$
$$= \frac{\gamma}{2}.$$

Meanwhile, as in the proof of Lemma C.1, $\mu\text{score}_1(Q', n-1) + (1-\mu)\text{score}_2(Q', n-1) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < \frac{\gamma}{2}$ so long as $\mu > \mu^*$. Thus, branch-and-bound will branch next on $x_{p_{i'}}$. $\square$

$\square$

$\square$

Theorem C.4. *Let*

$$\text{score}_1(Q, i) = \min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\} \text{ and } \text{score}_2(Q, i) = \max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}.$$

For all even $n \geq 6$ and all $\mu^* \in \left(\frac{1}{3}, \frac{2}{3}\right)$, there exists an infinite family $\mathcal{G}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{G}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\text{score}_1 + (1-\mu)\text{score}_2$ results in a branch-and-bound tree with $\Omega\left(2^{(n-5)/4}\right)$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results a tree with $O(1)$ nodes.

Proof. For any constant $\gamma \geq 1$, let $c_1 \in \mathbb{R}^{n-3}$ be a vector such that

$$c_1[i] = \begin{cases} 0 & \text{if } i < (n-3)/2 \\ 1.5 & \text{if } i = \lceil (n-3)/2 \rceil \\ 3 - \frac{1}{2\mu^*} & \text{if } i > (n-3)/2 + 1 \end{cases}$$

and let $c_2 = (1, 2, 3)$. Let $c = \gamma(c_1, c_2) \in \mathbb{R}^n$ be the concatenation of c_1 and c_2 multiplied with γ. For example, if $\gamma = 1$ and $n = 8$, then $c = \left(0, 0, 1.5, 3 - \frac{1}{2\mu^*}, 3 - \frac{1}{2\mu^*}, 1, 2, 3\right)$ Next, define the

n-dimensional vectors $\boldsymbol{a}_1 = 2\sum_{i=1}^{n-3} \boldsymbol{e}_i$ and $\boldsymbol{a}_2 = 2\sum_{i=1}^{3} \boldsymbol{e}_{n-3+i}$, and let A be a matrix whose first row is $\boldsymbol{a}_1$ and second row is $\boldsymbol{a}_2$. Let $Q_{\gamma,n}$ be the MILP

$$
\begin{aligned}
\text{maximize} \quad & \boldsymbol{c} \cdot \boldsymbol{x} \\
\text{subject to} \quad & A\boldsymbol{x} = (n-3, 3)^\top \\
& \boldsymbol{x} \in \{0,1\}^n.
\end{aligned}
$$

We define $\mathcal{G}_{n,\mu^*} = \{Q_{n,\gamma} : \gamma \geq 1\}$.

For every even $n \geq 6$, $Q_{\gamma,n}$, both of the constraints $\boldsymbol{a}_1 \cdot \boldsymbol{x} = 2\sum_{i=1}^{n-3} x_i = n-3$ and $\boldsymbol{a}_2 \cdot \boldsymbol{x} = 2\sum_{i=1}^{3} x_{n-3+i} = 3$ are infeasible for $\boldsymbol{x} \in \{0,1\}^n$ since $2\sum_{i=1}^{n-3} x_i$ and $2\sum_{i=1}^{3} x_{n-3+i}$ are even numbers but 3 and $n-3$ are odd numbers. The key idea of this proof is that if branch-and-bound branches on all variables in $\{x_{n-2}, x_{n-1}, x_n\}$ first, it will terminate upon making a tree of size at most $2^3 = 8$, since at most three branches are necessary to determine that $\boldsymbol{a}_1 \cdot \boldsymbol{x} = 2\sum_{i=1}^{3} x_{n-3+i} = 3$ is infeasible. However, if branch-and-bound branches on all variables in $\{x_1, \ldots, x_{n-3}\}$ first, it will create a tree with exponential size before it terminates.

Lemma C.5. *Suppose $\mu > \mu^*$. Then for any MILP $Q_{\gamma,n} \in \mathcal{G}_{n,\mu^*}$, $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ branches on all variables in $\{x_{n-2}, x_{n-1}, x_n\}$ before branching on variables in $\{x_1, \ldots, x_{n-3}\}$.*

Proof of Lemma C.5. For ease of notation, for the remainder of this proof, we drop the subscript (γ, n) from $Q_{\gamma,n}$ and denote this MILP as Q. The optimal solution to the LP relaxation of Q has the following form:

$$
\breve{\boldsymbol{x}}_Q[j] = \begin{cases} 0 & \text{if } j \leq \lfloor (n-3)/2 \rfloor \text{ or } j = n-2 \\ \frac{1}{2} & \text{if } j = \lceil (n-3)/2 \rceil \text{ or } j = n-1 \\ 1 & \text{if } \lceil (n-3)/2 \rceil \leq j \leq n-3 \text{ or } j = n. \end{cases}
$$

For example, if $n = 8$, then $\breve{\boldsymbol{x}}_Q = \left(0, 0, \frac{1}{2}, 1, 1, 0, \frac{1}{2}, 1\right)$. Therefore, the only candidate variables to branch on are x_{n-1} and x_i where $i = \lceil (n-3)/2 \rceil$. Branching on x_i, we have $\breve{\boldsymbol{x}}_{Q_i^-} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_i + \frac{1}{2}\boldsymbol{e}_{i-1}$ and $\breve{\boldsymbol{x}}_{Q_i^+} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{i+1}$. Branching on x_{n-1}, we have $\breve{\boldsymbol{x}}_{Q_{n-1}^-} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\breve{\boldsymbol{x}}_{Q_{n-1}^+} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$
\begin{aligned}
\breve{c}_{Q_i^-} &= \breve{c}_Q - \frac{3\gamma}{4}, \\
\breve{c}_{Q_i^+} &= \breve{c}_Q + \frac{3\gamma}{4} - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right) = \breve{c}_Q - \frac{3\gamma}{4}\left(1 - \frac{1}{\mu^*}\right), \\
\breve{c}_{Q_{n-1}^-} &= \breve{c}_Q - \gamma + \frac{\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}, \text{ and} \\
\breve{c}_{Q_{n-1}^+} &= \breve{c}_Q + \gamma - \frac{3\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}.
\end{aligned}
$$

This means that $\breve{c}_Q - \breve{c}_{Q_i^-} = 3\gamma/4$, $\breve{c}_Q - \breve{c}_{Q_i^+} = \frac{\gamma}{4}\left(3 - \frac{1}{\mu^*}\right)$, and $\breve{c}_Q - \breve{c}_{Q_{n-1}^-} = \breve{c}_Q - \breve{c}_{Q_{n-1}^+} = \gamma/2$. Therefore, $\mu\texttt{score}_1(Q, i) + (1-\mu)\texttt{score}_2(Q, i) = \frac{\mu\gamma}{4}\left(3 - \frac{1}{\mu^*}\right) + \frac{3\gamma(1-\mu)}{4} = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$ and $\mu\texttt{score}_1(Q, n-1) + (1-\mu)\texttt{score}_2(Q, n-1) = \gamma/2$. This means that $\mu\texttt{score}_1(Q, i) + (1-\mu)\texttt{score}_2(Q, i) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < \gamma/2 = \mu\texttt{score}_1(Q, n-1) + (1-\mu)\texttt{score}_2(Q, n-1)$ so long as $\mu > \mu^*$.

The next node branch-and-bound will explore is Q_{n-1}^-. The vector $\breve{\boldsymbol{x}}_{Q_{n-1}^-}$ has fractional values only in positions i and $n-2$. Branching on i, we again have that $\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,i}^{-,-}} = 3\gamma/4$ and

$\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,i}^{-,+}} = \frac{\gamma}{4}\left(3 - \frac{1}{\mu^*}\right)$. Branching on $n-2$, $Q_{n-1,n-2}^{-,-}$ is infeasible, so $\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,n-2}^{-,-}}$ equals some large number $B \geq \|\mathbf{c}\|_1$. Next, $\breve{\mathbf{x}}_{Q_{n-1,n-2}^{-,+}} = \breve{\mathbf{x}}_{Q_{n-1}^-} + \frac{1}{2}\mathbf{e}_{n-2} - \frac{1}{2}\mathbf{e}_n$, so $\breve{c}_{Q_{n-1,n-2}^{-,+}} = \breve{c}_{Q_{n-1}^-} - \gamma$. Therefore, $\mu\mathtt{score}_1\left(Q_{n-1}^-, i\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1}^-, i\right) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$ and

$$
\begin{aligned}
\mu\mathtt{score}_1\left(Q_{n-1}^-, n-2\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1}^-, n-2\right) &= \mu\gamma + (1-\mu)B \\
&= B + \mu\left(\gamma - B\right) \\
&= B - \frac{\mu\gamma}{4\mu^*} + \mu\left(\gamma + \frac{\gamma}{4\mu^*} - B\right) \\
&> B - \frac{\mu\gamma}{4\mu^*} + \frac{3\gamma/4 - B}{\gamma + \frac{\gamma}{4\mu^*} - B}\left(\gamma + \frac{\gamma}{4\mu^*} - B\right) \\
&= B - \frac{\mu\gamma}{4\mu^*} + 3\gamma/4 - B \\
&= \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*},
\end{aligned}
$$

where the final inequality holds because $\mu < 1 < \frac{B - 3\gamma/4}{B - (\gamma + \gamma/(4\mu^*))}$. Therefore, x_{n-2} will be branched on next.

Since $Q_{n-1,n-2}^{-,-}$ is infeasible, the next node branch-and-bound will explore is $Q_{n-1,n-2}^{-,+}$. The vector $\breve{\mathbf{x}}_{Q_{n-1,n-2}^{-,+}}$ has fractional values only in positions i and n. Since both MILP instances $Q_{n-1,n-2,n}^{-,+,-}$ and $Q_{n-1,n-2,n}^{-,+,+}$ are infeasible, so $\mu\mathtt{score}_1\left(Q_{n-1,n-2}^{-,+}, n\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1,n-2}^{-,+}, n\right) = B$ whereas $\mu\mathtt{score}_1\left(Q_{n-1,n-2}^{-,+}, i\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1,n-2}^{-,+}, i\right) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < B$. Therefore, branch-and-bound will branch on x_n and fathom both children.

The next node branch-and-bound will explore is Q_{n-1}^+. The vector $\breve{\mathbf{x}}_{Q_{n-1}^+}$ has fractional values only in positions i and n. Branching on i, we again have that $\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,i}^{-,-}} = 3\gamma/4$ and $\breve{c}_{Q_{n-1}^-} - \breve{c}_{Q_{n-1,i}^{-,+}} = \frac{\gamma}{4}\left(3 - \frac{1}{\mu^*}\right)$. Branching on x_n, $\breve{\mathbf{x}}_{Q_{n-1,n}^{+,-}} = \breve{\mathbf{x}}_{Q_{n-1}^+} - \frac{1}{2}\mathbf{e}_n + \frac{1}{2}\mathbf{e}_{n-2}$, so $\breve{c}_{Q_{n-1,n}^{+,-}} = \breve{c}_{Q_{n-1}^+} - \gamma$. Meanwhile, $Q_{n-1,n}^{+,+}$ is infeasible, so $\breve{c}_{Q_{n-1}^+} - \breve{c}_{Q_{n-1,n-2}^{+,+}}$ equals some large number $B \geq \|\mathbf{c}\|_1$. Therefore, $\mu\mathtt{score}_1\left(Q_{n-1}^+, i\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1}^+, i\right) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$ and $\mu\mathtt{score}_1\left(Q_{n-1}^+, n\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1}^+, n\right) = \mu\gamma + (1-\mu)B > \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$. Therefore, x_n will be branched on next.

The next node branch-and-bound will explore is $Q_{n-1,n}^{+,-}$. The vector $\breve{\mathbf{x}}_{Q_{n-1,n}^{+,-}}$ has fractional values only in positions i and $n-2$. Since both MILP instances $Q_{n-1,n,n-2}^{+,-,-}$ and $Q_{n-1,n,n-2}^{+,-,-}$ are infeasible, so $\mu\mathtt{score}_1\left(Q_{n-1,n}^{-,+}, n-2\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1,n}^{-,+}, n-2\right) = B$ whereas $\mu\mathtt{score}_1\left(Q_{n-1,n-2}^{-,+}, i\right) + (1-\mu)\mathtt{score}_2\left(Q_{n-1,n-2}^{-,+}, i\right) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < B$, as before. Therefore, branch-and-bound will branch on x_{n-2} and fathom both children.

At this point, all children have been fathomed, so branch-and-bound will terminate. $\qquad\square$

Lemma C.6. *Suppose $\mu < \mu^*$. Then for any MILP $Q_{\gamma,n} \in \mathcal{G}_{n,\mu^*}$, branch-and-bound with the scoring rule $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ will create a tree of depth at least $2^{(n-5)/4}$.*

Proof. Let $i = \lceil(n-3)/2\rceil$. We first prove two useful claims.

Claim C.7. *Let j be an even number such that $2 \leq j \leq i - 2$ and let $J = \left\{x_{i-j/2}, \ldots, x_{i+j/2-1}\right\}$. Suppose that B&B has branched on exactly the variables in J and suppose that the number of variables set to 1 equals the number of variables set to 0. Then B&B will next branch on the*

variable $x_{i+j/2}$. Similarly, suppose $J = \{x_{i-j/2+1}, x_{i-j/2+2}, \ldots, x_{i+j/2}\}$. Suppose that B&B has branched on exactly the variables in J and suppose that the number of variables set to 1 equals the number of variables set to 0. Then B&B will next branch on the variable $x_{i-j/2}$.

Proof. Let Q be the MILP contained in the node at the end of the path. This proof has two cases.

Case 1: $J = \{x_{i-j/2}, x_{i-j/2+1}, \ldots, x_{i+j/2-1}\}$. In this case, there is a set $J_< = \{x_1, \ldots, x_{i-j/2-1}\}$ of $i - \frac{j}{2} - 1$ variables smaller than x_i that have not yet been branched on and there is a set $J_> = \{x_{i+j/2}, \ldots, x_{n-3}\}$ of $n - 3 - \left(i + \frac{j}{2} - 1\right) = 2i - 1 - \left(i + \frac{j}{2} - 1\right) = i - \frac{j}{2}$ variables in $\{x_{i+1}, \ldots, x_{n-3}\}$ that have not yet been branched on. Since the number of variables set to 1 in J equals the number of variables set to 0, the LP relaxation will set the $i - \frac{j}{2} - 1$ variables in $J_<$ to 0, the $i - \frac{j}{2} - 1$ variables in $J_> \setminus \{x_{i+j/2}\}$ to 1, and $x_{i+j/2}$ to $\frac{1}{2}$. It will also set $x_{n-2} = 0$, $x_{n-1} = \frac{1}{2}$, and $x_n = 1$. Therefore, the two fractional variables are $x_{i+j/2}$ and x_{n-1}. Branching on $x_{i+j/2}$, we have $\breve{\boldsymbol{x}}_{Q^-_{i+j/2}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i+j/2} + \frac{1}{2}\boldsymbol{e}_{i-j/2-1}$ and $\breve{\boldsymbol{x}}_{Q^+_{i+j/2}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i+j/2} - \frac{1}{2}\boldsymbol{e}_{i+j/2+1}$. Branching on x_{n-1}, we have that $\breve{\boldsymbol{x}}_{Q^-_{n-1}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\breve{\boldsymbol{x}}_{Q^+_{n-1}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\breve{c}_{Q^-_{i+j/2}} = \breve{c}_Q - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$$

$$\breve{c}_{Q^+_{i+j/2}} = \breve{c}_Q$$

$$\breve{c}_{Q^-_{n-1}} = \breve{c}_Q - 1 + \frac{1}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

$$\breve{c}_{Q^+_{n-1}} = \breve{c}_Q + 1 - \frac{3}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

This means that $\breve{c}_Q - \breve{c}_{Q^-_{i+j/2}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$, $\breve{c}_Q - \breve{c}_{Q^+_{i+j/2}} = 0$, and $\breve{c}_Q - \breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \breve{c}_{Q^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\mu\texttt{score}_1(Q, i+j/2) + (1-\mu)\texttt{score}_2(Q, i+j/2) = \frac{\gamma(1-\mu)}{2}\left(3 - \frac{1}{2\mu^*}\right)$ and $\mu\texttt{score}_1(Q, n-1) + (1-\mu)\texttt{score}_2(Q, n-1) = \frac{\gamma}{2}$. Since $\mu < \mu^*$ and $\mu^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, we have that

$$\mu\texttt{score}_1(Q, i+j/2) + (1-\mu)\texttt{score}_2(Q, i+j/2) = \frac{\gamma(1-\mu)}{2}\left(3 - \frac{1}{2\mu^*}\right)$$
$$\geq \frac{\gamma(1-\mu^*)}{2}\left(3 - \frac{1}{2\mu^*}\right)$$
$$\geq \frac{\gamma}{2}.$$

Therefore, $x_{i+j/2}$ will be branched on next.

Case 2: $J = \{x_{i-j/2+1}, x_{i-j/2+2}, \ldots, x_{i+j/2}\}$. In this case, there is a set $J_< = \{x_1, \ldots, x_{i-j/2}\}$ of $i - \frac{j}{2}$ variables smaller than x_i that have not yet been branched on and a set $J_> = \{x_{i+j/2+1}, \ldots, x_{n-3}\}$ of $n - 3 - \left(i + \frac{j}{2}\right) = 2i - 1 - \left(i + \frac{j}{2}\right) = i - \frac{j}{2} - 1$ variables in $\{x_{i+1}, \ldots, x_{n-3}\}$ that have not yet been branched on. Since the number of variables set to 1 in J equals the number of variables set to 0, the LP relaxation will set the $i - \frac{j}{2} - 1$ variables in $J_< \setminus \{x_{i-j/2}\}$ to 0, the $i - \frac{j}{2} - 1$ variables in $J_> \setminus \{x_{i+j/2}\}$ to 1, and $x_{i-j/2}$ to $\frac{1}{2}$. It will also set $x_{n-2} = 0$, $x_{n-1} = \frac{1}{2}$, and $x_n = 1$. Therefore, the two fractional variables are $x_{i-j/2}$ and x_{n-1}. Branching on $x_{i-j/2}$, we have $\breve{\boldsymbol{x}}_{Q^-_{i-j/2}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i-j/2} + \frac{1}{2}\boldsymbol{e}_{i-j/2-1}$ and

35

$\breve{\boldsymbol{x}}_{Q^+_{i-j/2}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i-j/2} - \frac{1}{2}\boldsymbol{e}_{i+j/2+1}$. Branching on x_{n-1}, we have that $\breve{\boldsymbol{x}}_{Q^-_{n-1}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\breve{\boldsymbol{x}}_{Q^+_{n-1}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\breve{c}_{Q^-_{i-j/2}} = \breve{c}_Q$$

$$\breve{c}_{Q^+_{i-j/2}} = \breve{c}_Q - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$$

$$\breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \gamma + \frac{\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

$$\breve{c}_{Q^+_{n-1}} = \breve{c}_Q + \gamma - \frac{3\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

This means that $\breve{c}_Q - \breve{c}_{Q^-_{i-j/2}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$, $\breve{c}_Q - \breve{c}_{Q^+_{i-j/2}} = 0$, and $\breve{c}_Q - \breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \breve{c}_{Q^+_{n-1}} = \frac{\gamma}{2}$ as in the previous case, so $x_{i-j/2}$ will be branched on next. $\qquad\square$

Claim C.8. *Suppose that J is the set of variables branched on along a path of depth $2 \leq j \leq i-2$ where j is odd and let $j' = \lfloor j/2 \rfloor$. Suppose that $J = \{x_{i-j'}, x_{i-j'+1}, \ldots, x_{i+j'}\}$. Moreover, suppose that the number of variables set to 1 in J equals the number of variables set to 0, plus or minus 1. Then B&B will either branch on $x_{i-j'-1}$ or $x_{i+j'+1}$.*

Proof. Let Q be the MILP contained a the end of the path. There are $i - j' - 1$ variables $J_< = \{x_1, \ldots, x_{i-j'-1}\}$ that are smaller than x_i that have not yet been branched on, and $n - 3 - (i+j') = 2i - 1 - (i+j') = i - j' - 1$ variables $J_> = \{x_{i+j'+1}, \ldots, x_{n-3}\}$ in $\{x_{i+1}, \ldots, x_{n-3}\}$ that have not yet been branched on. Let z be the number of variables in J set to 0 and let o be the number of variables set to 1. This proof has two cases:

Case 1: $z = o + 1$. Since $z + o = j$, we know that $z = j' + 1$ and $o = j'$. Therefore, the LP relaxation will set the variables in $J_< \setminus \{x_{i-j'-1}\}$ to zero for a total of $|J_< \setminus \{x_{i-j'-1}\}| + z = i - j' - 2 + j' + 1 = i - 1$ zeros, it will set the variables in $J_>$ to one for a total of $|J_>| + o = i - j' - 1 + j' = i - 1$ ones, and it will set $x_{i-j'-1}$ to $\frac{1}{2}$. It will also set $x_{n-2} = 0$, $x_{n-1} = \frac{1}{2}$, and $x_n = 1$. Therefore, the two fractional variables are $x_{i-j'-1}$ and x_{n-1}. Branching on $x_{i-j'-1}$, we have $\breve{\boldsymbol{x}}_{Q^-_{i-j'-1}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i-j'-1} + \frac{1}{2}\boldsymbol{e}_{i-j'-2}$ and $\breve{\boldsymbol{x}}_{Q^+_{i-j'-1}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i-j'-1} - \frac{1}{2}\boldsymbol{e}_{i+j'+1}$. Branching on x_{n-1}, we have that $\breve{\boldsymbol{x}}_{Q^-_{n-1}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\breve{\boldsymbol{x}}_{Q^+_{n-1}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\breve{c}_{Q^-_{i-j'-1}} = \breve{c}_Q$$

$$\breve{c}_{Q^+_{i-j'-1}} = \breve{\boldsymbol{x}}_Q - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$$

$$\breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \gamma + \frac{\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

$$\breve{c}_{Q^+_{n-1}} = \breve{c}_Q + \gamma - \frac{3\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

This means that $\breve{c}_Q - \breve{c}_{Q^-_{i-j'-1}} = 0$, $\breve{c}_Q - \breve{c}_{Q^+_{i-j'-1}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)$, and $\breve{c}_Q - \breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \breve{c}_{Q^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\mu\mathsf{score}_1(Q, i-j'-1) + (1-\mu)\mathsf{score}_2(Q, i-j'-1) = \frac{\gamma(1-\mu)}{2}\left(3 - \frac{1}{2\mu^*}\right)$ and $\mu\mathsf{score}_1(Q, n-1) + (1-\mu)\mathsf{score}_2(Q, n-1) = \frac{\gamma}{2}$. Since $\mu < \mu^*$ and $\mu^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, we have that

$$\mu\mathsf{score}_1(Q, i-j'-1) + (1-\mu)\mathsf{score}_2(Q, i-j'-1) = \frac{\gamma(1-\mu)}{2}\left(3 - \frac{1}{2\mu^*}\right)$$

36

$$\geq \frac{\gamma\left(1-\mu^*\right)}{2}\left(3-\frac{1}{2\mu^*}\right)$$

$$\geq \frac{\gamma}{2}.$$

Therefore, $x_{i-j'-1}$ will be branched on next.

Case 1: $z = o - 1$. Since $z + o = j$, we know that $z = j'$ and $o = j' + 1$. Therefore, the LP relaxation will set the variables in $J_<$ to zero for a total of $|J_<|+z = i-j'-1+j' = i-1$ zeros, it will set the variables in $J_> \setminus \{x_{i+j'+1}\}$ to one for a total of $|J_> \setminus \{x_{i+j'+1}\}|+o = i-j'-2+j'+1 = i-1$ ones, and it will set $x_{i+j'+1}$ to $\frac{1}{2}$. It will also set $x_{n-2} = 0$, $x_{n-1} = \frac{1}{2}$, and $x_n = 1$. Therefore, the two fractional variables are $x_{i+j'+1}$ and x_{n-1}. Branching on $x_{i+j'+1}$, we have $\breve{\boldsymbol{x}}_{Q^-_{i+j'+1}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i+j'+1} + \frac{1}{2}\boldsymbol{e}_{i-j'-1}$ and $\breve{\boldsymbol{x}}_{Q^+_{i+j'+1}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i+j'+1} - \frac{1}{2}\boldsymbol{e}_{i+j'+2}$. Branching on x_{n-1}, we have that $\breve{\boldsymbol{x}}_{Q^-_{n-1}} = \breve{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\breve{\boldsymbol{x}}_{Q^+_{n-1}} = \breve{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\breve{c}_{Q^-_{i+j'+1}} = \breve{c}_Q - \frac{\gamma}{2}\left(3-\frac{1}{2\mu^*}\right)$$

$$\breve{c}_{Q^+_{i+j'+1}} = \breve{\boldsymbol{x}}_Q$$

$$\breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \gamma + \frac{\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

$$\breve{c}_{Q^+_{n-1}} = \breve{c}_Q + \gamma - \frac{3\gamma}{2} = \breve{c}_Q - \frac{\gamma}{2}$$

This means that $\breve{c}_Q - \breve{c}_{Q^-_{i+j'+1}} = \frac{\gamma}{2}\left(3-\frac{1}{2\mu^*}\right)$, $\breve{c}_Q - \breve{c}_{Q^+_{i+j'+1}} = 0$, and $\breve{c}_Q - \breve{c}_{Q^-_{n-1}} = \breve{c}_Q - \breve{c}_{Q^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\mu\mathtt{score}_1(Q, i+j'+1) + (1-\mu)\mathtt{score}_2(Q, i+j'+1) = \frac{\gamma(1-\mu)}{2}\left(3-\frac{1}{2\mu^*}\right)$ and $\mu\mathtt{score}_1(Q, n-1) + (1-\mu)\mathtt{score}_2(Q, n-1) = \frac{\gamma}{2}$. As in the previous case, this means that $x_{i+j'+1}$ will be branched on next. $\square$

We now prove by induction that there are $2^{(\lceil (n-3)/2 \rceil)-1)/2} \geq 2^{(n-5)/4}$ paths in the B&B tree of length at least $i - 2$. Therefore, the size of the tree is at least $2^{(n-5)/4}$.

Inductive hypothesis. Let j be an arbitrary integer between 1 and $i - 2$. If j is even, then there exist at least $2^{j/2}$ paths in the B&B tree from the root to nodes of depth j such that if J is the set indices branched on along a given path, then $J = \{x_{i-j/2}, x_{i-j/2+1}, \ldots, x_{i+j/2-1}\}$ or $J = \{x_{i-j/2+1}, x_{i-j/2+2}, \ldots, x_{i+j/2}\}$. Moreover, the number of variables set to 0 in J equals the number of variables set to 1. Meanwhile, if j is odd, let $j' = \lfloor j/2 \rfloor$. There exist at least $2^{(j+1)/2}$ paths in the B&B tree from the root to nodes of depth j such that if J is the set indices branched on along a given path, then $J = \{x_{i-j'}, x_{i-j'+1}, \ldots, x_{i+j'}\}$. Moreover, the number of variables set to 0 in J equals the number of variables set to 1, plus or minus 1.

Base case. To prove the base case, we need to show that B&B first branches on x_i. We saw that this will be the case in Lemma C.5 so long as $\mu < \mu^*$.

Inductive step. Let j be an arbitrary integer between 1 and $i - 3$. There are two cases, one where j is even and one where j is odd. First, suppose j is even. From the inductive hypothesis, we know that there exist at least $2^{j/2}$ paths in the B&B tree from the root to nodes of depth j such that

if J is the set varibles branched on along a given path, then $J = \{x_{i-j/2}, x_{i-j/2+1}, \ldots, x_{i+j/2-1}\}$ or $J = \{x_{i-j/2+1}, x_{i-j/2+2}, \ldots, x_{i+j/2}\}$. Moreover, the number of variables set to 0 in J equals the number of variables set to 1. From Claim C.7, we know that in the first case, $x_{i+j/2}$ will be the next node B&B will branch on. This will create two new paths: $\{x_{i-j/2}, x_{i-j/2+1}, \ldots, x_{i+j/2}\}$ will be the set of variables branched along each path, and the number of variables set to 0 will equal the number of variables set to 1, plus or minus 1. Also from Claim C.7, we know that in the second case, $x_{i-j/2}$ will be the next node B&B will branch on. This will also create two new paths: $\{x_{i-j/2}, x_{i-j/2+1}, \ldots, x_{i+j/2}\}$ will be the set of variables branched along each path, and the number of variables set to 0 will equal the number of variables set to 1, plus or minus 1. Since this is true for all $2^{j/2}$ paths, this leads to a total of $2^{j/2+1} = 2^{(j+2)/2}$ paths, meaning the inductive hypothesis holds.

Next, suppose j is odd and let $j' = \lfloor j/2 \rfloor$. From the inductive hypothesis, we know that there exist at least $2^{(j+1)/2}$ paths in the B&B tree from the root to nodes of depth j such that if J is the set variables branched on along a given path, then $J = \{x_{i-j'}, x_{i-j'+1}, \ldots, x_{i+j'}\}$. Moreover, the number of variables set to 0 in J equals the number of variables set to 1, plus or minus 1. From Claim C.7, we know that B&B will either branch on $x_{i-j'-1}$ or $x_{i+j'+1}$. Suppose the number of variables set to 0 in J is 1 greater than the number of variables set to 1. If B&B branches on $x_{i-j'-1}$, we can follow the path where $x_{i-j'-1} = 1$, and this will give us a new path where

$$\{x_{i-j'-1}, x_{i-j'}, \ldots, x_{i+j'}\} = \{x_{i-(j+1)/2}, x_{i-(j+1)/2+1}, \ldots, x_{i+(j+1)/2-1}\}$$

are the variables branched on and the number of variables set to 0 equals the number of variables set to 1. If B&B branches on $x_{i+j'+1}$, we can follow the path where $x_{i+j'+1} = 1$, and this will give us a new path where

$$\{x_{i-j'}, x_{i-j'+1}, \ldots, x_{i+j'}, x_{i+j'+1}\} = \{x_{i-(j+1)/2+1}, x_{i-(j+1)/2+1}, \ldots, x_{i+(j+1)/2}\}$$

are the variables branched on and the number of variables set to 0 equals the number of variables set to 1. A symmetric argument holds if the number of variables set to 0 in J is 1 less than the number of variables set to 1. Therefore, all $2^{(j+1)/2}$ paths can be extended by one edge, so the statement holds. $\qquad\square$

$\square$

Theorem C.9. *Let Q be an infeasible MILP, let NSP and NSP' be two node selection policies, and let* `score` *be a path-wise scoring rule. The tree $\mathcal{T}$ B&B builds using NSP and* `score` *equals the tree $\mathcal{T}'$ it builds using NSP' and* `score`.

Proof. For a contradiction, suppose $\mathcal{T} \neq \mathcal{T}'$. There must be a node Q_0 in $\mathcal{T}$ where if $\mathcal{T}_{Q_0}$ is the path from the root of $\mathcal{T}$ to Q_0, then $\mathcal{T}_{Q_0}$ is a rooted subtree of $\mathcal{T}'$, but either:

1. In $\mathcal{T}$, the node Q_0 is fathomed but in $\mathcal{T}'$, Q_0 is not fathomed, or

2. In $\mathcal{T}$, the node Q_0 is not fathomed, but for all children Q_0' of Q_0 in $\mathcal{T}$, if $\mathcal{T}_{Q_0'}$ is the path from the root of $\mathcal{T}$ to Q_0', $\mathcal{T}_{Q_0'}$ is not a rooted subtree of $\mathcal{T}'$.

We will show that neither case is possible, thus arriving at a contradiction. First, we know that since Q is infeasible, B&B will only fathom a node if it is infeasible. Therefore, the first case is impossible: if Q_0 is fathomed in $\mathcal{T}$, it must be infeasible, so it will also be fathomed in $\mathcal{T}'$, and vice versa. Therefore, we know that B&B must branch on Q_0 in both $\mathcal{T}$ and $\mathcal{T}'$. Let $\bar{\mathcal{T}}$ be the state of the tree B&B has built using NSP and `score` by the time it branches on Q_0 and let $\bar{\mathcal{T}}'$ be

the state of the tree B&B has built using NSP' and `score` by the time it branches on Q_0. Since $\mathcal{T}_{Q_0}$ is a rooted subtree of both $\bar{\mathcal{T}}$ and $\bar{\mathcal{T}}'$, we know that for all variables x_i, $\texttt{score}(\bar{\mathcal{T}}, Q_0, i) = \texttt{score}(\mathcal{T}_Q, Q_0, i) = \texttt{score}(\bar{\mathcal{T}}', Q_0, i)$. Therefore, B&B will branch on the same variable in both $\mathcal{T}$ and $\mathcal{T}'$, which is a contradiction, since this means that for all children Q_0' of Q_0 in $\mathcal{T}$, if $\mathcal{T}_{Q_0'}$ is the path from the root of $\mathcal{T}$ to Q_0', $\mathcal{T}_{Q_0'}$ is a rooted subtree of $\mathcal{T}'$. $\qquad\square$

Lemma C.10 (Shalev-Shwartz and Ben-David [2014])**.** *Let $a \geq 1$ and $b > 0$. Then $x < a \log x + b$ implies that $x < 4a \log(2a) + 2b$.*

Claim C.11. *There are $T \leq 2^{n(n-1)/2} n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, the search tree A' builds using the scoring rule $\mu \texttt{score}_1 + (1-\mu) \texttt{score}_2$ is invariant across all $\mu \in I_j$.*[3]

Proof. We prove this claim by induction.

Inductive hypothesis. For $i \in \{1, \ldots, n\}$, there are $T \leq 2^{i(i-1)/2} n^i$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j and any two parameters $\mu, \mu' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees A' builds using the scoring rules $\mu \texttt{score}_1 + (1-\mu) \texttt{score}_2$ and $\mu' \texttt{score}_1 + (1-\mu') \texttt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$.

Base case. Before branching on any variables, the branch-and-bound tree $\mathcal{T}_0$ consists of a single root node Q. Given a parameter μ, A' will branch on variable x_k so long as

$$k = \operatorname{argmax}_{\ell \in [n]} \{\mu \texttt{score}_1(\mathcal{T}_0, Q, \ell) + (1 - \mu) \texttt{score}_2(\mathcal{T}_0, Q, \ell)\}.$$

Since $\mu \texttt{score}_1(\mathcal{T}_0, Q, \ell) + (1 - \mu) \texttt{score}_2(\mathcal{T}_0, Q, \ell)$ is a linear function of μ for each $\ell \in [n]$, we know that for any $k \in [n]$, there is at most one interval I of the parameter space $[0, 1]$ where $k = \operatorname{argmax}_{\ell \in [n]} \{\mu \texttt{score}_1 + (1 - \mu) \texttt{score}_2\}$. Thus, there are $T \leq n = 2^{1 \cdot (1-1)/2} n^1$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, A' branches on the same variable at the root node using the scoring rule $\mu \texttt{score}_1 + (1 - \mu) \texttt{score}_2$ across all $\mu \in I_j$.

Inductive step. Let $i \in \{2, \ldots, n\}$ be arbitrary. From the inductive hypothesis, we know that there are $T \leq 2^{(i-2)(i-1)/2} n^{i-1}$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j and any two parameters $\mu, \mu' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees A' builds using the scoring rules $\mu \texttt{score}_1 + (1 - \mu) \texttt{score}_2$ and $\mu' \texttt{score}_1 + (1 - \mu') \texttt{score}_2$, respectively, then $\mathcal{T}[i - 1] = \mathcal{T}'[i - 1]$. Consider an arbitrary node Q in $\mathcal{T}[i - 1]$ (or equivalently, $\mathcal{T}'[i - 1]$) at depth $i - 1$. If Q is integral or infeasible, then it will be fathomed no matter which parameter $\mu \in I_j$ the algorithm A' uses. Otherwise, for all $\mu \in I_j$, let $\mathcal{T}_\mu$ be the state of the search tree A' builds using the scoring rule $\mu \texttt{score}_1 + (1 - \mu) \texttt{score}_2$ at the point when it branches on Q. By the inductive hypothesis, we know that across all $\mu \in I_j$, the path from the root to Q in $\mathcal{T}_\mu$ is invariant, and we refer to this path as $\mathcal{T}_Q$. Given a parameter $\mu \in I_j$, the variable x_k will be branched on at node Q so long as $k = \operatorname{argmax}_\ell \{\mu \texttt{score}_1(\mathcal{T}_\mu, Q, \ell) + (1 - \mu) \texttt{score}_2(\mathcal{T}_\mu, Q, \ell)\}$, or equivalently, so long as $k = \operatorname{argmax}_\ell \{\mu \texttt{score}_1(\mathcal{T}_Q, Q, \ell) + (1 - \mu) \texttt{score}_2(\mathcal{T}_Q, Q, \ell)\}$. In other words, the decision of which variable to branch on is determined by a convex combination of the constant values $\texttt{score}_1(\mathcal{T}_Q, Q, \ell)$ and $\texttt{score}_2(\mathcal{T}_Q, Q, \ell)$ no matter which parameter $\mu \in I_j$ the algorithm A' uses. Here, we critically use the fact that the scoring rule is path-wise.

---

[3]This claim holds even when $\texttt{score}_1$ and $\texttt{score}_2$ are members of the more general class of *depth-wise* scoring rules, which we define as follows. For any search tree $\mathcal{T}$ of depth $\text{depth}(\mathcal{T})$ and any $j \in [n]$, let $\mathcal{T}[j]$ be the subtree of $\mathcal{T}$ consisting of all nodes in $\mathcal{T}$ of depth at most j. We say that $\texttt{score}$ is a *depth-wise* scoring rule if for all search trees $\mathcal{T}$, all $j \in [\text{depth}(\mathcal{T})]$, all nodes Q of depth j, and all variables x_i, $\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}[j], Q, i)$.

Since $\mu\texttt{score}_1(\mathcal{T}_Q, Q, \ell) + (1 - \mu)\texttt{score}_2(\mathcal{T}_Q, Q, \ell)$ is a linear function of μ for all ℓ, there are at most n intervals subdividing the interval I_j such that the variable branched on at node Q is fixed. Moreover, there are at most 2^{i-1} nodes at depth $i-1$, and each node similarly contributes a subpartition of I_j of size n. If we merge all 2^{i-1} partitions, we have $T' \leq 2^{i-1}(n-1) + 1$ intervals $I'_1, \ldots, I'_{T'}$ partitioning I_j where for any interval I'_p and any two parameters $\mu, \mu' \in I'_p$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees A' builds using the scoring rules $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ and $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. We can similarly subdivide each interval $I_1, \ldots, I_T$ for a total of

$$\bar{T} \leq 2^{(i-1)(i-2)/2}n^{i-1}\left(2^{i-1}(n-1) + 1\right) \leq 2^{(i-1)(i-2)/2}n^{i-1}\left(2^{i-1}n\right) = 2^{i(i-1)/2}n^i$$

intervals $\bar{I}_1, \ldots, \bar{I}_{\bar{T}}$ partitioning $[0, 1]$ such that for any interval $\bar{I}_t$, across all $\mu \in \bar{I}_t$ and any two parameters $\mu, \mu' \in \bar{I}_t$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees A' builds using the scoring rules $\mu\texttt{score}_1 + (1 - \mu)\texttt{score}_2$ and $\mu'\texttt{score}_1 + (1 - \mu')\texttt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. $\qquad\square$

C.1 Empirical Risk Minimization Algorithm

In this section we describe an empirical risk minimization algorithm capable of finding the best mixture of two variable selection scoring rules for a given set of problem instances $\Pi^{(1)}, \ldots, \Pi^{(m)}$. We modify the tree search algorithm so that given a problem instance Π and a mixing paramter $\mu \in [0, 1]$, the search algorithm keeps track of the largest interval $I \subset [0, 1]$ such that the behavior of the algorithm is identical to the current run when run with any parameter $\mu' \in I$. With this, we can enumerate all possible behaviors of the algorithm for a single instance Π by running the algorithm with $\mu = 0$, followed by the smallest value of μ that will give a different outcome, and so on, until we have covered the entire interval $[0, 1]$. This procedure results in running the tree search algorithm on the instance Π exactly once for each possible behavior achievable across all values of the parameter $\mu \in [0, 1]$. By applying this algorithm to each problem $\Pi^{(i)}$ for $i = 1, \ldots, m$, we discover how the tree search algorithm would perform on every instance for any value of the mixing parameter μ. This allows us to divide the interval $[0, 1]$ into a finite number of intervals on which $\texttt{cost}$ is piecewise constant, and to compute the cost on each interval.

To see why this additional book keeping is possible, suppose we are choosing which variable to branch on in node Q of tree $\mathcal{T}$. We have two scoring rules $\texttt{score}_1$ and $\texttt{score}_2$ that each rank the candidate variables in Q, and when we run the algorithm with parameter μ, we combine these two scores as $(1 - \mu)\texttt{score}_1(\mathcal{T}, Q, i) + \mu\texttt{score}_2(\mathcal{T}, Q, i)$. For any parameter μ which results in the same variable having the highest score, the variable chosen for branching in this node will be identical. Let i^* be the variable chosen by the algorithm when run with parameter μ. The set of all μ' for which i^* is the variable of the highest score is an interval (and its end points can be found by solving a linear equation to determine the value of μ' for which some other variable overtakes i^* under the mixed score). Also, for every parameter μ' outside of this interval, the algorithm would indeed branch on a different node, resulting in a different outcome of the tree search algorithm. By taking the intersections of these intervals across all branching variable choices, we find the largest subset of $[0, 1]$ for which the algorithm would behave exactly the same, and this subset is an interval. The overhead of this book keeping is only linear in the number of candidate branch variables.

Psuedo-code for the ERM algorithm is given in Algorithm 3.

D Additional information about experiments

Facility location. Suppose there is a set I of customers and a set J of facilities that have not yet been built. The facilities each produce the same good, and each consumer demands one unit of

Algorithm 3 ERM Algorithm

Input: Problem instances $\Pi^{(1)}, \ldots, \Pi^{(m)}$, variable scoring rules $\texttt{score}_1, \texttt{score}_2$.

1: For each problem instance $\Pi^{(i)}$, compute the piecewise constant $\texttt{cost}$, as a function of the mixing parameter μ. (See discussion in Section C.1).

2: Compute the point-wise average of the resulting piecewise constant functions.

Output: The μ^* in the interval minimizing the average cost.

---

that good. Consumer i can obtain some fraction y_{ij} of the good from facility j, which costs them $d_{ij}y_{ij}$. Moreover, it costs f_j to construct facility j. The goal is to choose a subset of facilities to construct while minimizing total cost. We can model this problem as a MILP by assigning a binary variable x_j to each facility, which represents whether or not it is built. The optimization is then over the variables x_j and y_{ij}, as follows.

$$
\begin{array}{lll}
\text{minimize} & \sum_{j \in J} f_j x_j + \sum_{j \in J, i \in I} d_{ij} y_{ij} & \\
\text{s.t.} & \sum_{j \in J} y_{ij} = 1 & \forall i \in I \\
& y_{ij} \leq x_j & \forall i \in I, j \in J \\
& x_j \in \{0,1\} & \forall j \in J \\
& y_{ij} \in [0,1] & \forall i \in I, j \in J.
\end{array}
$$

Clustering. We can formulate k-means clustering as a MILP by assigning a binary variable x_i to each point p_i where $x_i = 1$ if and only if p_i is a center, as well as a binary variable y_{ij} for each pair of points p_i and p_j, where $y_{ij} = 1$ if and only if p_j is a center and p_j is the closest center to p_i. We want to solve the following problem:

$$
\begin{array}{lll}
\text{min} & \sum_{i,j \in [n]} d\left(p_i, p_j\right) y_{ij} & \\
\text{s.t.} & \sum_{i=1}^{n} x_i = k & \\
& \sum_{j=1}^{n} y_{ij} = 1 & \forall i \in [n] \\
& y_{ij} \leq x_j & \forall i, j \in [n] \\
& x_i \in \{0,1\} & \forall i \in [n] \\
& y_{ij} \in \{0,1\} & \forall i, j \in [n].
\end{array}
$$

Agnostically learning linear separators. We can formulate this problem as a MILP as follows. Let $M > \max \|\boldsymbol{p}_i\|_1$.

$$
\begin{array}{lll}
\text{min} & \sum_{i=1}^{n} x_i & \\
\text{s.t.} & z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle > -M x_i & \forall i \in [n] \\
& w[i] \in [-1, 1] & \forall i \in [n] \\
& x_i \in \{0,1\} & \forall i \in [n].
\end{array}
$$

Since $|\langle \boldsymbol{p}_i, \boldsymbol{w} \rangle| < M$, the inequality $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle > -M x_i$ ensures that if $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle > 0$, then x_i will equal 0, but if $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle \leq 0$, then x_i must equal 1.[4]

Theorem D.1. *Let $\texttt{cost}$ be a tree-constant cost function, let $\texttt{score}_1$ and $\texttt{score}_2$ be two path-wise scoring rules, and let $\mathcal{S}$ be a set of m problem instances over n binary variables. Let $\mathcal{C}$ be the set of functions $\{\texttt{cost}\left(\cdot, \mu \texttt{score}_1 + (1-\mu)\texttt{score}_2\right) : \mu \in [0,1]\}$. Then*

$$
\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{C}) \leq \kappa \sqrt{\frac{1}{m}\left(n^2 + 2n \log n + 2 \log m\right)}.
$$

---

[4]In practice, we implement this constraint by enforcing that $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle \geq -M x_i + \gamma$ for some tiny $\gamma > 0$ since MILP solvers cannot enforce strict inequalities.

Algorithm 4 Tree search

Input: A problem instance $\Pi = (X, D, f, g)$.

1: Let $\mathcal{T}$ be a tree that consists of a single node containing the empty partial solution $(\top, \ldots, \top)$.
2: **while** there remains an unfathomed leaf in $\mathcal{T}$ **do**
3:    Use a *node selection policy* to select a leaf of the tree $\mathcal{T}$. Let $\boldsymbol{y}$ be the partial solution contained in that leaf.
4:    Use a *variable selection policy* to choose a variable $x_i \in X$ to branch on at that leaf.
5:    For all $j \in D_i$, let $\boldsymbol{y}_i^{(j)}$ be the partial solution $\boldsymbol{y}$ except with the component $y[i] = j$.
6:    Create $|D_i|$ children of the node containing the partial solution $\boldsymbol{y}$, where the jth child contains the partial solution $\boldsymbol{y}_i^{(j)}$. Let $\mathcal{T}'$ be the resulting search tree.
7:    **for** $j \in D_i$ **do**
8:       **if** $\texttt{localFathom}(\Pi, \boldsymbol{y}_i^{(j)}) = \text{fathom}$ **then**
9:          Update $\mathcal{T}'$ so that the leaf containing the partial solution $\boldsymbol{y}_i^{(j)}$ is fathomed.
10:       **else if** $\texttt{globalFathom}(\Pi, \boldsymbol{y}_i^{(j)}, \mathcal{T}') = \text{fathom}$ **then**
11:          Update $\mathcal{T}'$ so that the leaf containing the partial solution $\boldsymbol{y}_i^{(j)}$ is fathomed.
12:    Set $\mathcal{T} = \mathcal{T}'$.

Output: The best known feasible solution $\boldsymbol{y}^*$, if one exists. Otherwise, return $\texttt{Null}$.

Proof. To prove this theorem, we rely on Lemma 3.3, which tells us that for any problem instance $\Pi \in \mathcal{S}$, there are $T \leq 2^{n(n-1)/2} n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, across all $\mu \in I_j$, the scoring rule $\mu\texttt{score}_1 + (1 - \mu)\texttt{score}_2$ results in the same search tree. Theorem D.1 then follows from this lemma and Massart's lemma [Massart, 2000], which tells us that for any class $\mathcal{F}$ of functions with range $[0, c]$, if $N = |\mathcal{F}(\mathcal{S})|$, then $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) \leq c\sqrt{\frac{2 \log N}{m}}$. Lemma 3.3 allows us to apply Massart's lemma with $N \leq \left| 2^{n^2/2} n^n m \right|$ and $c = \kappa$, since so long as the search tree is fixed for all μ in an interval I, we know that $\texttt{cost}(\Pi, \mu\texttt{score}_1 + (1 - \mu)\texttt{score}_2)$ is constant for all $\mu \in I$. $\qquad\square$

E   Tree search

A *tree search algorithm* takes as input a tuple $\Pi = (X, D, f, g)$, where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of domains where D_i is the finite set of values variable x_i can take on, $f : D_1 \times \cdots \times D_n \to \{0, 1\}$ is a feasibility function, and $g : D_1 \times \cdots \times D_n \to \mathbb{R}$ is an objective function (if the problem is a satisfiability problem, rather than an optimization problem, we set g to be the constant zero function). We use $\boldsymbol{y} \in (D_1 \cup \{\top\}) \times \cdots \times (D_n \cup \{\top\})$ to denote a partial solution to the problem instance Π, where $y[i]$ is an assignment of the variable x_i and if $y[i] = \top$, it means the variable x_i has not yet been assigned a value.

Tree search builds a tree of partial solutions to Π until it finds the optimal solution. We define two fathoming functions tree search can use to prune branches of the search tree, $\texttt{localFathom}$ and $\texttt{globalFathom}$. The function $\texttt{localFathom}(\Pi, \boldsymbol{y}) \in \{\text{fathom}, \text{explore}\}$ takes as input an instance Π and a partial solution $\boldsymbol{y}$ and determines whether or not to fathom the node containing the partial solution $\boldsymbol{y}$. Its output is only based on the local information contained in the partial solution $\boldsymbol{y}$, not the remainder of the search tree. For example, in MIP, $\texttt{localFathom}(\Pi, \boldsymbol{y}) = \text{fathom}$ if the LP relaxation of the MIP given the partial solution $\boldsymbol{y}$ is integral or infeasible. The function $\texttt{globalFathom}(\Pi, \boldsymbol{y}, \mathcal{T}) \in \{\text{fathom}, \text{explore}\}$ takes as input an instance Π, a partial solution $\boldsymbol{y}$, and

a partial search tree $\mathcal{T}$ and determines whether or not to fathom the node in $\mathcal{T}$ containing the partial solution $\boldsymbol{y}$. For example, in MIP, the function `globalFathom` covers the case where a node is fathomed because LP relaxation's objective value given the partial solution contained in that node is no better than the objective value evaluated on the best known integral solution. In a bit more detail, suppose there is a fathomed leaf node in $\mathcal{T}$ containing a partial solution $\boldsymbol{y}^*$ such that the LP relaxation of the MIP given the partial solution $\boldsymbol{y}^*$ is integral, and let c^* be the objective value. Let c be the objective value of the LP relaxation of the MIP given the partial solution $\boldsymbol{y}$. We know that `globalFathom`$(\Pi, \boldsymbol{y}, \mathcal{T}) =$ fathom if $c \leq c^*$.

See Algorithm 4 for the tree search pseudo-code.

E.1 Problem statement

The problem statement for general tree search is nearly identical to that in Section 3.1. We state it hear for clarity's sake.

Let $\mathcal{D}$ be a distribution over problem instances Π. Let $\mathtt{score}_1, \dots, \mathtt{score}_d$ be a set of variable selection scoring rules, such as those in Section 5.0.2. Our goal is to learn a convex combination $\mu_1\mathtt{score}_1 + \dots + \mu_d\mathtt{score}_d$ of the scoring rules that is nearly optimal in expectation over $\mathcal{D}$. More formally, let `cost` be an abstract cost function that takes as input a problem instance Π and a scoring rule `score` and returns some measure of the quality of tree search using `score` on input Π. We say that an algorithm (ϵ, δ)-learns a convex combination of the d scoring rules $\mathtt{score}_1, \dots, \mathtt{score}_d$ if for any distribution $\mathcal{D}$, with probability at least $1 - \delta$ over the draw of a sample $\{\Pi_1, \dots, \Pi_m\} \sim \mathcal{D}^m$, the algorithm returns a convex combination $\mathtt{score} = \hat{\mu}_1\mathtt{score}_1 + \dots + \hat{\mu}_d\mathtt{score}_d$ such that $\mathbb{E}_{\Pi \sim \mathcal{D}}\big[\mathtt{cost}(\Pi, \mathtt{score})\big] - \mathbb{E}_{\Pi \sim \mathcal{D}}\big[\mathtt{cost}(\Pi, \mathtt{score}^*)\big] \leq \epsilon$, where $\mathtt{score}^*$ is the convex combination of $\mathtt{score}_1, \dots, \mathtt{score}_d$ with minimal expected cost. In this work, we prove that only a small number of samples is sufficient to ensure (ϵ, δ)-learnability.

We assume that the problem instances in the support of $\mathcal{D}$ are over n D-ary variables, for some $n, D \in \mathbb{N}$.[5]

Our results hold for cost functions that are *tree-constant*, which means that for any problem instance Π, so long as the scoring rules $\mathtt{score}_1$ and $\mathtt{score}_2$ result in the same search tree, $\mathtt{cost}(\Pi, \mathtt{score}_1) = \mathtt{cost}(\Pi, \mathtt{score}_2)$. For example, the size of the search tree is tree-constant.

E.2 Path-wise scoring rules

We now slightly tweak the definition of *score-based variable selection policies* and *path-wise scoring rules* so that they apply to tree search more generally. The only difference is that a scoring rule will now be defined in terms of a partial solution, rather than a MILP.

Definition E.1 (Score-based variable selection policy). Let `score` be a function that takes as input a partial search tree $\mathcal{T}$, a partial solution $\boldsymbol{y}$ contained in a leaf of $\mathcal{T}$, and an index i and returns a real value ($\mathtt{score}(\mathcal{T}, Q, i) \in \mathbb{R}$). For a partial solution $\boldsymbol{y}$ contained in a leaf of a tree $\mathcal{T}$, let $N_{\mathcal{T}, \boldsymbol{y}}$ be the set of variables that have not yet been branched on along the path from the root of $\mathcal{T}$ to the leaf. A score-based variable selection policy selects the variable $\operatorname{argmax}_{x_j \in N_{\mathcal{T}, \boldsymbol{y}}}\{\mathtt{score}(\mathcal{T}, \boldsymbol{y}, j)\}$ to branch on at the node Q.

Definition E.2 (Path-wise scoring rule). Suppose $\boldsymbol{y}$ is a partial solution contained in the node of a search tree $\mathcal{T}$. We say that $\mathtt{score}(\mathcal{T}, \boldsymbol{y}, i)$ is a path-wise scoring rule if the value of $\mathtt{score}(\mathcal{T}, \boldsymbol{y}, i)$ depends only on the node Q, the variable x_i, and the path from the root of $\mathcal{T}$ to the node containing $\boldsymbol{y}$, which we denote as $\mathcal{T}_{\boldsymbol{y}}$. Specifically, $\mathtt{score}(\mathcal{T}, \boldsymbol{y}, i) = \mathtt{score}(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, i)$.

---

[5]A variable is D-ary if it can take on at most D distinct values.

The following lemma parallels Lemma 3.3.

Lemma E.1. *Let* `cost` *be a tree-constant cost function, let* `score₁` *and* `score₂` *be two path-wise scoring rules, and let* Π *be an arbitrary problem instance over n D-ary variables. There are $T \leq D^{n(n-1)/2}n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval I_j, across all $\mu \in I_j$, the scoring rule $\mu \texttt{score}_1 + (1-\mu)\texttt{score}_2$ results in the same search tree.*

Proof. We prove this lemma first by considering the actions of an alternative algorithm TS' which runs exactly like Algorithm 4 except it does not use the function `globalFathom`, but rather skips Steps 10 and 11 of Algorithm 4. We then relate the behavior of TS' to the behavior of Algorithm 4 to prove the lemma.

First, we prove the following bound on the number of search trees TS' will build on a given instance over the entire range of parameters. Note that this bound matches that in the lemma statement.

Claim E.2. *There are $T \leq D^{n(n-1)/2}n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval I_j, the search tree TS' builds using the scoring rule $\mu \texttt{score}_1 + (1-\mu)\texttt{score}_2$ is invariant across all $\mu \in I_j$.*

Proof. We prove this claim by induction.

Inductive hypothesis. For $i \in \{1, \ldots, n\}$, there are $T \leq D^{i(i-1)/2}n^i$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval I_j and any two parameters $\mu, \mu' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees TS' builds using the scoring rules $\mu \texttt{score}_1 + (1-\mu)\texttt{score}_2$ and $\mu' \texttt{score}_1 + (1-\mu')\texttt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$.

Base case. Before branching on any variables, the seasrch tree $\mathcal{T}_0$ consists of a single root node containing the empty partial solution $\boldsymbol{y} = (\top, \ldots, \top)$. Given a parameter μ, TS' will branch on variable x_k so long as

$$k = \text{argmax}_{\ell \in [n]} \left\{ \mu \texttt{score}_1(\mathcal{T}_0, \boldsymbol{y}, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_0, \boldsymbol{y}, \ell) \right\}.$$

Since $\mu \texttt{score}_1(\mathcal{T}_0, \boldsymbol{y}, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_0, \boldsymbol{y}, \ell)$ is a linear function of μ for each $\ell \in [n]$, we know that for any $k \in [n]$, there is at most one interval I of the parameter space $[0,1]$ where $k = \text{argmax}_{\ell \in [n]} \{ \mu \texttt{score}_1 + (1-\mu)\texttt{score}_2 \}$. Thus, there are $T \leq n = D^{1 \cdot (1-1)/2}n^1$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval I_j, TS' branches on the same variable at the root node using the scoring rule $\mu \texttt{score}_1 + (1-\mu)\texttt{score}_2$ across all $\mu \in I_j$.

Inductive step. Let $i \in \{2, \ldots, n\}$ be arbitrary. From the inductive hypothesis, we know that there are $T \leq D^{(i-2)(i-1)/2}n^{i-1}$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval I_j and any two parameters $\mu, \mu' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees TS' builds using the scoring rules $\mu \texttt{score}_1 + (1-\mu)\texttt{score}_2$ and $\mu' \texttt{score}_1 + (1-\mu')\texttt{score}_2$, respectively, then $\mathcal{T}[i-1] = \mathcal{T}'[i-1]$. Consider an arbitrary node containing a partial solution $\boldsymbol{y}$ in $\mathcal{T}[i-1]$ (or equivalently, $\mathcal{T}'[i-1]$) at depth $i-1$. If `localFathom`$(\Pi, \boldsymbol{y}) = $ fathom, then it will be fathomed no matter which parameter $\mu \in I_j$ the algorithm TS' uses. Otherwise, for all $\mu \in I_j$, let $\mathcal{T}_\mu$ be the state of the search tree TS' builds using the scoring rule $\mu \texttt{score}_1 + (1-\mu)\texttt{score}_2$ at the point when it branches on the node containing $\boldsymbol{y}$. By the inductive hypothesis, we know that across all $\mu \in I_j$, the path from the root to this node in $\mathcal{T}_\mu$ is invariant, and we refer to this path as $\mathcal{T}_{\boldsymbol{y}}$. Given a parameter $\mu \in I_j$, the variable x_k will be branched on at this node so long as $k = \text{argmax}_\ell \{ \mu \texttt{score}_1(\mathcal{T}_\mu, \boldsymbol{y}, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_\mu, \boldsymbol{y}, \ell) \}$, or equivalently, so long as $k = \text{argmax}_\ell \{ \mu \texttt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell) \}$. In other words,

the decision of which variable to branch on is determined by a convex combination of the constant values $\mathtt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell)$ and $\mathtt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell)$ no matter which parameter $\mu \in I_j$ the algorithm TS' uses. Here, we critically use the fact that the scoring rule is path-wise.

Since $\mu\mathtt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell) + (1-\mu)\mathtt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell)$ is a linear function of μ for all ℓ, there are at most n intervals subdividing the interval I_j such that the variable branched on at the node containing $\boldsymbol{y}$ is fixed. Moreover, there are at most D^{i-1} nodes at depth $i-1$, and each node similarly contributes a subpartition of I_j of size n. If we merge all D^{i-1} partitions, we have $T' \leq D^{i-1}(n-1)+1$ intervals $I'_1, \ldots, I'_{T'}$ partitioning I_j where for any interval I'_p and any two parameters $\mu, \mu' \in I'_p$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees TS' builds using the scoring rules $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ and $\mu'\mathtt{score}_1 + (1-\mu')\mathtt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. We can similarly subdivide each interval $I_1, \ldots, I_T$ for a total of

$$\bar{T} \leq D^{(i-1)(i-2)/2}n^{i-1}\left(D^{i-1}(n-1)+1\right) \leq D^{(i-1)(i-2)/2}n^{i-1}\left(D^{i-1}n\right) = D^{i(i-1)/2}n^i$$

intervals $\bar{I}_1, \ldots, \bar{I}_{\bar{T}}$ partitioning $[0,1]$ such that for any interval $\bar{I}_t$, across all $\mu \in \bar{I}_t$ and any two parameters $\mu, \mu' \in \bar{I}_t$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees TS' builds using the scoring rules $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ and $\mu'\mathtt{score}_1 + (1-\mu')\mathtt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. $\qquad\square$

Next, we explicitly relate the behavior of Algorithm 4 to TS', proving that the search tree Algorithm 4 builds is a rooted subtree of the search tree TS' builds.

Claim E.3. *Given a parameter $\mu \in [0,1]$, let $\mathcal{T}$ and $\mathcal{T}'$ be the trees Algorithm 4 and TS' build, respectively, using the scoring rule $\mu\mathbf{score}_1 + (1-\mu)\mathbf{score}_2$. For an arbitrary node of $\mathcal{T}$, let $\boldsymbol{y}$ be the partial solution contained in that node and let $\mathcal{T}_{\boldsymbol{y}}$ be the path from the root of $\mathcal{T}$ to the node. Then $\mathcal{T}_{\boldsymbol{y}}$ is a rooted subtree of $\mathcal{T}'$.*

Proof of Claim E.3. Note that the path $\mathcal{T}_{\boldsymbol{y}}$ can be labeled by a sequence of indices from $\{1, \ldots, D\}$ and a sequence of variables from $\{x_1, \ldots, x_n\}$ describing which variable is branched on and which value it takes on along the path $\mathcal{T}_{\boldsymbol{y}}$. Let $((j_1, x_{i_1}), \ldots, (j_t, x_{i_t}))$ be this sequence of labels, where t is the number of edges in $\mathcal{T}_{\boldsymbol{y}}$. We can similarly label every edge in $\mathcal{T}'$. We claim that there exists a path beginning at the root of $\mathcal{T}'$ with the labels $((j_1, x_{i_1}), \ldots, (j_t, x_{i_t}))$.

For a contradiction, suppose no such path exists. Let (j_τ, x_{i_τ}) be the earliest label in the sequence $((j_1, x_{i_1}), \ldots, (j_t, x_{i_t}))$ where there is a path beginning at the root of $\mathcal{T}'$ with the labels $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$, but there is no way to continue the path using an edge labeled (j_τ, x_{i_τ}). There are exactly two reasons why this could be the case:

1. The node at the end of the path with labels $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$ was fathomed by TS'.

2. The algorithm TS' branched on a variable other than x_{i_τ} at the end of the path labeled $((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}}))$.

Let $\boldsymbol{y}'$ be the partial solution contained in the node at the end of the path with labels

$$((j_1, x_{i_1}), \ldots, (j_{\tau-1}, x_{i_{\tau-1}})).$$

We refer to this path as $\mathcal{T}_{\boldsymbol{y}'}$. In the first case, since TS' only fathoms the node containing the partial solution $\boldsymbol{y}'$ if $\mathtt{localFathom}(\Pi, \boldsymbol{y}') = \text{fathom}$, we know that Algorithm 4 will also fathom this node. However, this is not the case since Algorithm 4 next branches on the variable x_{i_τ}.

The second case is also not possible since the scoring rules are both path-wise. Let $\bar{\mathcal{T}}$ (respectively, $\bar{\mathcal{T}}'$) be the state of the search tree Algorithm 4 (respectively, A') has built at the point it branches on the node containing $\boldsymbol{y}'$. We know that $\mathcal{T}_{\boldsymbol{y}'}$ is the path from the root this node in both of the trees $\bar{\mathcal{T}}$ and $\bar{\mathcal{T}}'$. Therefore, for all variables x_k, $\mu\mathtt{score}_1(\bar{\mathcal{T}}, \boldsymbol{y}', k) + (1-\mu)\mathtt{score}_2(\bar{\mathcal{T}}, \boldsymbol{y}', k) =$

$\mu\mathtt{score}_1(\mathcal{T}_{\boldsymbol{y}'}, \boldsymbol{y}', k) + (1 - \mu)\mathtt{score}_2(\mathcal{T}_{\boldsymbol{y}'}, \boldsymbol{y}', k) = \mu\mathtt{score}_1(\bar{\mathcal{T}}', \boldsymbol{y}', k) + (1 - \mu)\mathtt{score}_2(\bar{\mathcal{T}}', \boldsymbol{y}', k)$. This means that Algorithm 4 and TS' will choose the same variable to branch on at the node containing the partial solution $\boldsymbol{y}'$.

Therefore, we have reached a contradiction, so the claim holds. □

Next, we use Claims E.2 and E.3 to prove Lemma E.1. Let $I_1, \ldots, I_T$ be one of the interval guaranteed to exist by Claim E.2 and let I_t be an arbitrary one of the intervals. Let μ' and μ'' be two arbitrary parameters from I_t. We will prove that the scoring rules $\mu'\mathtt{score}_1 + (1 - \mu')\mathtt{score}_2$ and $\mu''\mathtt{score}_1 + (1 - \mu'')\mathtt{score}_2$ result in the same search tree. For a contradiction, suppose that this is not the case. Consider the first iteration where of Algorithm 4 using the scoring rule $\mu'\mathtt{score}_1 + (1 - \mu')\mathtt{score}_2$ differs from Algorithm 4 using the scoring rule $\mu''\mathtt{score}_1 + (1 - \mu'')\mathtt{score}_2$, where an iteration corresponds to lines 3 through 12 of Algorithm 4. Up until this iteration, Algorithm 4 has built the same partial search tree $\mathcal{T}$. Since the node selection policy does not depend on μ' or μ'', Algorithm 4 will choose the same leaf of the search tree to branch on no matter which scoring rule it uses. Let $\boldsymbol{y}$ be the partial solution contained in this leaf.

Suppose Algorithm 4 chooses different variables to branch on depending on the scoring rule. Let $\mathcal{T}_{\boldsymbol{y}}$ be the path from the root of $\mathcal{T}$ to the node containing the partial solution $\boldsymbol{y}$. By Claim E.2, we know that the algorithm TS' builds the same search tree using the two scoring rules. Let $\bar{\mathcal{T}}'$ (respectively, $\bar{\mathcal{T}}''$) be the state of the search tree TS' has built using the scoring rule $\mu'\mathtt{score}_1 + (1 - \mu')\mathtt{score}_2$ (respectively, $\mu''\mathtt{score}_1 + (1 - \mu'')\mathtt{score}_2$) by the time it branches on the node containing the partial solution $\boldsymbol{y}$. By Claims E.2 and E.3, we know that $\mathcal{T}_{\boldsymbol{y}}$ is the path of from the root to the node containing the partial solution $\boldsymbol{y}$ of both $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$. By Claim E.2, we know that TS' will branch on the same variable x_i at this node in both the trees $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$, so $i = \mathrm{argmax}_j \left\{\mu'\mathtt{score}_1(\bar{\mathcal{T}}', \boldsymbol{y}, j) + (1 - \mu')\mathtt{score}_2(\bar{\mathcal{T}}', \boldsymbol{y}, j)\right\}$, or equivalently,

$$i = \mathrm{argmax}_j \left\{\mu'\mathtt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j) + (1 - \mu')\mathtt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j)\right\}, \tag{5}$$

and $i = \mathrm{argmax}_j \left\{\mu''\mathtt{score}_1(\bar{\mathcal{T}}'', \boldsymbol{y}, j) + (1 - \mu'')\mathtt{score}_2(\bar{\mathcal{T}}'', \boldsymbol{y}, j)\right\}$, or equivalently,

$$i = \mathrm{argmax}_j \left\{\mu''\mathtt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j) + (1 - \mu'')\mathtt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j)\right\}. \tag{6}$$

Returning to the search tree $\mathcal{T}$ that Algorithm 4 is building, Equation (5) implies that

$$i = \mathrm{argmax}_j \left\{\mu'\mathtt{score}_1(\mathcal{T}, \boldsymbol{y}, j) + (1 - \mu')\mathtt{score}_2(\mathcal{T}, \boldsymbol{y}, j)\right\}$$

and Equation (6) implies that $i = \mathrm{argmax}_j \left\{\mu''\mathtt{score}_1(\mathcal{T}, \boldsymbol{y}, j) + (1 - \mu'')\mathtt{score}_2(\mathcal{T}, \boldsymbol{y}, j)\right\}$. Therefore, Algorithm 4 will branch on x_i at the node containing the partial solution $\boldsymbol{y}$ no matter which scoring rule it uses.

Finally, since `localFathom` and `globalFathom` do not depend on the parameter μ, whether or not Algorithm 4 fathoms any of the nodes in Steps 7 through 11 does not depend on μ' or μ''.

We have reached a contradiction by showing that the two iterations of Algorithm 4 are identical. Therefore, the lemma holds. □

E.2.1 General scoring rules

In this section, we provide generalization guarantees that apply to learning convex combinations of any set of scoring rules, as in Section 3.3.2. The following lemma corresponds to Lemma E.1 for this setting.

46

Lemma E.4. *Let* cost *be a tree-constant cost function, let* $score_1, \ldots, score_d$ *be d arbitrary scoring rules, and let Π be an arbitrary problem instance over n D-ary variables. Suppose we limit Algorithm 4 to producing search trees of size $\bar{\kappa}$. There is a set $\mathcal{H}$ of at most $n^{2(\bar{\kappa}+1)}$ hyperplanes such that for any connected component R of $[0,1]^d \setminus \mathcal{H}$, the search tree Algorithm 4 builds using the scoring rule $\mu_1 score_1 + \cdots + \mu_d score_d$ is invariant across all $(\mu_1, \ldots, \mu_d) \in R$.*

Proof. The proof has two steps. In Claim E.5, we show that there are at most $n^{\bar{\kappa}}$ different search trees that Algorithm 4 might produce for the instance Π as we vary the mixing parameter vector $(\mu_1, \ldots, \mu_d)$. In Claim E.6, for each of the possible search trees $\mathcal{T}$ that might be produced, we show that the set of parameter values $(\mu_1, \ldots, \mu_d)$ which give rise to that tree lie in the intersection of $n^{\bar{\kappa}+2}$ halfspaces. These facts together prove the lemma.

Claim E.5. *There are only $n^{\bar{\kappa}}$ different search trees that can be achieved by varying the parameter vector $(\mu_1, \ldots, \mu_d)$.*

Proof of Claim E.5. Fix any d mixing parameters $(\mu_1, \ldots, \mu_d)$ and let $v_1, \ldots, v_{\bar{\kappa}} \in [n]$ be the sequence of branching variables chosen by Algorithm 4 run with scoring rule $\mu_1 score_1 + \cdots + \mu_d score_d$, ignoring which node of the tree each variable was chosen for. That is, v_1 is the variable branched on at the root, v_2 is the variable branched on at the next unfathomed node chosen by the node selection policy, and so on. If Algorithm 4 with scoring rule $\mu_1 score_1 + \cdots + \mu_d score_d$ produces a tree of size $k < \bar{\kappa}$, then define $v_t = 1$ for all $t \geq k$ (we are just padding the sequence $v_1, v_2, \ldots$ so that it has length $\bar{\kappa}$). We will show that whenever two sets of mixing parameters $(\mu_1, \ldots, \mu_d)$ and $(\mu'_1, \ldots, \mu'_d)$ give rise to the same sequence of branching variable selections, they in fact produce identical search trees. This will imply that the number of distinct trees that can be produced by Algorithm 4 with scoring rules of the form $\mu_1 score_1 + \cdots + \mu_d score_d$ is at most $n^{\bar{\kappa}}$, since there are only $n^{\bar{\kappa}}$ distinct sequences of $\bar{\kappa}$ variables $v_1, \ldots, v_{\bar{\kappa}} \in [n]$.

Let $(\mu_1, \ldots, \mu_d)$ and $(\mu'_1, \ldots, \mu'_d)$ be two sets of mixing parameters, and suppose running Algorithm 4 with $\mu_1 score_1 + \cdots + \mu_d score_d$ and $\mu'_1 score_1 + \cdots + \mu'_d score_d$ both results in the sequence of branching variable decisions being $v_1, \ldots, v_{\bar{\kappa}}$. We prove that the resulting search trees are identical by induction on the iterations of the algorithm, where an iteration corresponds to lines 3 through 12 of Algorithm 4. Our base case is before the first iteration when the two trees are trivially equal, since they both contain just the root node. Now suppose that up until the beginning of iteration t the two trees were identical. Since the two trees are identical, the node selection policy will choose the same node to branch on in both cases. In both trees, the algorithm will choose the same variable to branch on, since the sequence of branching variable choices $v_1, \ldots, v_{\bar{\kappa}}$ is shared. Finally, if any of the children are fathomed, they will be fathomed in both trees, since the they are identical. It follows that all steps of Algorithm 4 maintain equality between the two trees, and the claim follows. Also, whenever the sequence of branching variables differ, then the search tree produced will not be the same. In particular, on the first iteration where the two sequences disagree, the tree built so far will be identical up to that point, but the next variable branched on will be different, leading to different trees. $\square$

Next, we argue that for any given search tree $\mathcal{T}$ produced by Algorithm 4, the set of mixing parameters $(\mu_1, \ldots, \mu_d)$ giving rise to $\mathcal{T}$ is defined by the intersection of $n^{\bar{\kappa}+2}$ halfspaces.

Claim E.6. *For a given search tree $\mathcal{T}$, there are at most $n^{\bar{\kappa}+2}$ halfspaces such that Algorithm 4 using the scoring rule $\mu_1 score_1 + \cdots + \mu_d score_d$ builds the tree $\mathcal{T}$ if and only if $(\mu_1, \ldots, \mu_d)$ lies in the intersection of those halfspaces.*

Proof of Claim E.6. Let $v_1, \dots, v_{\bar{\kappa}}$ be the sequence of branching variable choices that gives rise to tree $\mathcal{T}$. We will prove the claim by induction on iterations completed by Algorithm 4. Let $\mathcal{T}_t$ be the state of Algorithm 4 after t iterations.

Induction hypothesis. For a given index $t \in [\bar{\kappa}]$, there are at most tn^2 halfspaces such that Algorithm 4 using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ builds the partial tree $\mathcal{T}_t$ after t iterations if and only if $(\mu_1, \dots, \mu_d)$ lies in the intersection of those halfspaces.

Base case. In the base case, before the first iteration, the set of parameters that will produce the partial search tree consisting of just the root is the entire set of parameters, which vacuously is the intersection of zero hyperplanes.

Inductive step. For the inductive step, let $t < \bar{\kappa}$ be an arbitrary tree size. By the inductive hypothesis, we know that there exists a set $\mathcal{B}$ of at most tn^2 halfspaces such that Algorithm 4 using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ builds the partial tree $\mathcal{T}_t$ after t iterations if and only if $(\mu_1, \dots, \mu_d)$ lies in the intersection of those halfspaces. Let $\boldsymbol{y}$ be the partial solution contained in the next node that Algorithm 4 will branch on given $\mathcal{T}_t$. We know that Algorithm 4 will choose to branch on variable v_{t+1} at node Q if and only if

$$\mu_1\texttt{score}_1(\mathcal{T}, \boldsymbol{y}, v_{t+1}) + \cdots + \mu_d\texttt{score}_d(\mathcal{T}, \boldsymbol{y}, v_{t+1})$$
$$> \max_{v' \neq v_{t+1}} \left\{ \mu_1\texttt{score}_1(\mathcal{T}, \boldsymbol{y}, v') + \cdots + \mu_d\texttt{score}_d(\mathcal{T}, \boldsymbol{y}, v') \right\}.$$

Since these functions are linear in $(\mu_1, \dots, \mu_d)$, there are at most n^2 halfspaces defining the region where $v_{t+1} = \operatorname{argmax}\{\mu_1\texttt{score}_1(\mathcal{T}, \boldsymbol{y}, v') + \cdots + \mu_d\texttt{score}_d(\mathcal{T}, \boldsymbol{y}, v')\}$. Let $\mathcal{B}'$ be this set of halfspaces. Algorithm 4 using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ builds the partial tree $\mathcal{T}_t$ after t iterations if and only if $(\mu_1, \dots, \mu_d)$ lies in the intersection of the $(t+1)n^2$ halfspaces in the set $\mathcal{B} \cup \mathcal{B}'$. $\qquad\square$

$\hfill\square$

Theorem E.7. *Let* $\texttt{cost}$ *be a tree-constant cost function and let* $\texttt{score}_1, \dots, \texttt{score}_d$ *be* d *arbitrary scoring rules. Suppose we limit Algorithm 4 to producing search trees of size* $\bar{\kappa}$. *Let* $\mathcal{C}$ *be the set of functions* $\left\{ \texttt{cost}(\cdot, \mu\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d) : (\mu_1, \dots, \mu_d) \in [0,1]^d \right\}$. *Then* $\operatorname{Pdim}(\mathcal{C}) = O\left(d\bar{\kappa}\log n + d\log d\right)$.

Proof. Suppose that $\operatorname{Pdim}(\mathcal{C}) = m$ and let $\mathcal{S} = \{\Pi_1, \dots, \Pi_m\}$ be a shatterable set of problem instances. We know there exists a set of targets $r_1, \dots, r_m \in \mathbb{R}$ that witness the shattering of $\mathcal{S}$ by $\mathcal{C}$. This means that for every $\mathcal{S}' \subseteq \mathcal{S}$, there exists a parameter vector $(\mu_{1,\mathcal{S}'}, \dots, \mu_{d,\mathcal{S}'})$ such that if $\Pi_i \in \mathcal{S}'$, then $\texttt{cost}(\Pi_i, \mu_{1,\mathcal{S}'}\texttt{score}_1 + \cdots + \mu_{d,\mathcal{S}'}\texttt{score}_d) \leq r_i$. Otherwise

$$\texttt{cost}(\Pi_i, \mu_{1,\mathcal{S}'}\texttt{score}_1 + \cdots + \mu_{d,\mathcal{S}'}\texttt{score}_d) > r_i.$$

Let $M = \left\{ (\mu_{1,\mathcal{S}'}, \dots, \mu_{d,\mathcal{S}'}) : \mathcal{S}' \subseteq \mathcal{S} \right\}$. We will prove that $|M| = O(dm^d n^{2d(\bar{\kappa}+1)})$, and since $2^m = |M|$, this means that $\operatorname{Pdim}(\mathcal{C}) = m = O\left(d\bar{\kappa}\log n + d\log d\right)$ (see Lemma C.10 in Appendix C).

To prove that $|M| \leq mn^{\bar{\kappa}} + 1$, we rely on Lemma E.4, which tells us that for any problem instance Π, there is a set $\mathcal{H}$ of at most $T \leq n^{2(\bar{\kappa}+1)}$ hyperplanes such that for any connected component R of $[0,1]^d \setminus \mathcal{H}$, the search tree Algorithm 4 builds using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ is invariant across all $(\mu_1, \dots, \mu_d) \in R$. If we merge all T hyperplanes for all samples in $\mathcal{S}$, we are left with a set $\mathcal{H}'$ of $T' \leq mn^{2(\bar{\kappa}+1)}$ hyperplanes where for any connected component R of $[0,1]^d \setminus \mathcal{H}'$ and any $\Pi_i \in \mathcal{S}$, the search tree Algorithm 4 builds using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ given as input Π_i is invariant across all $(\mu_1, \dots, \mu_d) \in R$. Therefore, at most one element of M can

come from each connected component, of which there are $O(d|\mathcal{H}'|^d) = O(dm^d n^{2d(\bar{\kappa}+1)})$. Therefore, $|M| = O(dm^d n^{2d(\bar{\kappa}+1)})$, as claimed. $\qquad\square$

Naturally, if the function $\texttt{cost}$ measures the size of the search tree Algorithm 4 returns capped at some value κ as described in Section 3.1, we obtain the following corollary.

Corollary E.8. *Let $\texttt{cost}$ be a tree-constant cost function, let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules, and let $\kappa \in \mathbb{N}$ be an arbitrary tree size bound. Suppose that for any problem instance Π, $\texttt{cost}(\Pi, \mu\texttt{score}_1 + \cdots + \mu_d \texttt{score}_d)$ equals the minimum of the following two values: 1) The number of nodes in the search tree Algorithm 4 builds using the scoring rule $\mu\texttt{score}_1 + \cdots + \mu_d \texttt{score}_d$ on input Π; and 2) κ. For any distribution $\mathcal{D}$ over problem instances Π with at most n variables, with probability at least $1 - \delta$ over the drawn $\{\Pi^{(1)}, \ldots, \Pi^{(m)}\} \sim \mathcal{D}^m$, for any $\mu \in [0, 1]$,*

$$\left| \mathbb{E}_{\Pi \sim \mathcal{D}}\left[\texttt{cost}(\Pi, \mu\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d)\right] - \frac{1}{m}\sum_{i=1}^{m} \texttt{cost}(\Pi^{(i)}, \mu\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d) \right|$$
$$= O\left(\sqrt{\frac{d\kappa^2(\kappa \log n + \log d)}{m}} + \kappa\sqrt{\frac{\ln(1/\delta)}{m}} \right).$$