Learning to Branch: Generalization Guarantees and Limits of Data-Independent Discretization

MARIA-FLORINA BALCAN, Carnegie Mellon University, United States
TRAVIS DICK, Google, United States
TUOMAS SANDHOLM, Carnegie Mellon University, United States, Strategic Machine, Inc., United States, Strategy Robot, Inc., United States, and Optimized Markets, Inc., United States
ELLEN VITERCIK, Stanford University, United States

Tree search algorithms, such as branch-and-bound, are the most widely used tools for solving combinatorial and non-convex problems. For example, they are the foremost method for solving (mixed) integer programs and constraint satisfaction problems. Tree search algorithms come with a variety of tunable parameters that are notoriously challenging to tune by hand. A growing body of research has demonstrated the power of using a data-driven approach to automatically optimize the parameters of tree search algorithms. These techniques use a *training set* of integer programs sampled from an application-specific instance distribution to find a parameter setting that has strong average performance over the training set. However, with too few samples, a parameter setting may have strong average performance on the training set but poor expected performance on future integer programs from the same application. Our main contribution is to provide the first *sample complexity guarantees* for tree search parameter tuning. These guarantees bound the number of samples sufficient to ensure that the average performance of tree search over the samples nearly matches its future expected performance on the unknown instance distribution. In particular, the parameters we analyze weight *scoring rules* used for variable selection. Proving these guarantees is challenging because tree size is a volatile function of these parameters: we prove that, for any discretization (uniform or not) of the parameter space, there exists a distribution over integer programs such that every parameter setting in the discretization results in a tree with exponential expected size, yet there exist parameter settings between the discretized points that result in trees of constant size. In addition, we provide data-dependent guarantees that depend on the volatility of these tree-size functions: our guarantees improve if the tree-size functions can be well approximated by

This work was supported in part by the National Science Foundation under grants CCF-1422910, CCF-1535967, CCF-1733556, CCF-1910321, IIS-1617590, IIS-1618714, IIS-1718457, SES-1919453, RI-2312342, and RI-1901403; a Microsoft Research Faculty Fellowship; an Amazon Research Award; an AWS Machine Learning Research Award; a Bloomberg Research Grant; a NSF Graduate Research Fellowship; a fellowship from Carnegie Mellon University's Center for Machine Learning and Health; an IBM PhD Fellowship; the ARO under awards W911NF2210266, W911NF-17-1-0082, and W911NF2010081; ONR award N00014-23-1-2876; the Defense Advanced Research Projects Agency under cooperative agreement HR0011202000; the Vannevar Bush Faculty Fellowship.
A short early version of this article appeared in 2018 at the *International Conference on Machine Learning (ICML)* [Balcan et al. 2018a] under the title "Learning to Branch." In addition, the results in Sections 3.3 and 4.2 appeared in 2020 at ICML [Balcan et al. 2020d].
Authors' addresses: M.-F. Balcan, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213; e-mail: ninamf@cs.cmu.edu; T. Dick, Google, 111 8th Ave, New York, NY, 10011; e-mail: tdick@google.com; T. Sandholm, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, Strategic Machine, Inc., Pittsburgh, PA 15213, Strategy Robot, Inc., Pittsburgh, PA 15213, and Optimized Markets, Inc., Pittsburgh, PA, 15213; e-mail: sandholm@cs.cmu.edu; E. Vitercik, Stanford University, 475 Via Ortega, Stanford, California, 94305; e-mail: vitercik@stanford.edu.

ACM 0004-5411/2024/04-ART13
https://doi.org/10.1145/3637840

simpler functions. Finally, via experiments, we illustrate that learning an optimal weighting of scoring rules reduces tree size.

CCS Concepts: • **Theory of computation → Integer programming**; **Branch-and-bound**; **Sample complexity and generalization bounds;**

Additional Key Words and Phrases: Parameter tuning, integer programming, tree search, branch-and-bound, learning theory, constraint satisfaction problems

ACM Reference Format:
Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. 2024. Learning to Branch: Generalization Guarantees and Limits of Data-Independent Discretization. *J. ACM* 71, 2, Article 13 (April 2024), 73 pages. https://doi.org/10.1145/3637840

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, time-consuming, and error-prone. The challenges associated with parameter tuning are especially evident for tree search algorithms, which are the most widely used tools for solving combinatorial and nonconvex problems. For example, CPLEX is a popular commercial software that uses branch-and-bound (B&B) [Land and Doig 1960], a tree search algorithm, to solve integer programs. CPLEX offers 172 parameters for the user to tune, detailed by a 170-page reference manual [IBM ILOG Inc 2017], in addition to many internal parameters that are not readily available for the user to tune. CPLEX's performance is sensitive to small changes in these parameters, and particular parameter settings have varying performance across application domains. Properly tuning these parameters has been shown to lead to significant speedups.[1]

A growing body of research has studied how to use a data-driven approach to automate tree search parameter tuning [e.g., Alvarez et al. 2017; Ansótegui et al. 2009; Balcan et al. 2020c, d; Di Liberto et al. 2016; Gasse et al. 2019; Gomes and Selman 2001; He et al. 2014; Horvitz et al. 2001; Hutter et al. 2010, 2011, 2009, 2014; Kadioglu et al. 2010; Khalil et al. 2016, 2017; Kleinberg et al. 2017, 2019; Kruber et al. 2017; Lagoudakis and Littman 2001; Leyton-Brown et al. 2009; Liang et al. 2016; Lobjois and Lemaître 1998; Sabharwal et al. 2017; Sandholm 2013; Song et al. 2020, 2018; Tang et al. 2020; Weisz et al. 2018, 2019; Xia and Yap 2018; Xu et al. 2011]. Many of these techniques use a *training set* of typical problem instances from the particular application domain to learn parameter settings that have strong average empirical performance over the training set. This training set is assumed to be sampled from an unknown, application-specific distribution, such as a distribution over winner determination problem instances from combinatorial sourcing auctions [Sandholm 2013]. These applied approaches to parameter tuning have led to breakthroughs in discrete optimization.

In this article, we provide the first *generalization guarantees* for tree search parameter tuning. With too few samples, a parameter setting may have strong performance on average over the training set but poor future performance on problems from the same application domain that are not already in the training set. Therefore, it is crucial to understand how many samples are sufficient to ensure that this type of overfitting does not occur. The guarantees we provide bound, for any parameter setting, the difference between the size of the search tree the algorithm builds on average over the training set and the expected size of the tree the algorithm will build on future,

[1] ParamILS, for example, led to speedup factors ranging from 2 to 23 [Hutter et al. 2009].

unseen instances. Therefore, we can be confident that no matter how we tune the parameters—optimally or suboptimally, manually or automatically—if a parameter setting leads to small trees on average over the training set, it will also lead to small trees on problem instances from the same application domain that are not contained in the training set.

1.1 Our Contributions

We provide guarantees for learning tree search *variable-selection policies.* A tree search algorithm systematically partitions the input problem's feasible region 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 parent node. This partition is typically defined by constraining a carefully selected variable. For example, when solving a binary program, one set of the partition might be defined by adding the constraint $x[i] = 0$ for some variable $x[i]$, in which case, the other set of the partition would be defined by adding the constraint $x[i] = 1$. 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 impact on the size of the tree and, therefore, the speed of the overall algorithm. For the typical sound tree search algorithms, any variable-selection policy preserves soundness.

There is no known optimal variable-selection 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 [Liberatore 2000]. Most existing variable-selection policies assign a real-valued score to each variable and select the variable with the highest score. Researchers have proposed a variety of scoring rules [Achterberg 2009; Beale 1979; Bénichou et al. 1971; Gauthier and Ribière 1977; Gilpin and Sandholm 2011; Linderoth and Savelsbergh 1999], none of which is universally optimal.

As our main contribution, we provide generalization bounds for learning how to weight any set of scoring rules. In other words, given any d scoring rules $\texttt{score}_1, \ldots, \texttt{score}_d$, we provide generalization bounds for learning weighted scoring rules $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$, where $\mu_1, \ldots, \mu_d \in [0, 1]$ are parameters that can be fine-tuned to the application domain at hand. The tree search algorithm then branches on the variable that maximizes this weighted score. Our results also generalize to learning weighted products $\texttt{score}_1^{\mu_1} \cdots \texttt{score}_d^{\mu_d}$. As we describe in Section 3.1, prior research has investigated the empirical performance of tree search using various scoring rules and parameter settings [Achterberg 2009; Beale 1979; Bénichou et al. 1971; Gauthier and Ribière 1977; Linderoth and Savelsbergh 1999]. We provide the first generalization guarantees for tuning these parameters.

In order to prove these guarantees, we analyze how these parameters impact the search tree that the algorithm builds. This is challenging because, as we prove in Section 3.5, a tiny shift in parameters can trigger an exponential jump in tree size. Specifically, for any discretization of the parameter space, we identify an infinite family of distributions over *mixed integer linear programming (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 significant change in the algorithm's behavior. This is unlike sample complexity guarantees that are typically found in machine learning theory, where there is typically a straightforward connection between a function's parameters and its value on any input. Since the tree size of a search algorithm is a complicated and discontinuous function of its parameters, we must carefully analyze the way in which the parameters influence each step of the procedure in order to derive our generalization guarantees.

Indeed, we show that there is structure governing this volatile relationship between the parameters and the search tree size. In particular, we prove that for any integer program, hyperplanes partition the parameter space $[0, 1]^d$ into a finite number of regions such that in any one region, tree search builds the same tree no matter what parameter setting it uses from that region. In other words, tree size is a piecewise-constant function of the parameters, defined by a finite number of pieces with linear boundaries. At a high level, this means that although there are an infinite number of parameter settings, there are only a finite number of parameter settings that lead the algorithm to behave differently. This analysis allows us to prove our main generalization bound, which grows logarithmically in the number of pieces that these piecewise-constant functions have.

Moreover, we show that, although the piecewise-constant tree size functions may have many little pieces, they can often be well-approximated by piecewise-constant functions with only a few pieces. We use this observation to prove data-dependent guarantees that improve based on the simplicity of these approximating functions. Moreover, we show that this observation can be used to provide data-dependent guarantees beyond the context of integer programming, applying whenever an algorithm's performance as a function of its parameters can be approximated by a "simple" function. We show that if this approximation holds under the L_∞- norm, we can provide strong generalization bounds. On the flip side, if the approximation holds only under the L_p-norm for $p < \infty$, it is not possible to provide meaningful sample complexity bounds in the worst case.

In our experiments section, we show that, on many datasets based on real-world NP-hard problems, different parameter settings can result in search trees of different sizes. Using an optimal parameter for one distribution on problems from a different distribution can lead to an increase in tree size. We additionally show via experiments that our data-dependent generalization bound based on simple approximations of the tree size function can be significantly smaller than the worst-case bound.

1.2 Related Research

Algorithm parameter tuning has been studied for decades, beginning with a seminal paper by Rice [1976]. This research area studies how an algorithm's parameters (both continuous and discrete) impact its performance. In this section, we highlight related research on this topic in roughly chronological order. Much of this research has studied how to algorithmically find high-performing parameter settings. Later research has studied theoretical problems on topics such as sample complexity, as in the present article.

Early research on using machine learning and data-driven approaches for algorithm design and selection dates back to the late 1990s and early 2000s. This includes research on *algorithm portfolios and selection* (that is, using the input problem instance to choose which algorithm to use from some finite set of possibilities) [Gomes and Selman 2001; Lobjois and Lemaître 1998]; *parameter control*, which addresses how to adapt the parameters of an algorithm online while solving an input problem [Lagoudakis and Littman 2001]; and runtime prediction [Horvitz et al. 2001].

Theoretical Research on Algorithm Scheduling. In a series of theoretical papers, Sayag et al. [2006]; Streeter et al. [2007], and Streeter and Golovin [2009] studied a learning-theoretic model for algorithm scheduling, which is a similar but distinct problem from algorithm parameter tuning. In their setup, there is a finite set of blackbox heuristic algorithms capable of computing a correct solution to a given problem, but with different costs. The user can run multiple algorithms until one terminates with the correct solution. Given a training set of problem instances, these papers show how to learn a *schedule* with nearly minimum expected cost. In contrast, in this article, our goal is to provide guarantees when there are infinitely-many possible parameter settings using the structure of the branch-and-bound algorithm itself.

Portfolio-Based Algorithm Selection. A high-level distinction among automated approaches to parameter tuning is whether one is trying to find one parameter setting that performs well for the entire application-specific distribution over problem instances, or one is trying to select different parameter settings for different problem instances based on problem instance features. In this paper, we focus on the former problem. The latter approach—known as portfolio-based algorithm selection—has also been used to great success for a variety of applications [Kadioglu et al. 2010; Leyton-Brown et al. 2009; Sandholm 2013; Xu et al. 2008, 2010, 2011].

Early General-Purpose Parameter Tuning Procedures. Moving on to early applied approaches to algorithm parameter tuning, ParamILS is a general-purpose procedure proposed by Hutter et al. [2010, 2009], which uses iterated local search to find high-performing parameter settings, employing a careful adaptive capping procedure to make sure time is not wasted evaluating bad parameter settings. Around this time, Ansótegui et al. [2009] proposed an algorithm called *GGA* which relies on genetic algorithms to find high-performing parameter settings.

ParamILS and GGA are *model-free* approaches, whereas the next generation of algorithm parameter tuning procedures, such as SMAC [Hutter et al. 2011], are *model-based* approaches. SMAC iterates between fitting models that predict algorithmic performance and using those models to select promising parameter settings to further investigate.

Fielded Applications. Automated algorithm configuration has had success in industry already. From 2002 to 2010, it was used to configure integer programming approaches for winner determination in $60 billion of combinatorial auctions [Sandholm 2013]. They used automated configuration approaches both to select among modeling choices and choices within the integer programming algorithms themselves.

Several major commercial integer programming solvers now ship with algorithm configuration tools that allow customers to automatically tune the solver's parameters to their specific problems, thereby bringing automated algorithm configuration to potentially all applications of integer programming. For example, CPLEX introduced an automated parameter tuning tool in 2007 [IBM ILOG Inc 2007].

As another example, algorithm configuration and selection have been an integral part of the US Federal Communications Commission's spectrum reallocation auctions [Leyton-Brown et al. 2017]. These auctions require solving large satisfiability problems to check the feasibility of an allocation.

Node Selection Policies. Several articles have studied machine learning approaches to the specific task of *node selection* in branch-and-bound, including Sabharwal et al. [2017] and He et al. [2014], whereas we study variable selection policies.

Piecewise-Structure in the Context of Runtime Prediction. Hutter et al. [2014] studied how to predict the runtime of SAT, MIP, and TSP solvers. Among many other results, they found that solver runtime as a function of parameter settings could be predicted quite well by many different models, including random forests. In effect, these random forests define a partition of the parameter space into regions where the predicted performance is fixed. The success of random forests for this task could be seen an empirical parallel of the theoretical results we prove in this article: true performance is a piecewise-constant function of the parameters.

Applied Approaches to Variable Selection. As in the present article, Khalil et al. [2016] and Alvarez et al. [2017] studied variable-selection policies. Both of these articles provide strategies for learning variable-selection strategies that mimic the behavior of the classic branching strategy known as

strong branching while running faster than strong branching. Neither of these articles comes with theoretical guarantees, unlike the present article.

Additional Applied Approaches to B&B Tuning beyond Variable and Node Selection. A variety of research has studied machine learning approaches to improving additional aspects of branch-and-bound beyond variable and node selection [e.g., Bengio et al. 2020a; Bonami et al. 2018; Di Liberto et al. 2016; Gasse et al. 2019; Khalil et al. 2017; Kruber et al. 2017; Song et al. 2020; Tang et al. 2020]. Many recent advances are covered in the survey by Bengio et al. [2020b].

Theoretical Analyses of Variable Selection. From a theoretical perspective, Le Bodic and Nemhauser [2017] presented a variable-selection model based on an abstraction to a simpler setting in which it is possible to analytically evaluate the dual bound improvement of choosing a given variable. Based on that model, they presented a new variable selection policy that has strong performance on many MIPLIB instances.

Learning-Theoretic Guarantees for Data-Driven Algorithm Design. A recent, active line of research has studied learning-theoretic questions related to data-driven algorithm design. Much of this work has been related to algorithm parameter tuning [e.g., Ahmadi et al. 2022; Balcan et al. 2018a, 2022, 2017, 2020a; Bartlett et al. 2022; Blum et al. 2021; Gupta and Roughgarden 2017; Sakaue and Oki 2022] and was synthesized in a book chapter by Balcan [2020]. In these articles, an application domain is modeled as a distribution over problem instances, and the goal is to tune an algorithm's parameters using samples from that distribution.

Gupta and Roughgarden [2017] initiated this line of research by proving generalization guarantees for tuning the parameters of various greedy and sorting algorithms, and for tuning the step size of gradient descent. Balcan et al. [2017] then provided guarantees for tuning the parameters of clustering algorithms. They analyze families of polynomial-time clustering algorithms that are designed to approximately optimize various objective functions—such as k-medians, k-means, and k-center—using a linkage routine. In this context, they prove that for any data-independent discretization of the parameter space, there exists a family of clustering instances such that any parameter setting in the discretization will output a clustering that is significantly worse than the optimal parameter setting. This result parallels our result for tree search but requires a completely different proof since the algorithms are different.

Balcan et al. [2017] also analyzed parameterized approximation algorithms for integer quadratic programming, which are based on semi-definite programming and randomized rounding. The approximation algorithm's parameter controls the randomized rounding technique and is unrelated to tree search.

Several articles have also studied online algorithm parameter tuning, where there is no distribution over problem instances, but rather instances arrive one-by-one [Balcan et al. 2018b, 2020b, 2021b; Balcan and Sharma 2021; Cohen-Addad and Kanade 2017; Gupta and Roughgarden 2017; Sharma et al. 2020]. Before the arrival of each instance, the learning algorithm commits to a parameter setting. The goal is to minimize *regret*, which is the cumulative difference between the performance of the best choice of a parameter setting in hindsight, and the performance of the learner's selected parameter settings. Minimizing regret in this context is impossible in the worst case [Balcan et al. 2018b; Gupta and Roughgarden 2017], so these articles' guarantees depend on beyond-worst-case assumptions on the input problem instances. In contrast, we study the batch learning model and we do not make any assumptions about the distribution over problem instances.

Parameter Tuning via 'Structured Procrastination". A line of research initiated by Kleinberg et al. [2017] has provided learning-based parameter tuning procedures with provable guarantees. Their

algorithms return nearly optimal parameter settings from within a finite set. In that finite setting, the primary challenge is bounding the training time. Those algorithms can also be used when the parameter space is infinite by first sampling $\tilde{\Omega}(1/\gamma)$ parameter settings for some $\gamma \in (0, 1)$ and then running the algorithm over this finite set. Kleinberg et al. [2017] guaranteed that the output parameter setting will be within the top γ-quantile.

The article by Kleinberg et al. [2017] described above was subsequently built upon by Weisz et al. [2018, 2019] and Kleinberg et al. [2019], providing improved algorithms with faster training times. Balcan et al. [2020c] aimed to marry the research presented in this article with the research by Kleinberg et al. [2017, 2019] and Weisz et al. [2018, 2019] described above. Balcan et al. [2020c] presented an algorithm that identifies a finite set of promising parameters within an infinite set given sample access to a distribution over problem instances. They proved that this set contains a nearly optimal parameter vector with high probability. The set can serve as the input to a parameter-tuning algorithm for finite parameter spaces [Kleinberg et al. 2017, 2019; Weisz et al. 2018, 2019], which Balcan et al. [2020c] proved will then return a nearly optimal parameter from the infinite set.

Algorithms Aided by Machine-Learned Advice. A related line of research has designed algorithms that are aided by "machine learned advice" [e.g., Hsu et al. 2019; Lykouris and Vassilvitskii 2018; Mitzenmacher 2018; Purohit et al. 2018]. In essence, these algorithms use machine learning to make predictions about structural aspects of the input. If the prediction is accurate, the algorithm's performance (for example, its error or runtime) is superior to the best-known worst-case algorithm, and if the prediction is incorrect, the algorithm performs as well as that worst-case algorithm.

Guarantees for Portfolio-Based Algorithm Selection. Balcan et al. [2021a] provided sample complexity guarantees for portfolio-based algorithm selection. Portfolio-based algorithm selection, which has seen tremendous success in practice [e.g., Xu et al. 2010, 2011], first uses the training set to compile a portfolio of algorithm parameter settings. At runtime, a machine learning model is used to predict which parameter setting in the portfolio will have the best performance. The guarantees by Balcan et al. [2021a] apply whenever the algorithm's performance on any input is a piecewise constant function of its parameters, as we prove is the case for tree search in this article.

Heuristic Selection. Speck et al. [2021] studied how to dynamically select heuristics within heuristic search. They provided theoretical guarantees proving that this approach can lead to an exponential improvement in the number of states explored compared to (1) only using features of the instance to choose a heuristic and (2) only using the timestep to choose a heuristic (as in a round-robin approach, for example).

Branch-and-Bound on Random Instances. To better understand the strong performance of branch-and-bound in practice, several articles have studied the algorithm's performance on random integer programs [Dey et al. 2021a, b]. Dey et al. [2021a] proved that in polynomial time, branch-and-bound solves integer programs with uniformly random entries and a constant number of constraints. Meanwhile, Dey et al. [2021b] provided lower bounds showing that there exist packing instances, set covering instances, and Traveling Salesman Problem instances where branch-and-bound builds a tree of exponential size, even when it branches on general disjunctions. This remains true even when the instances are perturbed with random noise (*a la smoothed analysis* [Spielman and Teng 2004]).

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**

(MILPs)-and describe how tree search finds optimal solutions to problems from this family. Tree search for solving MILPs is called *branch-and-bound*. For ease of exposition, we present our main results in Section 3 in terms of branch-and-bound. Later on, in Section 3.4 and Appendix D, we describe how tree search can be used to solve constraint satisfaction problems and generalize our results to that setting.

2.1 Mixed Integer Linear Programs

We study 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 integers. 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 article, 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 $v_i(b) \in \mathbb{R}$ 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 (for all items $j \in \{1, \ldots, m\}$, $\sum_{i=1}^{n} \sum_{b \in B_i, j \ni b} x_{i,b} \leq 1$) and if each bidder receives at most one bundle (for all buyers $i \in \{1, \ldots, n\}$,$\sum_{b \in B_i} x_{i,b} \leq 1$). Therefore, the MILP is:

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

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 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. It finds the solution $\check{\boldsymbol{x}}_Q$ to the LP relaxation of 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^+ (respectively, Q_i^-) be the MILP Q with the additional constraint that $x[i] \geq \lceil \check{x}_Q[i] \rceil$ (respectively, $x[i] \leq \lfloor \check{x}_Q[i] \rfloor$). B&B sets the right (respectively, left) child of Q in $\mathcal{T}$ to be a node containing the MILP Q_i^+ (respectively, Q_i^-). B&B then tries to "fathom" these leaves: the leaf containing Q_i^+ (respectively, Q_i^-) is *fathomed* if:

(1) The optimal solution to the LP relaxation of Q_i^+ (respectively, Q_i^-) satisfies the constraints of the original MILP Q';
(2) The relaxation of Q_i^+ (respectively, Q_i^-) is infeasible, so Q_i^+ (respectively, Q_i^-) must be infeasible as well; or
(3) The objective value of the LP relaxation of Q_i^+ (respectively, Q_i^-) is smaller than the objective value of the best known feasible solution, so the optimal solution to Q_i^+ (respectively, 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.

ALGORITHM 1: Branch and bound

Input: A MILP Q'.
Let $\mathcal{T}$ be a tree that consists of a single node containing the MILP Q'.
Let $c^* = -\infty$ be the objective value of the best-known feasible solution.
while *there remains an unfathomed leaf in* $\mathcal{T}$ **do**
  Use a *node selection policy* to select a leaf of the tree $\mathcal{T}$, which corresponds to a MILP Q.
  Use a *variable-selection policy* to choose a variable $x[i]$ of the MILP Q to branch on.
  Let $\check{\boldsymbol{x}}_Q$ be the solution to the LP relaxation of Q.
  Let Q_i^+ (respectively, Q_i^-) be the MILP Q with the extra constraint that $x[i] \geq \lceil \check{x}_Q[i] \rceil$ (respectively, $x[i] \leq \lfloor \check{x}_Q[i] \rfloor$).
  Set the right (respectively, left) child of Q in $\mathcal{T}$ to be a node containing the MILP Q_i^+ (respectively, Q_i^-).
  for $\tilde{Q} \in \{Q_i^+, Q_i^-\}$ **do**
    if *the LP relaxation of* $\tilde{Q}$ *is feasible* **then**
      Let $\check{\boldsymbol{x}}_{\tilde{Q}}$ be an optimal solution to the LP and let $\check{c}_{\tilde{Q}}$ be its objective value.
      if *the vector* $\check{\boldsymbol{x}}_{\tilde{Q}}$ *satisfies the constraints of the original MILP* Q' **then**
        Fathom the leaf containing $\tilde{Q}$.
        if $c^* < \check{c}_{\tilde{Q}}$ **then**
          Set $c^* = \check{c}_{\tilde{Q}}$.
        end
      else if $\check{\boldsymbol{x}}_{\tilde{Q}}$ *is no better than the best known feasible solution, i.e.,* $c^* \geq \check{c}_{\tilde{Q}}$ **then**
        Fathom the leaf containing $\tilde{Q}$.
    else
      Fathom the leaf containing $\tilde{Q}$.
    end
  end
end

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 MILP [Kolesar 1967]:

$$\begin{aligned} \text{maximize} \quad & 40x[1] + 60x[2] + 10x[3] + 10x[4] + 3x[5] + 20x[6] + 60x[7] \\ \text{subject to} \quad & 40x[1] + 50x[2] + 30x[3] + 10x[4] + 10x[5] + 40x[6] + 30x[7] \leq 100 \\ & x[1], \ldots, x[7] \in \{0, 1\}. \end{aligned} \tag{1}$$

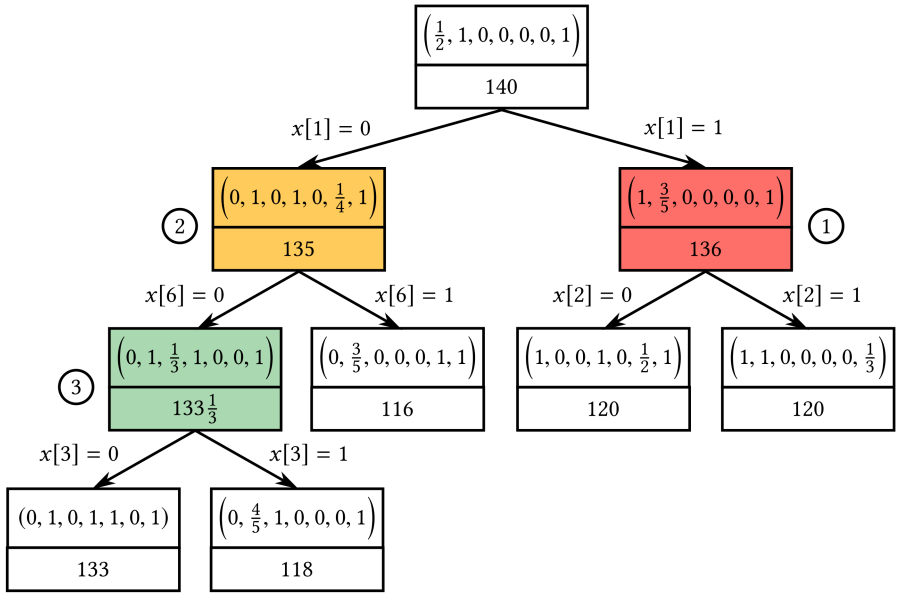


Fig. 1. Illustration of Example 2.2.

Each rectangle denotes a node in the B&B tree. Given a node Q, the top portion of its rectangle displays the optimal solution $\check{\boldsymbol{x}}_Q$ 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 $\check{c}_Q$ of the optimal solution to this LP relaxation, i.e., $\check{c}_Q = (40, 60, 10, 10, 3, 20, 60) \cdot \check{\boldsymbol{x}}_Q$. 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 $\check{x}_Q[i]$ is closest to $\frac{1}{2}$, i.e., $i = \text{argmax}\{\min\{1 - \check{x}_Q[i], \check{x}_Q[i]\}\}$.

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 leaves have smaller objective values, the algorithm cannot find a better solution by exploring those leaves. Therefore, the algorithm fathoms all of the leaves 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 2.3 (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 $\text{argmax}_{x[j] \in N_{\mathcal{T},Q}} \{\texttt{score}(\mathcal{T}, Q, j)\}$ 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 $\check{\boldsymbol{x}}_Q = (\check{x}_Q[1], \ldots, \check{x}_Q[n])$. We also use the notation $\check{c}_Q$ to denote the objective value of the optimal solution to the LP relaxation of Q, i.e., $\check{c}_Q = \boldsymbol{c}^\top \check{\boldsymbol{x}}_Q$. Finally, we use the notation Q_i^+ (respectively, Q_i^-) to denote the MILP Q with the additional constraint that $x[i] \geq \lceil \check{x}_Q[i] \rceil$ (respectively, $x[i] \leq \lfloor \check{x}_Q[i] \rfloor$). If Q_i^+ (respectively, Q_i^-) is infeasible, then we set $\check{c}_Q - \check{c}_{Q_i^+}$ (respectively, $\check{c}_Q - \check{c}_{Q_i^-}$) to be some large number greater than $||\boldsymbol{c}||_1$.

Most Fractional. In this case, $\texttt{score}(\mathcal{T}, Q, i) = \min\{\lceil \check{x}_Q[i] \rceil - \check{x}_Q[i], \check{x}_Q[i] - \lfloor \check{x}_Q[i] \rfloor\}$. The variable that maximizes $\texttt{score}(\mathcal{T}, Q, i)$ is the "most fractional" variable, since it is the variable such that $\check{x}_Q[i]$ is farthest from $\lceil \check{x}_Q[i] \rceil$ or $\lfloor \check{x}_Q[i] \rfloor$.

Linear Scoring Rule [Linderoth and Savelsbergh 1999]. In this case, $\texttt{score}(\mathcal{T}, Q, i) = (1-\mu) \cdot \min\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\} + \mu \cdot \max\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}$ 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 $\max\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}$, which corresponds to $\mu = 1$, and a pessimistic approach would choose the variable that maximizes $\min\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}$, which corresponds to $\mu = 0$.

Product Scoring Rule [Achterberg 2009]. In this case,

$$\texttt{score}(\mathcal{T}, Q, i) = \max\{\check{c}_Q - \check{c}_{Q_i^-}, \gamma\} \cdot \max\{\check{c}_Q - \check{c}_{Q_i^+}, \gamma\},$$

where $\gamma = 10^{-6}$. Comparing $\check{c}_Q - \check{c}_{Q_i^-}$ and $\check{c}_Q - \check{c}_{Q_i^+}$ to γ allows the algorithm to compare two variables even if $\check{c}_Q - \check{c}_{Q_i^-} = 0$ or $\check{c}_Q - \check{c}_{Q_i^+} = 0$. After all, suppose the scoring rule simply calculated the product $(\check{c}_Q - \check{c}_{Q_i^-}) \cdot (\check{c}_Q - \check{c}_{Q_i^+})$ without comparing to γ. If $\check{c}_Q - \check{c}_{Q_i^-} = 0$, then the score equals 0, canceling out the value of $\check{c}_Q - \check{c}_{Q_i^+}$ and thus losing the information encoded by this difference.

Entropic Lookahead Scoring Rule [Gilpin and Sandholm 2011]. Let

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

and for any variable $i \in [n]$ and MIP Q, let $f_{Q,i} = \check{x}_Q[i] - \lfloor \check{x}_Q[i] \rfloor$ be the fractional part of $\check{x}_Q[i]$. Set

$$\texttt{score}(\mathcal{T}, Q, i) = -\sum_{j=1}^{n} \left(1 - f_{Q,i}\right) \cdot e\left(f_{Q_i^-, j}\right) + f_{Q,i} \cdot e\left(f_{Q_i^+, j}\right).$$

Alternative Definitions of the Linear and Product Scoring Rules. In practice, it is often too slow to compute the differences $\check{c}_Q - \check{c}_{Q_i^-}$ and $\check{c}_Q - \check{c}_{Q_i^+}$ 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 Q_i^+, starting at $\check{\boldsymbol{x}}_Q$ 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 $\texttt{score}(\mathcal{T}, Q, i) = (1-\mu) \cdot \min\{\check{c}_Q - \tilde{c}_{Q_i^+}, \check{c}_Q - \tilde{c}_{Q_i^-}\} + \mu \cdot \max\{\check{c}_Q - \tilde{c}_{Q_i^+}, \check{c}_Q - \tilde{c}_{Q_i^-}\}$ and the product scoring rule to be $\texttt{score}(\mathcal{T}, Q, i) = \max\{\check{c}_Q - \tilde{c}_{Q_i^-}, \gamma\} \cdot \max\{\check{c}_Q - \tilde{c}_{Q_i^+}, \gamma\}$. Other popular alternatives to computing $\check{c}_{Q_i^-}$ and $\check{c}_{Q_i^+}$ that fit within our framework are *pseudocost branching* [Bénichou et al. 1971; Gauthier and Ribière 1977; Linderoth and Savelsbergh 1999] and *reliability branching* [Achterberg et al. 2005]. Appendix A reviews these two branching rules in detail.

3 GUARANTEES FOR DATA-DRIVEN LEARNING TO BRANCH

In this section, we begin with our problem statement, formally defining the notion of sample complexity in the context of branch-and-bound parameter tuning. Next, in Section 3.2, we prove our main generalization guarantees for branch-and-bound. The main challenge we face in proving our generalization guarantees is the volatility of the branch-and-bound search tree as a function of the algorithm's parameters. In Section 3.5, we illustrate this volatility by proving that for any discretization of the parameter space, there exists a distribution such that every parameter setting in the discretization leads to exponential expected tree size, yet there exist parameter settings between the discretized points that lead to constant tree size. Our guarantees can be extended to general tree search for constraint satisfaction problems, as we highlight later in Section 3.4. Throughout the remainder of this article, 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 high-level goal is to provide generalization guarantees for learning convex combinations $\sum_{i=1}^{d} \mu_i \texttt{score}_i$ of the scoring rules. More formally, let $\texttt{cost}_{\boldsymbol{\mu}}$ be an abstract cost function that takes as input a problem instance Q and returns some measure of the quality of B&B using $\sum_{i=1}^{d} \mu_i \texttt{score}_i$ on input Q. For example, $\texttt{cost}_{\boldsymbol{\mu}}$ might be the number of nodes produced by running B&B using $\sum_{i=1}^{d} \mu_i \texttt{score}_i$ on input Q. A generalization guarantee is a function $\epsilon : \mathbb{N} \times \delta \to \mathbb{R}$ such that for any number of samples m and $\delta \in (0, 1)$, with probability $1 - \delta$ over the draw of m samples $Q_1, \ldots, Q_m \sim \mathcal{D}$, for any convex combination $\sum_{i=1}^{d} \mu_i \texttt{score}_i$, the average cost over the samples is $\epsilon(m, \delta)$-close to its expected cost:

$$\left| \frac{1}{m} \sum_{i=1}^{m} \texttt{cost}_{\boldsymbol{\mu}}(Q_i) - \mathbb{E}_{Q \sim \mathcal{D}}[\texttt{cost}_{\boldsymbol{\mu}}(Q)] \right| \leq \epsilon(m, \delta). \tag{2}$$

This type of bound is useful for any algorithm that tunes the parameters using the samples since it guarantees that the average cost of the output parameter setting is indicative of its expected cost. The bound also guarantees that if the learning algorithm finds a parameter setting that minimizes average cost over the samples, that parameter setting will nearly minimize expected cost as well. More formally, suppose that $\sum_{i=1}^{d} \hat{\mu}_i \texttt{score}_i$ minimizes the average cost over the samples: $(\hat{\mu}_1, \ldots, \hat{\mu}_d) = \operatorname{argmin} \sum_{j=1}^{m} \texttt{cost}_{\boldsymbol{\mu}}(Q_j)$. Also, suppose Equation (2) holds for all convex combinations $\sum_{i=1}^{d} \mu_i \texttt{score}_i$. Then $\sum_{i=1}^{d} \hat{\mu}_i \texttt{score}_i$ is nearly optimal overall:

$$\mathbb{E}_{Q \sim \mathcal{D}}[\texttt{cost}_{\hat{\boldsymbol{\mu}}}(Q)] \leq \min_{\mu_1, \ldots, \mu_d \in [0,1]} \mathbb{E}_{Q \sim \mathcal{D}}[\texttt{cost}_{\boldsymbol{\mu}}(Q)] + 2\epsilon(m, \delta).$$

We also generalize our analysis to cover non-linear combinations of the form

$$\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}. \tag{3}$$

To provide generalization bounds, we will use structure exhibited by the *dual* functions $\texttt{cost}^*_Q(\boldsymbol{\mu})$, where $\texttt{cost}^*_Q(\boldsymbol{\mu}) = \texttt{cost}_{\boldsymbol{\mu}}(Q)$. When analyzing the dual functions, we fix a MILP Q and analyze B&B's cost as a function of the parameter vector $\boldsymbol{\mu}$.

Following prior work (e.g., Hutter et al. [2009] and Kleinberg et al. [2017]), we assume that there is some cap κ on the size of the tree that B&B builds. For example, 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 variables for some $n \in \mathbb{N}$ and that the range of $\texttt{cost}_{\mu}$ is bounded in $[0, H]$ for some H.

Our results hold for cost functions that are *tree-constant*, which means that for any problem instance Q, so long as the scoring rules $\sum_{i=1}^{d} \mu_i \texttt{score}_i$ and $\sum_{i=1}^{d} \mu_i' \texttt{score}_i$ result in the same search tree, $\texttt{cost}_{\mu}(Q) = \texttt{cost}_{\mu'}(Q)$. For example, the size of the search tree is tree-constant.

Prior Research on Combining Scoring Rules. For decades, convex combinations of scoring rules $\sum_{i=1}^{d} \mu_i \texttt{score}_i$ have been used in B&B. For example, suppose

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

so by employing the scoring rule $(1-\mu)\texttt{score}_1 + \mu\texttt{score}_2$, we balance an "optimistic" and "pessimistic" approach to branching. Over time, researchers have proposed many different candidates for the parameter setting μ. Gauthier and Ribière [1977] proposed setting $\mu = 1/2$. Bénichou et al. [1971] and Beale [1979] suggested setting $\mu = 1$. Linderoth and Savelsbergh [1999] found that $\mu = 2/3$ performs well. Achterberg [2009] found that experimentally, $\mu = 1/6$ performed best when comparing among $\mu \in \{0, 1/6, 1/3, 1/2, 1\}$. In our experiments, we find that the best choice of a parameter setting depends on the specific application domain at hand; no one parameter setting is optimal overall. In Section 3.5, we also prove that no one parameter setting is always optimal, and in fact, we prove something significantly stronger. Namely, we prove that for *any* discretization of the parameter space (such as $\{0, 1/6, 1/3, 1/2, 1\}$), there exists a distribution such that every parameter setting in the discretization leads B&B to build a tree with exponential size in expectation. Yet, there exist parameter settings between the discretized points that lead B&B to build a tree of constant size.

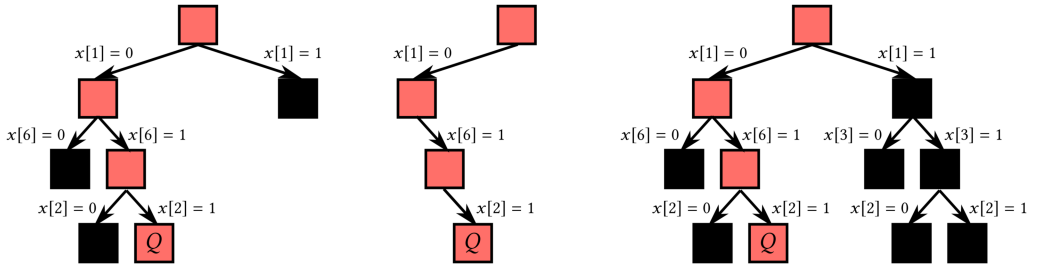
Non-linear combinations of scoring rules are also popular in B&B. SCIP [Achterberg 2009], the best open-source solver, uses the product scoring rule $\texttt{score} = \texttt{score}_1\texttt{score}_2$ where $\texttt{score}_1(\mathcal{T}, Q, i) = \max\{\check{c}_Q - \check{c}_{Q_i^-}, 10^{-6}\}$ and $\texttt{score}_2(\mathcal{T}, Q, i) = \max\{\check{c}_Q - \check{c}_{Q_i^+}, 10^{-6}\}$. This is equivalent to Equation (3) with $\mu_1 = \mu_2 = \frac{1}{2}$. In our experiments, we show that alternative parameter choices can outperform this baseline.

3.2 Generalization Guarantees

We now provide generalization bounds for learning combinations of scoring rules. These results bound, for any convex combination $\texttt{score} = \sum_{i=1}^{d} \mu_i \texttt{score}_i$ of scoring rules, the difference between the empirical cost of using score and its expected cost. Therefore, the algorithm designer can tune the parameters $(\mu_1, \ldots, \mu_d)$ over the samples without any further knowledge of the distribution. Moreover, these guarantees apply no matter how the parameters $(\mu_1, \ldots, \mu_d)$ are tuned—optimally or suboptimally, manually or automatically. For *any* parameter setting, the empirical cost will be close to its expected cost. In Section 3.2.3, we generalize our analysis to cover non-linear combinations of the form $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$.

In Section 3.2.1, we provide guarantees for a family of scoring rules we call *path-wise*, which includes many well-known scoring rules as special cases. In this case, our bound is surprisingly small given the complexity of these problems: it grows only quadratically with the number of variables. In Section 3.2.2, we provide guarantees that apply to any scoring rule, path-wise or otherwise.

3.2.1 Path-Wise Scoring Rules. In this section, we provide guarantees for MILPs where the non-continuous variables are constrained to be binary. Later, in Sections 3.2.2 and 3.2.3, we provide guarantees for more general MILPs with arbitrary integer constraints. 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



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

(b) The path $\mathcal{T}_Q$ from the root of the tree $\mathcal{T}$ in Figure 2a to the node labeled Q.

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

Fig. 2. 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 $x[i]$, $\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}', Q, i) = \texttt{score}(\mathcal{T}_Q, Q, i)$.

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 2(b) illustrates the path $\mathcal{T}_Q$ from the root of the tree $\mathcal{T}$ in Figure 2(a) 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 $x[i]$,

$$\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}_Q, Q, i) \tag{4}$$

where $\mathcal{T}_Q$ is the path from the root of $\mathcal{T}$ to Q.[2] See Figure 2 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; this is our approach in our experiments. In fact, those scoring rules depend only on the node in question, rather than the path from the root to the node. We present our 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 3.2.2 do apply to those strategies.

In order to prove our guarantees, we make use of the following key structure, which bounds the number of search trees B&B will build on a given instance over the 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 the algorithm class's intrinsic complexity implies this section's generalization guarantee.

[2]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 (4) always holds.

Lemma 3.2. *Let* $\texttt{cost}_\mu$ *be a tree-constant cost function, let* $\texttt{score}_1$ *and* $\texttt{score}_2$ *be two path-wise scoring rules, and let* Q *be an arbitrary MILP 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 search tree.*

Lemma 3.2 implies that for any MILP Q, the dual function $\texttt{cost}^*_Q(\mu)$ is piecewise-constant with at most $2^{n(n-1)/2}n^n$ pieces.

Proof of Lemma 3.2. 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 feasible solutions to the original MILP or if they are 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. The full proof of the following claim is in Appendix B.

Claim 3.3. *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 Sketch of Claim 3.3. 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 $\mathcal{T}_\mu$ and $\mathcal{T}_{\mu'}$ 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 $\mathcal{T}_\mu[i] = \mathcal{T}_{\mu'}[i]$. Suppose that this is indeed the case for some $i \in \{1, \ldots, n-1\}$ and consider an arbitrary interval I_j and any two parameters $\mu, \mu' \in I_j$. Consider an arbitrary node Q in $\mathcal{T}_\mu[i]$ (or equivalently, $\mathcal{T}_{\mu'}[i]$) at depth i. 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 $\mathcal{T}_{\mu,Q}$ 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 for any two parameter settings $\mu, \mu' \in I_j$, the path from the root to Q in $\mathcal{T}_{\mu,Q}$ is the same as the path from the root to Q in $\mathcal{T}_{\mu',Q}$, and we refer to this path as $\mathcal{T}_Q$. Given a parameter setting $\mu \in I_j$, the variable $x[k]$ will be branched on at node Q so long as $k = \text{argmax}_\ell\{\mu\texttt{score}_1(\mathcal{T}_{\mu,Q}, Q, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_{\mu,Q}, Q, \ell)\}$, or equivalently, so long as $k = \text{argmax}_\ell\{\mu\texttt{score}_1(\mathcal{T}_Q, Q, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_Q, Q, \ell)\}$. 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.

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. This is illustrated in Figure 3. Moreover, there are at most 2^i nodes at depth i, and each node similarly contributes a subpartition of I_j of size n. If we merge all 2^i partitions, we have $T' \leq 2^i(n-1)+1$ 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 $\mathcal{T}_\mu$ and $\mathcal{T}_{\mu'}$ 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 $\mathcal{T}_\mu[i+1] = \mathcal{T}_{\mu'}[i+1]$. We can similarly subdivide each interval $I_1, \ldots, I_T$. The claim then follows from counting the number of subdivisions. □

[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 $x[i]$, $\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}[j], Q, i)$.

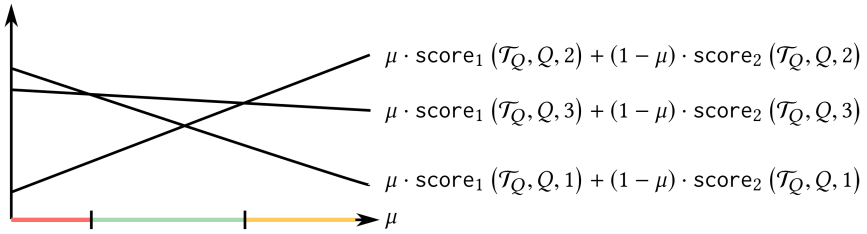


Fig. 3. Illustrations of the proof of Claim 3.3 for a hypothetical MILP Q where the algorithm can either branch on $x[1]$, $x[2]$, or $x[3]$ next. In the leftmost interval (colored pink), $x[1]$ will be branched on next, in the central interval (colored green), $x[3]$ will be branched on next, and in the rightmost interval (colored orange), $x[2]$ next will be branched on next.

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.

Claim 3.4. *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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$. 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 3.4. 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 $(j_\tau, x[i_\tau])$ 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 $(j_\tau, x[i_\tau])$. 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 $x[i_\tau]$ 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 $x[i_\tau]$.

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 $x[k]$, $\mu\texttt{score}_1(\bar{\mathcal{T}}, Q', k) + (1-\mu)\texttt{score}_2(\bar{\mathcal{T}}, Q', k) = \mu\texttt{score}_1(\mathcal{T}_{Q'}, Q', k) + (1-\mu)\texttt{score}_2(\mathcal{T}_{Q'}, Q', k) = \mu\texttt{score}_1(\bar{\mathcal{T}}', Q', k) + (1-\mu)\texttt{score}_2(\bar{\mathcal{T}}', Q', k)$. 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 3.3 and 3.4 to prove Lemma 3.2. Let $I_1, \ldots, I_T$ be the intervals guaranteed to exist by Claim 3.3 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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ and $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$ 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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ differs from B&B using the scoring rule $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$. By iteration, we mean lines 3 through 23 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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ or $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$. Let $\mathcal{T}_Q$ be the path from the root of $\mathcal{T}$ to Q. By Claim 3.3, 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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ (respectively, $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$) by the time it branches on the node Q. By Claims 3.3 and 3.4, 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 3.3, 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 $i = \text{argmax}_j\{\mu'\texttt{score}_1(\bar{\mathcal{T}}', Q, j) + (1-\mu')\texttt{score}_2(\bar{\mathcal{T}}', Q, j)\}$, or equivalently,

$$i = \text{argmax}_j \left\{\mu'\texttt{score}_1(\mathcal{T}_Q, Q, j) + (1-\mu')\texttt{score}_2(\mathcal{T}_Q, Q, j)\right\}, \tag{5}$$

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

$$i = \text{argmax}_j \left\{\mu''\texttt{score}_1(\mathcal{T}_Q, Q, j) + (1-\mu'')\texttt{score}_2(\mathcal{T}_Q, Q, j)\right\}. \tag{6}$$

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

$$i = \text{argmax}_j \{\mu'\texttt{score}_1(\mathcal{T}, Q, j) + (1-\mu')\texttt{score}_2(\mathcal{T}, Q, j)\}$$

and Equation (6) implies that $i = \text{argmax}_j \{\mu''\texttt{score}_1(\mathcal{T}, Q, j) + (1-\mu'')\texttt{score}_2(\mathcal{T}, Q, j)\}$. 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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ or $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$. Therefore, we have reached a contradiction: the iterations were identical. We conclude that the lemma holds. □

We now show how this structure implies a generalization guarantee. We formulate our guarantees in terms of *Rademacher complexity*[4] [Koltchinskii 2001]. According to classic results from learning theory, a Rademacher complexity bound immediately implies a generalization bound. The Rademacher complexity of a function class $\mathcal{F}$ intuitively measures the extent to which functions in $\mathcal{F}$ match random noise vectors $\boldsymbol{\sigma} \in \{-1, 1\}^m$ and is formally defined as follows:

Definition 3.5 (Rademacher Complexity). Let $\mathcal{F} = \{f_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in [0,1]^d\}$ be a set of functions mapping an abstract domain $\mathcal{Z}$ to $[0, H]$. The *empirical Rademacher complexity* of $\mathcal{F}$ given a set $\mathcal{S} = \{z_1, \ldots, z_m\} \subseteq \mathcal{Z}$ is

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{m}\mathbb{E}_{\boldsymbol{\sigma}\sim\{-1,1\}^m}\left[\sup_{\boldsymbol{\mu}\in[0,1]^d}\sum_{i=1}^{m}\sigma_i f_{\boldsymbol{\mu}}(z_i)\right],$$

where each σ_i equals -1 or 1 with equal probability.

The summation $\sum_{i=1}^m \sigma_i f_{\boldsymbol{\mu}}(z_i)$ measures the correlation between the random noise vector $\boldsymbol{\sigma}$ and the vector $(f_{\boldsymbol{\mu}}(z_1), \ldots, f_{\boldsymbol{\mu}}(z_m))$. By taking the supremum over all parameter vectors $\boldsymbol{\mu} \in [0,1]^d$, we measure how well functions in the class $\mathcal{F}$ correlate with $\boldsymbol{\sigma}$ over the sample $\mathcal{S}$. Therefore,

[4]In Theorem B.12 of Appendix B, we also show that Lemma 3.2 implies a pseudodimension bound.

$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F})$ measures how well functions in the class $\mathcal{F}$ correlate with random noise on average over $\mathcal{S}$. Rademacher complexity thus provides a way to measure the intrinsic complexity of $\mathcal{F}$ because the more complex the class $\mathcal{F}$ is, the better its functions can correlate with random noise.

Classic learning-theoretic results provide guarantees based on Rademacher complexity, such as the following:

Theorem 3.6 (e.g., Mohri et al. [2012]). *For any $\delta \in (0,1)$, with probability $1-\delta$ over the draw of m samples $\mathcal{S} = \{z_1, \ldots, z_m\} \sim \mathcal{D}^m$, for all functions $f_{\boldsymbol{\mu}} \in \mathcal{F}$,*

$$\left| \frac{1}{m} \sum_{i=1}^{m} f_{\boldsymbol{\mu}}(z_i) - \mathbb{E}\left[f_{\boldsymbol{\mu}}(z) \right] \right| = O\left(\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) + H \sqrt{\frac{1}{m} \log \frac{1}{\delta}} \right).$$

Theorem 3.6 is a *generalization guarantee* because it measures the extent to which a function's empirical average value over the samples generalizes to its expected value.

Ideally, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F})$ converges to zero as the sample size m grows, so the bound in Theorem 3.6 also converges to zero. If the class $\mathcal{F}$ consists of just a single function, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = 0$, and Theorem 3.6 recovers Hoeffding's bound. If, for example, $\mathcal{Z} = [0,1]$ and $\mathcal{F} = [0,1]^{[0,1]}$, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{2}$, and the bound from Theorem 3.6 is meaningless.

To prove our Rademacher complexity bounds, we will use the following lemma:

Lemma 3.7 (Massart [2000]). *Let $A \subseteq [0,\kappa]^m$ be a finite set of vectors. Then*

$$\frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma} \sim \{-1,1\}^m} \left[\sup_{\boldsymbol{a} \in A} \sum_{i=1}^{m} \sigma_i a[i] \right] \leq \kappa \sqrt{\frac{2 \ln |A|}{m}}.$$

We now provide a Rademacher complexity bound for our B&B problem.

Theorem 3.8. *Let $\texttt{score}_1$ and $\texttt{score}_2$ be two path-wise scoring rules and let C be the set of functions $C = \{\texttt{cost}_\mu : \mu \in [0,1]\}$ mapping MILPs to $[0,H]$. For any set $\mathcal{S} = \{Q_1, \ldots, Q_m\}$ of MILPs,*

$$\widehat{\mathcal{R}}_{\mathcal{S}} \leq H \sqrt{\frac{2 \ln \left(m \left(2^{n(n-1)/2} n^n - 1 \right) + 1 \right)}{m}} = O\left(H \sqrt{\frac{n^2 + \ln m}{m}} \right).$$

Proof. We will use Massart's lemma (Lemma 3.7) to prove this lemma. Let $A \subseteq [0,H]^m$ be the following set of vectors:

$$A = \left\{ \begin{pmatrix} \texttt{cost}_\mu(Q_1) \\ \vdots \\ \texttt{cost}_\mu(Q_m) \end{pmatrix} : \mu \in [0,1] \right\}.$$

By definition, this means that

$$\widehat{\mathcal{R}}_{\mathcal{S}}(C) = \frac{1}{m} \mathbb{E}_{\boldsymbol{\sigma} \sim \{-1,1\}^m} \left[\sup_{\boldsymbol{a} \in A} \sum_{i=1}^{m} \sigma_i a[i] \right]$$

Moreover, by definition of the dual class,

$$A = \left\{ \begin{pmatrix} \texttt{cost}^*_{Q_1}(\mu) \\ \vdots \\ \texttt{cost}^*_{Q_m}(\mu) \end{pmatrix} : \mu \in [0,1] \right\}.$$

By Lemma 3.2, each dual function $\texttt{cost}^*_{Q_i}(\mu)$ is piecewise-constant with at most $2^{n(n-1)/2} n^n$ pieces. As we range μ from 0 to 1, the vector $(\texttt{cost}^*_{Q_1}(\mu), \ldots, \texttt{cost}^*_{Q_m}(\mu))$ will only change when μ crosses

a boundary of one of the m piecewise-constant functions' boundaries, of which there are at most $m(2^{n(n-1)/2}n^n - 1)$. In other words, $|A| \le m(2^{n(n-1)/2}n^n - 1) + 1$. The lemma statement therefore follows from Massart's lemma. □

Theorems 3.6 and 3.8 immediately imply the following guarantee.

Corollary 3.9. *Let* $\texttt{score}_1$ *and* $\texttt{score}_2$ *be two path-wise scoring rules and for all* $\mu \in [0,1]$, $\texttt{cost}_\mu$ *be a tree-constant cost function with range* $[0,H]$. *For any distribution* $\mathcal{D}$ *over MILPs with at most* n *binary variables and any* $\delta \in (0,1)$, *with probability at least* $1-\delta$ *over the draw of* m *samples* $Q_1,\dots,Q_m \sim \mathcal{D}$, *for any* $\mu \in [0,1]$,

$$\left|\mathbb{E}_{Q\sim\mathcal{D}}\left[\texttt{cost}_\mu(Q)\right] - \frac{1}{m}\sum_{i=1}^{m}\texttt{cost}_\mu(Q_i)\right| \le O\left(H\sqrt{\frac{1}{m}\left(n^2 + \ln\frac{m}{\delta}\right)}\right).$$

3.2.2 General Scoring Rules. In this section, we provide generalization guarantees that apply to learning convex combinations of any set of scoring rules. These guarantees apply to MILPs with arbitrary integer constraints, whereas in Section 3.2.1, we required the noncontinuous variables to be binary. Unlike the guarantees in Section 3.2.1, they depend on the size of the search trees B&B is allowed to build before terminating, which we denote as κ. The following lemma corresponds to Lemma 3.2 for this setting. The full proof is in Appendix B.

Lemma 3.10. *Let* $\texttt{score}_1,\dots,\texttt{score}_d$ *be* d *arbitrary scoring rules, let* $\texttt{cost}_{\boldsymbol{\mu}}$ *be a tree-constant cost function for all* $\boldsymbol{\mu} \in [0,1]^d$ *and let* Q *be an arbitrary MILP over* n *integer variables. Suppose we limit B&B to producing search trees of size* κ. *There is a set* $\mathcal{H}$ *of at most* $\kappa n^{\kappa+2}$ *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* $\sum_{j=1}^{d}\mu_j\texttt{score}_j$ *is invariant across all* $(\mu_1,\dots,\mu_d) \in R$. *In other words,* $\texttt{cost}^*_Q(\boldsymbol{\mu})$ *is constant for all* $\boldsymbol{\mu} \in R$.

Proof Sketch. The proof has two steps. In Claim D.7, we show that there are at most n^κ different search trees that B&B might produce for the instance Q as we vary the mixing parameter vector $(\mu_1,\dots,\mu_d)$. In Claim D.8, for each of the possible search trees $\mathcal{T}$ that might be produced, we show that the set of parameter values $(\mu_1,\dots,\mu_d)$ which give rise to that tree lie in the intersection of κn^2 halfspaces. Of course, each of these halfspaces is defined by a hyperplane. Let $\mathcal{H}$ be the union of all κn^2 hyperplanes over all n^κ trees, so $|\mathcal{H}| \le \kappa n^{\kappa+2}$. 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 $\sum_{j=1}^{d}\mu_j\texttt{score}_j$ is invariant across all $(\mu_1,\dots,\mu_d) \in R$, so the lemma statement holds. □

In the same way Lemma 3.2 implies the Rademacher complexity bound in Theorem 3.8, Lemma 3.10 also implies a Rademacher complexity[5] bound. The proof, which is in Appendix B, is similar to that of Theorem 3.8.

Theorem 3.11. *Let* $\texttt{score}_1,\dots,\texttt{score}_d$ *be* d *arbitrary scoring rules and let* $\texttt{cost}_{\boldsymbol{\mu}}$ *be a tree-constant cost function for all* $\boldsymbol{\mu} \in [0,1]^d$. *Let* C *be the set of functions* $C = \{\texttt{cost}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in [0,1]^d\}$ *mapping MILPs to* $[0,H]$. *For any set* $\mathcal{S} = \{Q_1,\dots,Q_m\}$ *of MILPs,*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(C) = O\left(H\sqrt{\frac{d(\ln(m\kappa) + \kappa\ln n)}{m}}\right).$$

This Rademacher complexity bound implies the following corollary.

[5]It also implies a pseudodimension bound: Theorem B.15 in Appendix B.

Corollary 3.12. *Let* $\texttt{score}_1, \ldots, \texttt{score}_d$ *be* d *arbitrary scoring rules and let* $\texttt{cost}_{\boldsymbol{\mu}}$ *be a tree-constant cost function for all* $\boldsymbol{\mu} \in [0,1]^d$. *For any distribution* $\mathcal{D}$ *over MILPs* Q *with at most* n *integer variables, with probability at least* $1-\delta$ *over the draw of* m *MILPs* $Q_1, \ldots, Q_m \sim \mathcal{D}$, *for any* $\boldsymbol{\mu} \in [0,1]^d$,

$$\left| \mathbb{E}_{Q\sim\mathcal{D}} \left[\texttt{cost}_{\boldsymbol{\mu}}(Q) \right] - \frac{1}{m} \sum_{i=1}^{m} \texttt{cost}_{\boldsymbol{\mu}}(Q_i) \right| = O\left(H \sqrt{\frac{1}{m} \left(d(\ln(m\kappa) + \kappa \ln n) + \log \frac{1}{\delta} \right)} \right).$$

3.2.3 Generalization to Nonlinear Combinations. In this section, we show that our analysis from the previous sections also implies guarantees for learning nonlinear combinations of the form $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$. In the following lemma, we prove that generalization bounds for learning convex combinations of scoring rules imply generalization bounds for learning nonlinear combinations of scoring rules. It formally shows that if m samples are sufficient to ensure that for any convex combination $\sum_{j=1}^{d} \mu_j \texttt{score}_j$, average cost over the samples is ϵ-close to expected cost, then m samples are also sufficient to ensure the same holds for the non-convex combination $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$. Technically, this lemma holds for any d scoring rules $\texttt{score}_1, \ldots, \texttt{score}_d$ from a set Ψ that is *closed under logarithm*: for any function $\texttt{score} \in \Psi$, $\log \texttt{score}$ is also in Ψ. For example, Ψ might be the set of all scoring rules, which we study in Section 3.2.2, or the set of path-wise scoring rules, which we study in Section 3.2.1. These sets are both closed under logarithm; for example, if score is path-wise, then log score is also path-wise.

In this section, to clarify whether the parameters control a sum or product, we use the notation $\texttt{cost}(Q, \sum_{j=1}^{d} \mu_j \texttt{score}_j)$ to denote B&B's cost when using the scoring rule $\sum_{j=1}^{d} \mu_j \texttt{score}_j$ and $\texttt{cost}(Q, \prod_{j=1}^{d} \texttt{score}_j^{\mu_j})$ to denote B&B's cost when using the scoring rule $\prod_{j=1}^{d} \texttt{score}_j^{\mu_j}$.

Lemma 3.13. *Let* Ψ *be a set of scoring rules that is closed under logarithm. Let* cost *be a tree-constant cost function. Suppose there exists a generalization function* $\epsilon_\Psi : \mathbb{N} \times (0,1) \to \mathbb{R}$ *such that for any distribution* $\mathcal{D}$ *over MILPs, any* $\texttt{score}_1, \ldots, \texttt{score}_d \in \Psi$, *any number of samples* $m \in \mathbb{N}$, *and any* $\delta \in (0,1)$, *with probability* $1-\delta$ *over the draw of* m *MILPs* $Q_1, \ldots, Q_m \sim \mathcal{D}$, *for any* $\mu_1, \ldots, \mu_d \in [0,1]$,

$$\left| \frac{1}{m} \sum_{i=1}^{m} \texttt{cost}\left(Q_i, \sum_{j=1}^{d} \mu_j \texttt{score}_j \right) - \mathbb{E}_{Q\sim\mathcal{D}} \left[\texttt{cost}\left(Q, \sum_{j=1}^{d} \mu_j \texttt{score}_j \right) \right] \right| \leq \epsilon_\Psi(N, \delta).$$

Then for any $\texttt{score}_1, \ldots, \texttt{score}_d \in \Psi$, *with probability at least* $1-\delta$ *over the draw of* m *MILPs* $Q_1, \ldots, Q_m \sim \mathcal{D}$, *for any* $\mu_1, \ldots, \mu_d \in [0,1]$,

$$\left| \frac{1}{m} \sum_{i=1}^{m} \texttt{cost}\left(Q_i, \prod_{j=1}^{d} \texttt{score}_j^{\mu_j} \right) - \mathbb{E}_{Q\sim\mathcal{D}} \left[\texttt{cost}\left(Q, \prod_{j=1}^{d} \texttt{score}_j^{\mu_j} \right) \right] \right| \leq \epsilon_\Psi(N, \delta).$$

Proof. In Lemma D.10 in Appendix D, we prove that, for any $\mu_1, \ldots, \mu_d \in [0,1]$, any MILP Q, and any scoring rules $\texttt{score}_1, \ldots, \texttt{score}_d$,

$$\texttt{cost}\left(Q, \prod_{i=1}^{d} \texttt{score}_i^{\mu_i} \right) = \texttt{cost}\left(Q, \sum_{i=1}^{d} \mu_i \log \texttt{score}_i \right). \tag{7}$$

This follows from the fact that for any partial tree $\mathcal{T}'$ and any MILP Q', if

$$j^* = \operatorname{argmax}_j \left\{ \prod_{i=1}^{d} \texttt{score}_i(\mathcal{T}', Q', j)^{\mu_i} \right\},$$

then $j^* = \operatorname{argmax}_j \left\{ \log \left(\prod_{i=1}^{d} \texttt{score}_i(\mathcal{T}', Q', j)^{\mu_i} \right) \right\}$, which means that

$$j^* = \operatorname{argmax}_j \left\{ \sum_{i=1}^{d} \mu_i \log \texttt{score}_i(\mathcal{T}', Q', j) \right\}.$$

The lemma statement follows from Equation (7) and the fact that Ψ is closed under logarithm. □

This implies the following corollary for learning non-linear combinations of path-wise scoring rules. It follows from Lemma 3.13, Corollary 3.9, and the fact that if score is path-wise, then log score is path-wise as well.

Corollary 3.14. *Let* cost *be a tree-constant cost function with range* $[0, H]$, *and let* $\texttt{score}_1$ *and* $\texttt{score}_2$ *be two path-wise scoring rules. For any distribution* $\mathcal{D}$ *over MILPs with at most* n *binary variables and any* $\delta \in (0, 1)$, *with probability at least* $1 - \delta$ *over the draw* m *MILPs* $Q_1, \ldots, Q_m \sim \mathcal{D}$, *for any* $\mu \in [0, 1]$,

$$\left| \mathbb{E}_{Q \sim \mathcal{D}} \left[\texttt{cost} \left(Q, \texttt{score}_1^{1-\mu} \texttt{score}_2^{\mu} \right) \right] - \frac{1}{m} \sum_{i=1}^{m} \texttt{cost} \left(Q_i, \texttt{score}_1^{1-\mu} \texttt{score}_2^{\mu} \right) \right|$$
$$= O\left(H \sqrt{\frac{1}{m} \left(n^2 + \ln \frac{m}{\delta} \right)} \right).$$

Finally, we prove a similar corollary for non-linear combinations of general scoring rules, which follows from Lemma 3.13 and Corollary 3.12.

Corollary 3.15. *Let* $\texttt{score}_1, \ldots, \texttt{score}_d$ *be* d *arbitrary scoring rules, let* cost *be a tree-constant cost function with range* $[0, H]$, *and let* κ *be a tree size cap. For any distribution* $\mathcal{D}$ *over MILPs* Q *with at most* n *integer variables, with probability at least* $1 - \delta$ *over the draw of* m *MILPs* $Q_1, \ldots, Q_m \sim \mathcal{D}$, *for any* $\mu_1, \ldots, \mu_d \in [0, 1]$,

$$\left| \mathbb{E}_{Q \sim \mathcal{D}} \left[\texttt{cost} \left(Q, \prod_{j=1}^{d} \texttt{score}_j^{\mu_j} \right) \right] - \frac{1}{m} \sum_{i=1}^{m} \texttt{cost} \left(Q_i, \prod_{j=1}^{d} \texttt{score}_j^{\mu_j} \right) \right|$$
$$= O\left(H \sqrt{\frac{1}{m} \left(d(\ln(m\kappa) + \kappa \ln n) + \log \frac{1}{\delta} \right)} \right).$$

3.3 Data-dependent Generalization Bounds via Dual Class Approximability

So far, we have proved generalization bounds by showing that the dual cost functions $\texttt{cost}^*_Q(\boldsymbol{\mu})$ are piecewise constant, and our bounds grow logarithmically with the number of pieces. However, these dual functions often exhibit additional structure that we can use to strengthen our generalization guarantees. In particular, these dual functions are often well-approximated by piecewise constant functions with far fewer pieces. In this section, we show how we can use the existence of these simple approximating functions to provide stronger bounds. The results in this section are data-dependent because they improve based on the number of pieces that define the simple approximating functions.

In Section 3.3.1, we begin by stating our data-dependent guarantees, abstracting away from integer programming. The results in this section apply to providing generalization bounds for any set of functions $\mathcal{F}$ with dual functions that are well-approximated under the L_∞-norm by some set of simple functions (in Section 3.3.2, we prove that this dependence on the L_∞-norm is necessary—it is not possible to provide learnability guarantees if the dual functions are only approximated

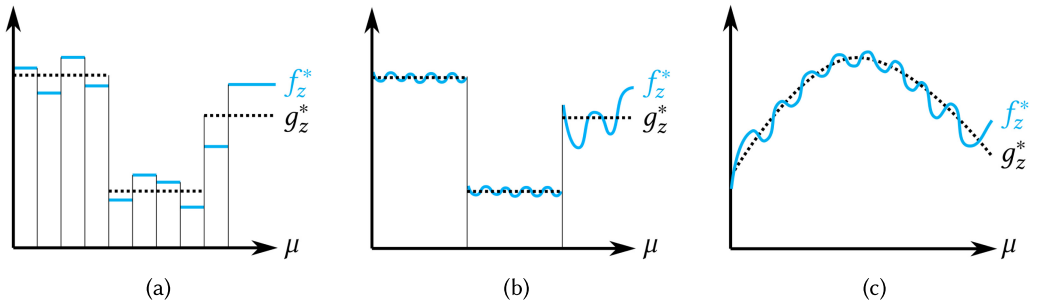


Fig. 4. Examples of dual functions $f_z^* : \mathbb{R} \to \mathbb{R}$ (solid blue lines) which are approximated by simpler functions g_z^* (dotted black lines).

under the L_p-norm for some $p < \infty$). In a bit more detail, suppose that $\mathcal{F}$ is a set of functions $f_{\boldsymbol{\mu}} : \mathcal{Z} \to [0, H]$ mapping an abstract domain $\mathcal{Z}$ (for example, a set of integer programs) to an interval $[0, H]$, parameterized by vectors $\boldsymbol{\mu}$ from some set $\mathcal{R}$. In the context of integer programming, $f_{\boldsymbol{\mu}} = \texttt{cost}_{\boldsymbol{\mu}}$—the cost function. The dual functions $f_z^* : \mathcal{R} \to [0, H]$ are defined as $f_z^*(\boldsymbol{\mu}) = f_{\boldsymbol{\mu}}(z)$, just as we defined the dual function $\texttt{cost}_Q^*(\boldsymbol{\mu}) = \texttt{cost}_{\boldsymbol{\mu}}(Q)$ in Section 3.1 for any integer program Q. We illustrate how one dual function might approximate another in Figure 4. In Section 3.3.3, we instantiate these guarantees in the context of integer programming.

3.3.1 Generally Applicable Generalization Bounds. First, we define what it means for two sets of dual functions to approximate each other. We use the notation $\mathcal{F}^* = \left\{f_z^* : z \in \mathcal{Z}\right\}$ to denote the *dual class*—the set of all dual functions.

Definition 3.16 ((γ, p)-approximate). Let $\mathcal{F} = \{f_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ and $\mathcal{G} = \{g_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ be two sets of functions mapping an abstract domain $\mathcal{Z}$ to $[0, H]$. We assume that all dual functions f_z^* and g_z^* are integrable over the domain $\mathcal{R}$. We say that the dual class $\mathcal{G}^*$ (γ, p)-*approximates* the dual class $\mathcal{F}^*$ if for every element $z \in \mathcal{Z}$, the distance between the functions f_z^* and g_z^* is at most γ under the L^p-norm. For $p \in [1, \infty)$, this means that $\left\|f_z^* - g_z^*\right\|_p := \sqrt[p]{\int_{\mathcal{R}} |f_z^*(\boldsymbol{\mu}) - g_z^*(\boldsymbol{\mu})|^p \, d\boldsymbol{\mu}} \leq \gamma$ and when $p = \infty$, this means that $\left\|f_z^* - g_z^*\right\|_\infty := \sup_{\boldsymbol{\mu} \in \mathcal{R}} |f_z^*(\boldsymbol{\mu}) - g_z^*(\boldsymbol{\mu})| \leq \gamma$.

We show that if the dual class $\mathcal{F}^*$ is (γ, ∞)-approximated by the dual of a class $\mathcal{G}$ with small Rademacher complexity, then the Rademacher complexity of $\mathcal{F}$ is also small. The full proof of the following theorem in Appendix B.1.

Theorem 3.17. *Let $\mathcal{F} = \{f_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ and $\mathcal{G} = \{g_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ consist of functions mapping $\mathcal{Z}$ to $[0, H]$. For any $\mathcal{S} \subseteq \mathcal{Z}$, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) \leq \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) + \frac{1}{|\mathcal{S}|} \sum_{z \in \mathcal{S}} \left\|f_z^* - g_z^*\right\|_\infty$.*

Proof Sketch. To prove this theorem, we use the fact that for any parameter vector $\boldsymbol{\mu} \in \mathcal{R}$, any element $z \in \mathcal{Z}$, and any binary value $\sigma \in \{-1, 1\}$, $\sigma f_{\boldsymbol{\mu}}(z) = \sigma f_z^*(\boldsymbol{\mu}) \leq \sigma g_z^*(\boldsymbol{\mu}) + \left\|f_z^* - g_z^*\right\|_\infty = \sigma g_{\boldsymbol{\mu}}(z) + \left\|f_z^* - g_z^*\right\|_\infty$. □

If the class $\mathcal{G}^*$ (γ, ∞)-approximates the class $\mathcal{F}^*$, then $\frac{1}{|\mathcal{S}|} \sum_{z \in \mathcal{S}} \left\|f_z^* - g_z^*\right\|_\infty$ is at most γ. If this term is smaller than γ for most sets $\mathcal{S} \sim \mathcal{D}^m$, then the bound on $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F})$ in Theorem 3.17 will often be even better than $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) + \gamma$.

Theorems 3.6 and 3.17 imply that with probability $1 - \delta$ over the draw of the set $\mathcal{S} \sim \mathcal{D}^m$, for all parameter vectors $\boldsymbol{\mu} \in \mathcal{R}$, the difference between the empirical average value of $f_{\boldsymbol{\mu}}$ over $\mathcal{S}$ and its

expected value is at most $\tilde{O}(\frac{1}{m}\sum_{z\in\mathcal{S}}\left\|f_z^* - g_z^*\right\|_\infty + \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) + H\sqrt{\frac{1}{m}})$. In our experiments, we show that this data-dependent generalization guarantee can be much tighter than worst-case bounds.

Algorithm for Finding Approximating Functions. We provide a **dynamic programming (DP)** algorithm (Algorithm 2 in Appendix B.1) for the common case where $\mathcal{R} = \mathbb{R}$ and the dual functions f_z^* are piecewise constant with a large number of pieces. Given an integer k, the algorithm returns a piecewise-constant function g_z^* with at most k pieces such that $\left\|f_z^* - g_z^*\right\|_\infty$ is minimized, as in Figure 4(a). Letting t be the number of pieces in the piecewise decomposition of f_z^*, the DP algorithm runs in $O(kt^2)$ time. As we describe later on in Section 3.3.3, when k and $\left\|f_z^* - g_z^*\right\|_\infty$ are small, Theorem 3.17 implies strong guarantees. We use this DP algorithm in our experiments.

Structural Risk Minimization. Theorem 3.17 illustrates a fundamental tradeoff in machine learning. The simpler the class $\mathcal{G}$, the smaller its Rademacher complexity, but—broadly speaking—the worse functions from its dual will be at approximating functions in $\mathcal{F}^*$. In other words, the simpler $\mathcal{G}$ is, the worse the approximation $\frac{1}{m}\sum_{z\in\mathcal{S}}\|f_z^* - g_z^*\|_\infty$ will likely be. Therefore, there is a tradeoff between generalizability and approximability. It may not be *a priori* clear how to balance this tradeoff. *Structural risk minimization (SRM)* is a classic, well-studied approach for optimizing tradeoffs between complexity and generalizability which we use in our experiments.

Our SRM approach is based on the following corollary of Theorem 3.17. Let $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \ldots$ be a countable sequence of function classes where each $\mathcal{G}_j = \{g_{j,\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ is a set of functions mapping $\mathcal{Z}$ to $[0, H]$. We use the notation $g_{j,z}^*$ to denote the duals of the functions in $\mathcal{G}_j$, so $g_{j,z}^*(\boldsymbol{\mu}) = g_{j,\boldsymbol{\mu}}(z)$.

Corollary 3.18. *With probability* $1-\delta$ *over the draw of the set* $\mathcal{S} \sim \mathcal{D}^m$, *for all* $\boldsymbol{\mu} \in \mathcal{R}$ *and all* $j \geq 1$,

$$\left|\frac{1}{m}\sum_{z\in\mathcal{S}} f_{\boldsymbol{\mu}}(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[f_{\boldsymbol{\mu}}(z)\right]\right| = O\left(\frac{1}{m}\sum_{z\in\mathcal{S}}\left\|f_z^* - g_{j,z}^*\right\|_\infty + \widehat{\mathcal{R}}_{\mathcal{S}}\left(\mathcal{G}_j\right) + H\sqrt{\frac{1}{m}\ln\frac{j}{\delta}}\right). \tag{8}$$

The proof of this corollary is in Appendix B.1.

In our experiments, each dual class $\mathcal{G}_j^*$ consists of piecewise-constant functions with at most j pieces. This means that as j grows, the class $\mathcal{G}_j^*$ becomes more complex, or in other words, the Rademacher complexity $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}_j)$ also grows. Meanwhile, the more pieces a piecewise-constant function g_z^* has, the better it is able to approximate the dual function f_z^*. In other words, as j grows, the approximation term $\frac{1}{m}\sum_{z\in\mathcal{S}}\|f_z^* - g_{j,z}^*\|_\infty$ shrinks. SRM is the process of finding the level j in the nested hierarchy that minimizes the sum of these two terms and therefore obtains the best generalization guarantee via Equation (8).

Remark 3.19. We conclude by noting that the empirical average $\frac{1}{m}\sum_{z\in\mathcal{S}}\|f_z^* - g_{j,z}^*\|_\infty$ in Equation (8) can be replaced by the expectation $\mathbb{E}_{z\sim\mathcal{D}}[\|f_z^* - g_{j,z}^*\|_\infty]$. See Corollary B.16 in Appendix B.1 for the proof.

3.3.2 Rademacher Complexity Lower Bound. In this section, we show that (γ, p)-approximability with $p < \infty$ does not necessarily imply strong generalization guarantees. We show that it is possible for a dual class $\mathcal{F}^*$ to be well approximated by the dual of a class $\mathcal{G}$ with $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) = 0$, yet for the primal $\mathcal{F}$ to have high Rademacher complexity.

Figures 5 and 6 help explain why there is this sharp constrast between the L^∞- and L^p-norms for $p < \infty$. Figure 5 illustrates two dual functions $f_{z_1}^*$ (the blue solid line) and $f_{z_2}^*$ (the grey dotted line). Let $\mathcal{G}$ be the extremely simple function class $\mathcal{G} = \{g_\mu : \mu \in \mathbb{R}\}$ where $g_\mu(z) = \frac{1}{2}$ for every $z \in \mathcal{Z}$. It is easy to see that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) = 0$ for any set $\mathcal{S}$. Moreover, every dual function g_z^* is also

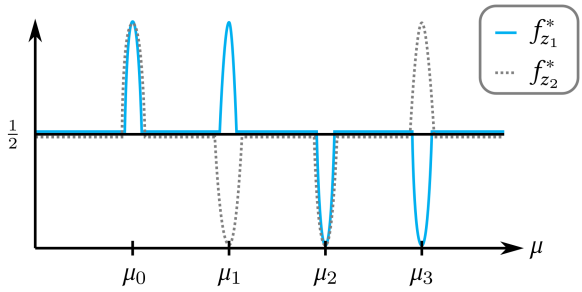


Fig. 5. The dual functions $f^*_{z_1}$ and $f^*_{z_2}$ are well-approximated by the constant function $\mu \mapsto \frac{1}{2}$ under, for example, the L^1-norm because the integrals $\int_{\mathcal{R}} |f^*_{z_i}(\mu) - \frac{1}{2}|\, d\mu$ are small; for most μ, $f^*_{z_i}(\mu) = \frac{1}{2}$. The approximation is not strong under the L^∞-norm, since $\max_{\mu\in\mathcal{R}} |f^*_{z_i}(\mu) - \frac{1}{2}| = \frac{1}{2}$. The function class $\mathcal{F}$ corresponding to these duals has a large Rademacher complexity.

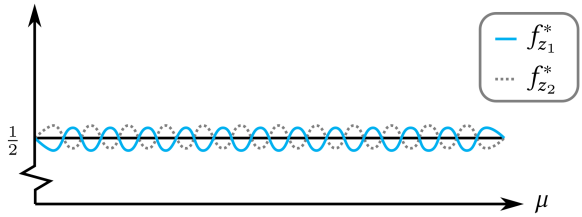


Fig. 6. The dual functions $f^*_{z_1}$ and $f^*_{z_2}$ are well-approximated by the constant function $\mu \mapsto \frac{1}{2}$ under the L^∞-norm since $\max_{\mu\in\mathcal{R}} |f^*_{z_i}(\mu) - \frac{1}{2}|$ is small. The function class $\mathcal{F}$ corresponding to these duals has a small Rademacher complexity.

simple, because $g^*_z(\mu) = g_\mu(z) = \frac{1}{2}$. From Figure 5, we can see that the functions $f^*_{z_1}$ and $f^*_{z_2}$ are well approximated by the constant function $g^*_{z_1}(\mu) = g^*_{z_2}(\mu) = \frac{1}{2}$ under, for example, the L^1-norm because the integrals $\int_{\mathcal{R}} |f^*_{z_i}(\mu) - \frac{1}{2}|\, d\mu$ are small. However, the approximation is not strong under the L^∞-norm, since $\max_{\mu\in\mathcal{R}} |f^*_{z_i}(\mu) - \frac{1}{2}| = \frac{1}{2}$ for $i \in \{1, 2\}$.

Moreover, despite the fact that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) = 0$, we have that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{2}$ when $\mathcal{S} = \{z_1, z_2\}$, which makes Theorem 3.6 meaningless. At a high level, this is because, when $\sigma_1 = 1$, we can ensure that $\sigma_1 f_\mu(z_1) = \sigma_1 f^*_{z_1}(\mu) = 1$ by choosing $\mu \in \{\mu_0, \mu_1\}$ and, when $\sigma_1 = -1$, we can ensure that $\sigma_1 f_\mu(z_1) = 0$ by choosing $\mu \in \{\mu_2, \mu_3\}$. A similar argument holds for σ_2. In summary, (γ, p)-approximability for $p < \infty$ does not guarantee low Rademacher complexity.

Meanwhile, in Figure 6, $g^*_{z_i}(\mu) = \frac{1}{2}$ and $f^*_{z_i}(\mu)$ are close for every parameter μ. As a result, for any noise vector $\boldsymbol{\sigma}$, $\sup_{\mu\in\mathbb{R}}\{\sigma_1 f^*_{z_1}(\mu) + \sigma_2 f^*_{z_2}(\mu)\}$ is close to $\sup_{\mu\in\mathbb{R}}\{\sigma_1 g^*_{z_1}(\mu) + \sigma_2 g^*_{z_2}(\mu)\}$. This implies that the Rademacher complexities $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G})$ and $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F})$ are close. This illustration exemplifies Theorem 3.17: (γ, ∞)-approximability implies strong Rademacher bounds.

We now prove that (γ, p)-approximability by a simple class for $p < \infty$ does not guarantee low Rademacher complexity.

Theorem 3.20. *For any $\gamma \in (0, 1/4)$ and any $p \in [1, \infty)$, there exist function classes $\mathcal{F}, \mathcal{G} \subset [0, H]^{\mathcal{Z}}$ such that the dual class $\mathcal{G}^*$ (γ, p)-approximates $\mathcal{F}^*$ and for any $m \geq 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=m} \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) = 0$ and $\sup_{\mathcal{S}:|\mathcal{S}|=m} \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{2}$.*

Proof. We begin by defining the classes $\mathcal{F}$ and $\mathcal{G}$. Let $\mathcal{R} = (0, \gamma^p]$, and $\mathcal{Z} = [\gamma^{-p}/2, \infty)$. For any $\mu \in \mathcal{R}$ and $z \in \mathcal{Z}$, let $f_\mu(z) = \frac{1}{2}(1 + \cos(\mu z))$ and $\mathcal{F} = \{f_\mu \mid \mu \in \mathcal{R}\}$. These sinusoidal functions are based on the intuition from Figure 5. As in Figure 5, for any μ and z, let $g_\mu(z) = \frac{1}{2}$ and $\mathcal{G} = \{g_\mu \mid \mu \in \mathcal{R}\}$. Since $\mathcal{G}$ consists of identical copies of a single function, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) = 0$ for any set $\mathcal{S} \subseteq \mathcal{Z}$. Meanwhile, in Lemma B.20 in Section B.1 in Appendix B, we prove that for any $m \geq 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=m} \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \frac{1}{2}$.

In Lemma B.19 in Appendix B, we prove that the dual class $\mathcal{G}^*$ (γ, p)-approximates $\mathcal{F}^*$. To prove this, we first show that $\left\|f_z^* - g_z^*\right\|_2 \leq \frac{1}{4}\sqrt{2\gamma^p + \frac{1}{z}}$. When $p = 2$, we know $\frac{1}{z} \leq 2\gamma^2$, so $\left\|f_z^* - g_z^*\right\|_2 < \gamma$. Otherwise, we use our bound on $\left\|f_z^* - g_z^*\right\|_2$, Hölder's inequality, and the log-convexity of the L^p-norm to prove that $\left\|f_z^* - g_z^*\right\|_p \leq \gamma$. □

Remark 3.21. Suppose, for example, that $\mathcal{R} = [0, 1]^d$. Theorem 3.20 implies that even if

$$\left|f_z^*(\boldsymbol{\mu}) - g_z^*(\boldsymbol{\mu})\right|$$

is small for all z in expectation over $\boldsymbol{\mu} \sim \text{Uniform}(\mathcal{R})$, the function class $\mathcal{F}$ may not have Rademacher complexity close to $\mathcal{G}$.

3.3.3 Generalization Bounds for Integer Programming. We now describe how the tools from Section 3.3.1 can be used to derive data-dependent generalization bounds for tuning the convex combination of any two scoring rules $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$.

To make use of Theorem 3.17 in the context of integer programming, we must define a function class whose duals approximate the functions $\texttt{cost}_Q^*$. We first define the dual class, then the primal class. To this end, fix some integer $j \geq 1$ and let $\mathcal{H}_j$ be the set of all piecewise-constant functions mapping $[0, 1]$ to $[0, 1]$ with at most j pieces. For every IP Q, we define $g_{j,Q}^* \in \text{argmin}_{h \in \mathcal{H}_j} \|\texttt{cost}_Q^* - h\|_\infty$, breaking ties in some fixed but arbitrary manner. We define the dual class $\mathcal{G}_j^*$ to be the set of functions $g_{j,Q}^*$. Therefore, the dual class $\mathcal{G}_j^*$ is consists of piecewise-constant functions with at most j pieces. For every parameter $\mu \in [0, 1]$ and IP Q, we define $g_{j,\mu}(Q) = g_{j,Q}^*(\mu)$. Finally, we define the primal class $\mathcal{G}_j = \{g_{j,\mu} \mid \mu \in [0, 1]\}$.

To apply our results from Section 3.3.1, we must bound the Rademacher complexity of the set $\mathcal{G}_j$. Doing so is simple due to the structure of the dual class $\mathcal{G}_j^*$. The following lemma is a corollary of Lemma B.23 in Appendix B.1 (which follows by the same logic as Theorem 3.8).

Lemma 3.22. *For any set $\mathcal{S}$ of integer programs, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}_j) \leq \sqrt{\frac{2\ln(|\mathcal{S}|(j-1)+1)}{|\mathcal{S}|}}$.*

This lemma together with Remark 3.19 and Corollary 3.18 imply that with probability $1 - \delta$ over $\mathcal{S} \sim \mathcal{D}^m$, for all parameters $\mu \in [0, 1]$ and $j \geq 1$, $|\frac{1}{m}\sum_{Q \in \mathcal{S}} \texttt{cost}_\mu(Q) - \mathbb{E}_{Q \sim \mathcal{D}}[\texttt{cost}_\mu(Q)]|$ is upper-bounded by

$$2\mathbb{E}_{Q \sim \mathcal{D}}\left[\left\|\texttt{cost}_Q^* - g_{j,Q}^*\right\|_\infty\right] + 2\sqrt{\frac{2\ln(m(j-1)+1)}{m}} + \sqrt{\frac{2}{m}\ln\frac{2(\pi j)^2}{3\delta}}. \tag{9}$$

As j grows, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}_j)$ grows, but the duals in $\mathcal{G}_j^*$ are better able to approximate the dual functions $\texttt{cost}_Q^*$. In our experiments, we optimize this tradeoff between generalizability and approximability.

3.4 Generalization to Constraint Satisfaction Problems

The generalization guarantees from Section 3.2 can be generalized to the problem of tree search for constraint satisfaction problems (CSPs). In this section, we provide an overview of tree search for CSPs. The generalization of our guarantees consists of a straightforward adaptation of the proofs from Section 3.2, so we include them in Appendix D for completeness.

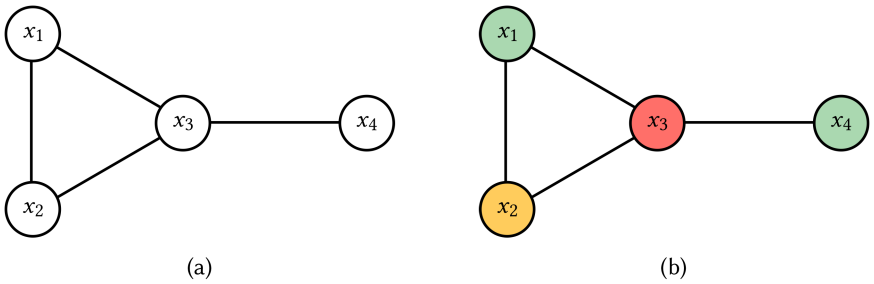


Fig. 7. Illustrations of Example 3.23.

A 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 $\left(\left(x_{i_1}, \ldots, x_{i_r}\right), \psi\right)$ 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 $\left(\left(x_{i_1}, \ldots, x_{i_r}\right), \psi\right)$ is satisfied if $\psi\left(y_{i_1}, \ldots, y_{i_r}\right) = 1$. 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 $\text{ddeg}(x, \boldsymbol{y})$ is the number of constraints involving x and at least one other unassigned variable.

Example 3.23 (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 7(a) 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 constraint on this problem is 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 7(b) for a coloring that satisfies all constraints ($y_1 = y_4 =$ green, $y_2 =$ orange, and $y_3 =$ pink).

3.4.1 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.

3.4.2 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

$$\texttt{score}(\mathcal{T}, Q, i) = \frac{\deg(x_i)}{|D_i|}$$

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.

3.5 Volatility of Branch-and-Bound Tree Size

In this section, we illustrate the volatility of the B&B tree size as a function of the variable selection parameters, which is one of the key challenges we faced in providing our generalization bounds. We prove that for any discretization of the parameter space [0, 1], there exists a distribution over MILP problem instances such that for any parameter in the discretization, the expected tree size is exponential in n. Yet, for any other parameter setting, the tree size is just a constant (with probability 1). The full proof of the following theorem is in Appendix B.

Theorem 3.24. *Let*

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

and $\texttt{cost}_\mu(Q)$ *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 exist a distribution* $\mathcal{D}$ *over MILP instances with* n *binary variables such that if* $\mu \in [0, 1] \setminus (a, b)$*, then*

$$\mathbb{E}_{Q\sim\mathcal{D}}[\texttt{cost}_\mu(Q)] = \Omega\left(2^{(n-9)/4}\right)$$

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

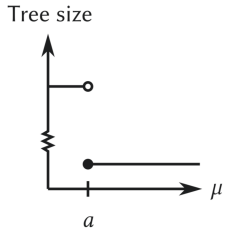
Proof Sketch. We populate the support of the distribution $\mathcal{D}$ by relying on two helpful theorems: Theorem 3.25 and B.5. In Theorem 3.25, we prove that for all $\mu^* \in (\frac{1}{3}, \frac{2}{3})$, there exists an infinite family $\mathcal{F}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{F}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ results in a B&B tree with $O(1)$ nodes and if $\mu \in (\mu^*, 1]$, the scoring rule results in a tree with $2^{(n-4)/2}$ nodes. Conversely, in Theorem B.5, 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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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 8 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 $\texttt{score} = \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ for any $\mu \in [0, 1]$, then the tree $\mathcal{T}$ B&B builds using NSP and score equals the tree $\mathcal{T}'$ it builds using NSP' and score (see Theorem B.10). Thus, the theorem holds for any node-selection policy. □

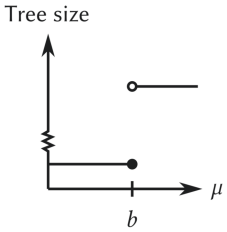
We now provide a proof sketch of Theorem 3.25, which helps us populate the support of the worst-case distribution in Theorem 3.24. The full proof is in Appendix B.

Theorem 3.25. *Let*

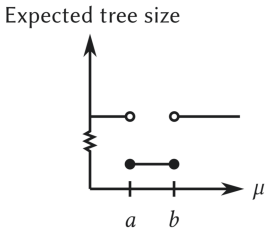
$$\texttt{score}_1(\mathcal{T}, Q, i) = \min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, \texttt{score}_2(\mathcal{T}, Q, i) = \max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\},$$



(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.

Fig. 8. Illustrations of the proof of Theorem 3.24.

and $\texttt{cost}_\mu(Q)$ *be the size of the tree produced by B&B. For all even* $n \geq 6$ *and all* $\mu^* \in (\frac{1}{3}, \frac{1}{2})$*, 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* $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ *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:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} & 2\sum_{i=1}^{n'} x[i] = n' \\ & \boldsymbol{x} \in \{0,1\}^{n'}. \end{array}$$

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 $\{x[n-2], x[n-1], x[n]\}$. B&B 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 $\boldsymbol{c}_2 = \gamma(0, \frac{3}{2}, 3 - \frac{1}{2\mu^*})$. 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

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} & 2\sum_{i=1}^{n-3} x[i] = n-3 \\ & 2\left(x[n-2] + x[n-1] + x[n]\right) = 3 \\ & \boldsymbol{x} \in \{0,1\}^n. \end{array}$$

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

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

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

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

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

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

Fig. 9. Illustrations of the construction from Theorem 3.25.

As is illustrated in Figure 9, 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 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 $\{x[1], \ldots, x[5]\}$ 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 $(0, 0, \frac{1}{2}, 1, 1)$. Similarly, the solution to the LP relaxation of $Q_{1,8}$ will set $(x[6], x[7], x[8]) = (0, \frac{1}{2}, 1)$ because $\frac{3}{2} < 3 - \frac{1}{2\mu^*}$ under our assumption that $\mu^* > \frac{1}{3}$. Thus, the solution to the LP relaxation of $Q_{1,8}$ is $(0, 0, \frac{1}{2}, 1, 1, 0, \frac{1}{2}, 1)$. 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, $\mu\texttt{score}_1(\mathcal{T}, Q_{\gamma,n}, 3) + (1-\mu)\texttt{score}_2(\mathcal{T}, Q_{\gamma,n}, 3) = \frac{\gamma}{2}$ and $\mu\texttt{score}_1(\mathcal{T}, Q_{\gamma,n}, 7) + (1-\mu)\texttt{score}_2(\mathcal{T}, Q_{\gamma,n}, 7) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$. This means that B&B will branch first on variable $x[7]$, which corresponds to the small version of Jeroslow's instance (see Figure 9(b)) if and only if $\frac{\gamma}{2} < \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$, 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 $\{x[1], \ldots, x[5]\}$, 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 9(a)) 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 $(x[1], \ldots, x[n-3])$, 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. □

4 EXPERIMENTS

In this section, we provide two sets of experiments. First, in Section 4.1, we show that the parameter of the variable-selection policy in B&B algorithms for MILP impacts the average tree size generated for several domains, and no parameter value is optimal across these distributions. We illustrate this phenomenon for the specific variable selection policies we analyze in Section 3 and refer the reader to the wealth of papers summarized in Section 1.2 that have illustrated this phenomenon for other aspects of branch-and-bound. Our focus in Section 4.1 is to provide additional evidence that using an automated approach to tuning these parameters can have a sizable impact on the performance of tree search (not to propose a state-of-the-art parameter optimization algorithm).

Next, in Section 4.2, we illustrate the data-dependent bounds developed in Section 3.3.1. We demonstrate that this data-dependent approach can lead to significantly smaller generalization bounds than the worst-case bounds from Section 3.2.2 when the dual functions are well approximated by piecewise-constant functions with few pieces.

Throughout this section, we use the following notation:

- $\texttt{score}_L(\mathcal{T}, Q, i) = \max\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}$ where (as defined in Section 2.1.2) $\check{c}_Q$, $\check{c}_{Q_i^-}$, and $\check{c}_{Q_i^-}$ are the objective values of the optimal solutions[6] to the LP relaxations of Q, Q_i^+, and Q_i^-. Under $\texttt{score}_L$, B&B branches on the variable leading to the **L**argest change in the LP objective value.
- $\texttt{score}_S(\mathcal{T}, Q, i) = \min\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}$. Under $\texttt{score}_S$, B&B branches on the variable leading to the **S**mallest change.
- $\texttt{score}_A(\mathcal{T}, Q, i) = \frac{1}{6}\texttt{score}_L(\mathcal{T}, Q, i) + \frac{5}{6}\texttt{score}_S(\mathcal{T}, Q, i)$. This is a scoring rule that **A**chterberg recommended [Achterberg 2009]. It balances the optimistic approach to branching under $\texttt{score}_L$ with the pessimistic approach under $\texttt{score}_S$.
- $\texttt{score}_P(\mathcal{T}, Q, i) = \max\{\check{c}_Q - \check{c}_{Q_i^+}, 10^{-6}\} \cdot \max\{\check{c}_Q - \check{c}_{Q_i^-}, 10^{-6}\}$: the ***P**roduct scoring rule.*

We use the C API of IBM ILOG CPLEX 12.8.0.0 to override the default variable-selection policy using a branch callback. The CPLEX node-selection policy is set to "best bound" (aka A^* in AI), which is the most typical choice in MILP. All experiments were run on a 64-core machine with 512 GB of RAM.[7]

4.1 Impact on Tree Size by Variable Selection Policy

We begin with an overview of the experiments in this section. We first fix two scoring rules $\texttt{score}_1$ and $\texttt{score}_2$. Our experiments in Section 4.1.3 illustrate tree size as a function of the parameter μ that controls the convex combination $(1-\mu)\texttt{score}_1 + \mu\texttt{score}_2$. Given a distribution over MILPs (the distributions we analyze are described in Section 4.1.1), we sample a set of MILPs. For each sampled MILP Q, we compute tree size as a function of the parameter $\mu \in [0, 1]$ which, as we know from Lemma 3.10, is a piecewise-constant function. We compute these piecewise-constant functions as described in Section 4.1.2. Taking the average of these piecewise-constant functions, we plot tree size as a function of the parameter μ on average[8] over the sampled MILPs. In Section 4.1.4, we run the same experiments, but we tune the parameter of the product $\texttt{score}_1^{1-\mu}\texttt{score}_2^{\mu}$.

4.1.1 *Distributions of MILPs.*

We now detail the distributions we analyze in our experiments.

Combinatorial Auctions. We generate 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. In Section 4.1.3, we use the "arbitrary" generator with 200 bids and 100 goods, resulting in MILPs with around 200 variables, and "regions" generator with 400 bids and 200 goods, resulting in MILPs with around 400 binary variables. In Section 4.1.4, we use the "arbitrary" generator with 150 bids and 100 goods, resulting in MILPs with around 150 binary variables, and "regions" generator with 300 bids and 200 goods, resulting in MILPs with around 300 binary variables.

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

[6] When computing $\check{c}_{Q_i^-}$ and $\check{c}_{Q_i^-}$ for all candidate variables, we limit CPLEX to run at most 10 dual steepest-edge iterations.

[7] Unlike the conference version of this article [Balcan et al. 2018a], we do not disable CPLEX's cuts and primal heuristics, and we do not disable its root-node preprocessing. The experiments in the conference version were run on a cluster of 64 c3.large Amazon AWS instances.

[8] In Appendix C, we also illustrate the geometric mean tree size as a function of the parameter μ, whereas the plots in this section illustrate the arithmetic mean tree size.

the following objective function is minimized: $\sum_{i=1}^{n} \min_{j\in[k]} d(p_i, c_j)^2$. In Appendix C, we show how to formulate this problem as a MILP. We generate 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 $\mathbb{R}^d$. These MILPs have 1,260 binary variables.

Agnostically Learning Linear Separators. Let $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N \in \mathbb{R}^d$ be 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.,

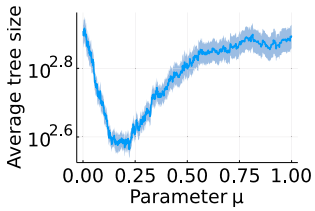
$$\sum_{i=1}^{N} \mathbf{1}_{\{z_i \langle \boldsymbol{p}_i, \boldsymbol{w}\rangle < 0\}}.$$

In Appendix C, we show how to formulate this problem as a MILP. We generate problem instances with 100 points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{100}$ 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 15 random points and flip their labels so that there is no consistent linear separator. These MILPs have 100 binary variables.

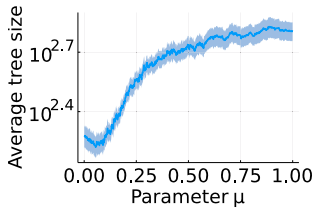
4.1.2 Constructing the Piecewise-Constant Functions. In this section, we describe how, for any MILP Q, we compute tree size as a function of the mixing parameter μ that controls the variable selection policy $(1-\mu)\texttt{score}_1 + \mu\texttt{score}_2$. We then describe how this procedure can be generalized to the product $\texttt{score}_1^{1-\mu}\texttt{score}_2^{\mu}$. From Section 3.5, we know that for any *data-independent* discretization of the parameter space $[0,1]$ (e.g., a naïve sweep $\{0, 0.01, 0.02, \ldots, 0.99, 1\}$), there are distributions where every parameter setting in the discretization will lead to exponential expected tree size, yet there are parameter settings between the discretized points that will lead to constant tree size. Therefore, in these experiments, we do not perform a naïve sweep, but rather, for every sampled MILP, compute the piecewise-constant function that defines tree size as a function of the parameter setting μ.

To do this, we use a CPLEX callback function so that given a parameter setting $\mu \in [0,1]$, we can keep track of the largest interval $I \subset [0,1]$ such that the tree CPLEX builds is identical to the tree it would have built using any parameter $\mu' \in I$. With this, we can enumerate all possible behaviors of the algorithm for a single instance Q by running the algorithm with $\mu = 0$, followed by the smallest value of μ that will lead to a different tree, and so on, until we have covered the entire interval $[0,1]$. To see why it is possible to find the interval I, suppose we are choosing which variable to branch on at 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 setting μ, we combine these two scores as $(1-\mu)\texttt{score}_1(\mathcal{T}, Q', i) + \mu\texttt{score}_2(\mathcal{T}, Q', i)$. Let i^* be the variable chosen by the algorithm when run with parameter setting μ. For any other parameter setting μ' such that i^* has the highest score, i^* will be the branching variable at this node, regardless of if the algorithm uses μ or μ'. The set of all μ' for which i^* is the variable of the highest score is an interval, as illustrated in Figure 3 (and its endpoints 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 variable, resulting in a different behavior of the tree search algorithm. By taking the intersection 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 bookkeeping is only linear in the number of candidate branching variables.

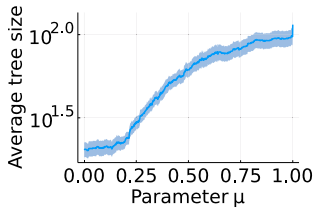
In Section 4.1.4 where we analyze the generalized product rule $\texttt{score}_1^{1-\mu}\texttt{score}_2^{\mu}$, we use the fact (proved Lemma D.10 from Appendix D) that for any $\mu \in [0,1]$, the tree that B&B builds



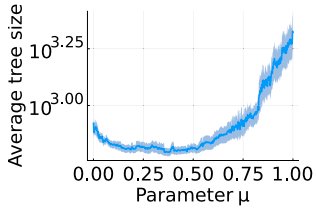
(a) CATS "arbitrary": 200 bids, 100 goods



(b) CATS "regions": 400 bids, 200 goods



(c) Clustering



(d) Linear separators

Fig. 10. The average tree size produced by B&B when run with the linear scoring rule with parameter μ (Equation (10)).

using the scoring rule $\texttt{score}_1^{1-\mu}\texttt{score}_2^{\mu}$ is the same as the tree B&B builds using the scoring rule $(1-\mu)\log \texttt{score}_1 + \mu \log \texttt{score}_2$. Therefore, to construct the piecewise-constant functions, we use the same procedure as described in the previous paragraph, though we first take the logarithm of each scoring rule.

4.1.3 Linear Scoring Rule. For the distributions described in Section 4.1.1, Figure 10 shows the average B&B tree size produced for each possible value of the μ parameter for the linear scoring rule, averaged over 100 independent samples from each distribution. Specifically, the scoring rule whose parameter we tune in Figure 10 has the form

$$\texttt{score}(\mathcal{T}, Q, i) = (1-\mu)\cdot \texttt{score}_S(\mathcal{T}, Q, i) + \mu \cdot \texttt{score}_L(\mathcal{T}, Q, i). \tag{10}$$

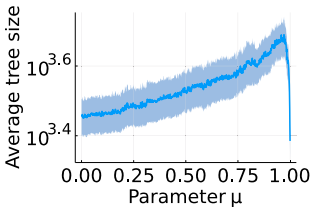
We use the standard error of the mean to plot uncertainty bands.

In Table 1, we specify the optimal parameter setting from each plot in Figure 10. We also indicate the total number of hours it took to compute all of the piecewise-constant functions that we average over in each plot. For example, in Figure 10(a), the parameter setting that minimizes average tree size is $\mu = 0.222$. It took a total of 109.16 hours to compute all 100 piecewise-constant functions that we average over in the plot.

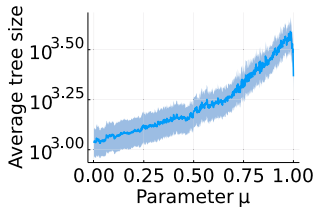
In Figure 11, we plot average tree size as a function of the μ parameter for the linear scoring rule, but we use pseudo-cost branching rather than strong branching (which we use in Figure 10). Using strong branching, B&B computes the changes in the LP relaxation objective values $\check{c}_Q - \check{c}_{Q_i^+}$ and $\check{c}_Q - \check{c}_{Q_i^-}$ for every variable, which is time-consuming since it involves evaluating $2n$ LPs at every node. Pseudocost branching instead estimates these values by averaging the LP objective value changes across all nodes in the tree where the ith variable was chosen to branch on. In Appendix A, we provide the formal definition of pseudocost branching. Pseudocost branching generally leads to larger trees than strong branching. In Figures 11(a), 11(b), and 11(d), we average over 100 randomly sampled IPs. Since Figure 11(c) is choppier than these other three, we average over 2,000 samples.

Table 1. For Each of the Plots in Figure 10, We Indicate the Empirically Optimal Parameter Setting and the Number of Hours It Took to Compute All of the Piecewise-constant Functions that We Average Over in Each Plot

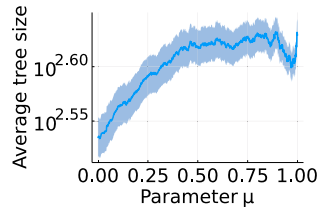
Distribution	Optimal parameter setting	Total compute time to find optimal parameter setting (hours)
CATS "arbitrary": 200 bids, 100 goods	0.222	109.16
CATS "regions": 400 bids, 200 goods	0.055	325.79
Clustering	0.020	4.51
Linear separators	0.379	45.49



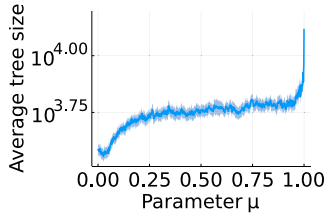
(a) CATS "arbitrary": 150 bids, 100 goods



(b) CATS "regions": 300 bids, 200 goods



(c) Clustering



(d) Linear separators

Fig. 11. The average tree size produced by B&B when run with the linear scoring rule with parameter μ using pseudo-cost branching.

Table 2, like Table 1, specifies the optimal parameter setting from each plot in Figure 11 and the total number of hours it took to compute all of the piecewise-constant functions that we average over in each plot.

4.1.4 Product Scoring Rule. Figure 12 illustrates the average B&B tree size produced when tuning the parameter of a generalized product rule, averaged over 100 samples from each distribution. This parameterized scoring rule has the form[9, 10]

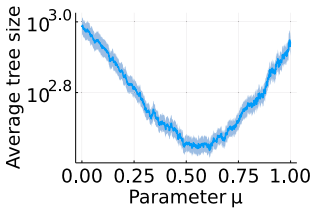
$$\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}_1(\mathcal{T}, Q, i)^{1-\mu}\texttt{score}_2(\mathcal{T}, Q, i)^{\mu}, \tag{11}$$

[9]Again, when computing $\check{c}_{Q_i^-}$ and $\check{c}_{Q_i^-}$ for all candidate variables, we limit CPLEX to run at most 10 dual steepest-edge iterations.

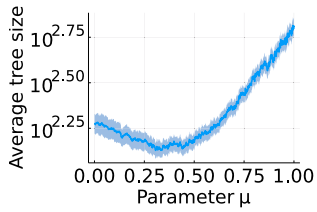
[10]As we write in Section 2.1.2, comparing $\check{c}_Q - \check{c}_{Q_i^-}$ and $\check{c}_Q - \check{c}_{Q_i^+}$ to 10^{-6} allows the algorithm to compare two variables even if $\check{c}_Q - \check{c}_{Q_i^-} = 0$ or $\check{c}_Q - \check{c}_{Q_i^+} = 0$. After all, suppose the scoring rule simply calculated the product $\min\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}^{1-\mu} \max\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\}^{\mu}$ without comparing to 10^{-6}. If one of these multiplicands equals 0, then the score equals 0, canceling out the value of the other multiplicand and thus losing the information encoded by that multiplicand.

Table 2. For Each of the Plots in Figure 11, We Indicate the Empirically Optimal Parameter Setting and the Number of Hours It Took to Compute All of the Piecewise-Constant Functions that We Average Over in Each Plot

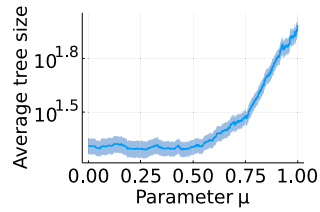
Distribution	Optimal parameter setting	Total compute time to find optimal parameter setting (hours)
CATS "arbitrary": 150 bids, 100 goods	1	3948.10
CATS "regions": 300 bids, 200 goods	0.029	4793.62
Clustering	0.009	2003.73
Linear separators	0.027	263.76



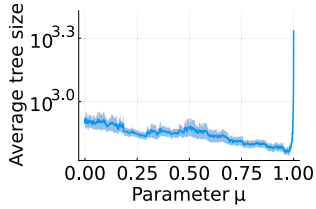
(a) CATS "arbitrary": 200 bids, 100 goods



(b) CATS "regions": 400 bids, 200 goods



(c) Clustering



(d) Linear separators

Fig. 12. The average tree size produced by B&B when run with the product scoring rule with parameter μ (Equation (11)) using strong branching.

where

$$\texttt{score}_1(\mathcal{T}, Q, i) = \max\left\{\min\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, 10^{-6}\right\}$$

and

$$\texttt{score}_2(\mathcal{T}, Q, i) = \max\left\{\max\left\{\check{c}_Q - \check{c}_{Q_i^+}, \check{c}_Q - \check{c}_{Q_i^-}\right\}, 10^{-6}\right\}.$$

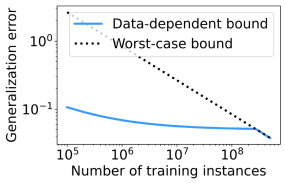
This scoring rule is equivalent to the product scoring rule from Section 2.1.2 when $\mu = \frac{1}{2}$ because for any $i, j \in [n]$, $\texttt{score}_1(\mathcal{T}, Q, i)^{\frac{1}{2}} \texttt{score}_2(\mathcal{T}, Q, i)^{\frac{1}{2}} \geq \texttt{score}_1(\mathcal{T}, Q, j)^{\frac{1}{2}} \texttt{score}_2(\mathcal{T}, Q, j)^{\frac{1}{2}}$ if and only if $\max\{\check{c}_Q - \check{c}_{Q_i^-}, 10^{-6}\} \cdot \max\{\check{c}_Q - \check{c}_{Q_i^+}, 10^{-6}\} \geq \max\{\check{c}_Q - \check{c}_{Q_j^-}, 10^{-6}\} \cdot \max\{\check{c}_Q - \check{c}_{Q_j^+}, 10^{-6}\}$.

As in Section 4.1.3, we also include Table 3, which specifies the optimal parameter setting from each plot in Figure 12 and the total number of hours it took to compute all of the piecewise-constant functions that we average over in each plot.

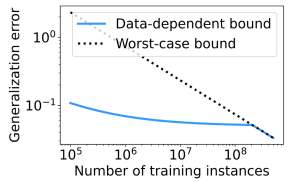
Discussion. The relationship between the variable-selection parameter and the average tree size varies from application to application. This implies that the parameters should be tuned on a per-

Table 3. For Each of the Plots in Figure 12, We Indicate the Empirically Optimal Parameter Setting and the Number of Hours It Took to Compute All of the Piecewise-Constant Functions that We Average Over in Each Plot

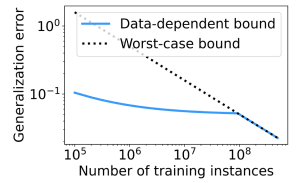
Distribution	Optimal parameter setting	Total compute time to find optimal parameter setting (hours)
CATS "arbitrary": 200 bids, 100 goods	0.611	421.19
CATS "regions": 400 bids, 200 goods	0.332	437.06
Clustering	0.275	7.04
Linear separators	0.961	77.35



(a) Results on the CATS "regions" generator with $\texttt{score}_1 = \texttt{score}_L$ and $\texttt{score}_2 = \texttt{score}_S$.



(b) Results on the CATS "arbitrary" generator with $\texttt{score}_1 = \texttt{score}_L$ and $\texttt{score}_2 = \texttt{score}_S$.



(c) Results on the CATS "arbitrary" generator with $\texttt{score}_1 = \texttt{score}_P$ and $\texttt{score}_2 = \texttt{score}_A$.

Fig. 13. Experiments where we compare the data-dependent generalization bound from Section 3.3.3 and the worst-case guarantee from Section 3.2.2. The blue solid line is our generalization bound: the minimum of Equations (12) and (14) as a function of the number of training examples m, divided by κ. The black dotted line is the worst-case bound from Equation (12).

application basis and that no parameter value is universally effective. For example, the optimal parameter in Figures 10(b), 10(c), and 10(d) is close to 0. However, $\mu = 0$ is suboptimal in Figure 10(a), resulting in trees that are twice the size of the trees obtained under the optimal parameter value.

In these experiments, average tree size is typically an unimodal function of the parameters, a phenomenon also observed by Pushak and Hoos [2018] in other algorithm parameter tuning contexts. We leave as an open question whether this structure could be used to provide data-dependent generalization bounds.

4.2 Data-Dependent Guarantees

In this section, we show how the data-dependent generalization guarantees from Section 3.3.3 can imply stronger bounds than the worst-case guarantees from Section 3.2.2. In this section, we again analyze distributions that we generate using the Combinatorial Auction Test Suite (CATS) [Leyton-Brown et al. 2000]. We use the "arbitrary" generator with 200 bids and 100 goods, resulting in IPs with 200 about variables, and the "regions" generator with 400 bids and 200 goods, resulting in IPs with 400 about variables. We use Algorithm 2 in Section B.1 in Appendix **??** to compute the approximating duals.

In Figure 13, we select $\texttt{score}_1, \texttt{score}_2 \in \{\texttt{score}_L, \texttt{score}_S, \texttt{score}_A, \texttt{score}_P\}$ and compare the worst-case and data-dependent bounds, divided by κ. In Section B.1 in Appendix B, we describe our methodology for choosing κ. First, we plot the worst-case bound implied by Corollary 3.12,

with $\delta = 0.01$, as a function of the number of training examples m. This is the black dotted line in Figure 13. With all of the proper constants, this bound is equal to

$$2\kappa\sqrt{\frac{2\ln(m\left(n^{2(\kappa+1)}-1\right)+1)}{m}} + 3\kappa\sqrt{\frac{1}{2m}\ln\frac{2}{0.01}}. \tag{12}$$

Next, we plot the data-dependent bound, which is the blue solid line in Figure 13. To calculate the data-dependent bound in Equation (9), we have to estimate $\mathbb{E}_{Q\sim\mathcal{D}}[\|\texttt{cost}^*_Q - g^*_{j,Q}\|_\infty]$ for all $j \in [1600]$.[11] To do so, we draw $M = 6000$ IPs $Q_1, \ldots, Q_M$ from the distribution $\mathcal{D}$. We estimate $\mathbb{E}_{Q\sim\mathcal{D}}[\|\texttt{cost}^*_Q - g^*_{j,Q}\|_\infty]$ via the empirical average $\frac{1}{M}\sum_{i=1}^M \|\texttt{cost}^*_{Q_i} - g^*_{j,Q_i}\|_\infty$. A Hoeffding bound guarantees that with probability 0.995, for all $j \in [1600]$,

$$\mathbb{E}\left[\left\|\texttt{cost}^*_Q - g^*_{j,Q}\right\|_\infty\right] \le \frac{1}{M}\sum_{i=1}^M \left\|\texttt{cost}^*_{Q_i} - g^*_{j,Q_i}\right\|_\infty + \frac{1}{40}. \tag{13}$$

We prove this inequality in Lemma B.24. We thereby estimate our data-dependent bound Equation (9) using the following bound:

$$\min_{j\in[1600]}\left\{2\left(\frac{1}{M}\sum_{i=1}^M \left\|\texttt{cost}^*_{Q_i} - g^*_{j,Q_i}\right\|_\infty + \frac{1}{40}\right) + 2\sqrt{\frac{2\ln(m(j-1)+1)}{m}} + \sqrt{\frac{2}{m}\ln\frac{(20\pi j)^2}{3}}\right\}. \tag{14}$$

The only difference between Equations (9) and (14) is that Equation (9) relies on the left-hand-side of Equation (13) and Equation (14) relies on the right-hand-side of Equation (13) and sets $\delta = 0.005$.[12] In Figure 13, the blue solid line equals the minimum of Equations (12) and (14) as a function of the number of training examples m.

In Figure 13, we see that our bound significantly beats the worst-case bound up until the point where there are approximately 100,000,000 training instances. At this point, the worst-case guarantee is better than the data-dependent bound, which makes sense because it goes to zero as m goes to infinity, whereas the term $\frac{1}{M}\sum_{i=1}^M \|\texttt{cost}^*_{Q_i} - g^*_{j,Q_i}\|_\infty + \frac{1}{40}$ in our bound (Equation (14)) is a constant.

Figure 13(a) also illustrates that even when there are only 10^5 training instances, our bound provides a generalization guarantee of approximately 0.1. Meanwhile, $7\cdot 10^7$ training instances are necessary to provide a generalization guarantee of 0.1 under the worst-case bound. Similarly, in Figure 13(b), 500 times fewer samples are required to obtain a generalization guarantee of 0.1 under our bound versus the worst-case bound. In Figure 13(c), 250 times fewer samples are required.

5 CONCLUSIONS AND BROADER APPLICABILITY

In this work, we provided the first generalization guarantees—both worst-case and data-dependent—for tree search parameter tuning. These guarantees bound the number of samples sufficient to ensure that the empirical cost incurred by using any mixture of scoring rules will be close to its expected cost, where cost is an abstract measure such as tree size. Through experiments, we showed that using the optimal parameter setting 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. While we presented the theory in the context of tree search, it also applies to other tree-growing applications. For example, it

[11] We choose the range $j \in [1600]$ because under these distributions, the functions $\texttt{cost}^*_Q$ generally have at most 1,600 pieces.

[12] Like the worst-case bound, Equation (14) holds with probability 0.99, because with probability 0.995, Equation (13) holds, and with probability 0.995, the bound from Equation (9) holds.

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.

For future research, Section 3.5 implies that for any discretization of the parameter space, an adversary could choose a distribution over MILPs that causes the difference between the best parameter setting within the discretization to be exponentially worse than any parameter setting outside of the discretization. However, if the adversary is first forced to choose a distribution from Section 3.5 and then a random sample of parameter settings is drawn, that random sample would certainly include a good parameter setting leading to constant tree size. Are there lower bounds that one can provide in this more challenging setup?

APPENDICES

A VARIABLE-SELECTION POLICIES

In practice, it is often too slow to compute the differences $\check{c}_Q - \check{c}_{Q_i^-}$ and $\check{c}_Q - \check{c}_{Q_i^+}$ for every variable, since it requires solving as many as $2n$ LPs. Pseudocost branching is an alternative to computing these values [Bénichou et al. 1971; Gauthier and Ribière 1977; Linderoth and Savelsbergh 1999]. To introduce pseudocost branching, we first define some notation, as presented in Achterberg's thesis [Achterberg 2007]. At a node Q, let $f^-_{Q,i} = \check{x}_Q[i] - \lfloor \check{x}_Q[i] \rfloor$ and $f^+_{Q,i} = \lceil \check{x}_Q[i] \rceil - \check{x}_Q[i]$ denote how far the ith component of the LP relaxation's solution is from being integral. Let $\varsigma^-_{Q,i}$ and $\varsigma^+_{Q,i}$ be the objective gains per unit change in variable $x[i]$ at node Q after branching in each direction. More formally,

$$\varsigma^-_{Q,i} = \frac{\check{c}_Q - \check{c}_{Q_i^-}}{f^-_{Q,i}} \quad \text{and} \quad \varsigma^+_{Q,i} = \frac{\check{c}_Q - \check{c}_{Q_i^+}}{f^+_{Q,i}}.$$

Let σ_i^- be the sum of $\varsigma^-_{Q,i}$ over all nodes Q where $x[i]$ was chosen as the branching variable and the LP relaxation of the node Q_i^- has already been solved. Let η_i^- be the number of such nodes. Let σ_i^+ and η_i^+ be the corresponding values for the upwards branches. The pseudocosts of variable $x[i]$ are defined as

$$\Psi_i^- = \frac{\sigma_i^-}{\eta_i^-} \quad \text{and} \quad \Psi_i^+ = \frac{\sigma_i^+}{\eta_i^+}.$$

We initialize the pseudocosts using strong branching: if $\eta_i^- = 0$ when we are choosing which variable to branch on at a node Q, we set

$$\Psi_i^- = \frac{\check{c}_Q - \check{c}_{Q_i^-}}{f^-_{Q,i}} \tag{15}$$

and similarly for Ψ_i^+.

In pseudocost branching, we estimate $\check{c}_Q - \check{c}_{Q_i^-}$ using the value $\Psi_i^- f^-_{Q,i}$ and we estimate $\check{c}_Q - \check{c}_{Q_i^+}$ using the value $\Psi_i^+ f^+_{Q,i}$. For example, the linear scoring rule with parameter $\mu \in [0, 1]$ using pseudocost branching is defined as

$$\texttt{score}(\mathcal{T}, Q, i) = (1 - \mu) \min \left\{ \Psi_i^- f^-_{Q,i}, \Psi_i^+ f^+_{Q,i} \right\} + \mu \max \left\{ \Psi_i^- f^-_{Q,i}, \Psi_i^+ f^+_{Q,i} \right\}.$$

Reliability branching [Achterberg et al. 2005] as a variation on pseudocost branching. Variable i's pseudocosts are said to be *unreliable* if $\min\{\eta_i^-, \eta_i^+\} < \eta_{\text{rel}}$, where $\eta_{\text{rel}} \in \mathbb{Z}$ is a tunable parameter. We set the pseudocosts of unreliable variables as in Equation (15). We refer the reader to Achterberg's thesis [Achterberg 2007] for guidance about how to tune the parameter η_{rel}.

B 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 3.24. *Let*

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

and $\texttt{cost}_\mu(Q)$ *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 exist a distribution* $\mathcal{D}$ *over MILP instances with* n *binary variables such that if* $\mu \in [0,1] \setminus (a,b)$*, then*

$$\mathbb{E}_{Q\sim\mathcal{D}}[\texttt{cost}_\mu(Q)] = \Omega\left(2^{(n-9)/4}\right)$$

and if $\mu \in (a,b)$*, then with probability 1,* $\texttt{cost}(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2) = O(1)$*. 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 3.25 and B.5. In Theorem 3.25, we prove that for all $\mu^* \in (\frac{1}{3}, \frac{2}{3})$, 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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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 B.5, 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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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 $\texttt{cost}(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2)$ is $\frac{1}{2}(O(1) + 2^{(n-5)/4}) \geq 2^{(n-9)/4}$. Meanwhile, if $\mu \in (a,b)$, then with probability 1,

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

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 $\texttt{score} = \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ for any $\mu \in [0,1]$, then tree $\mathcal{T}$ B&B builds using NSP and score equals the tree $\mathcal{T}'$ it builds using NSP' and score (see Theorem B.10). Thus, the theorem holds for any node-selection policy. □

Theorem 3.25. *Let*

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

and $\texttt{cost}_\mu(Q)$ *be the size of the tree produced by B&B. For all even* $n \geq 6$ *and all* $\mu^* \in (\frac{1}{3}, \frac{1}{2})$*, 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* $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ *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 $\texttt{score}_1$ and $\texttt{score}_2$ 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 $\boldsymbol{c}_2 = \gamma(0, 1.5, 3 - \frac{1}{2\mu^*})$. 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 $\boldsymbol{a}_1 = 2\sum_{i=1}^{n-3} \boldsymbol{e}_i$ 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

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

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

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

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

For every even $n \geq 6$, 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 B&B 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 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.

Lemma B.2. *Suppose $\mu < \mu^*$. Then, for any MILP $Q_{\gamma,n} \in \mathcal{F}_{n,\mu^*}$, $\mu\mathtt{score}_1 + (1-\mu)\mathtt{score}_2$ 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 B.2. 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 $\check{\boldsymbol{x}}_Q$, 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 $\boldsymbol{c} = (1,2,\ldots,n-3,0,1.5,3-\frac{1}{2\mu^*})$ is the vector defining the objective value of Q. Since $\mu^* > 1/3$, we know that $1.5 < 3-\frac{1}{2\mu^*}$, which means that $c[n-2] < c[n-1] < c[n]$. Since the goal is to maximize $\boldsymbol{c}\cdot\boldsymbol{x}$, the LP relaxation will set $x[n-2] = 0$, $x[n-1] = \frac{1}{2}$, 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 $\lfloor\frac{n-3}{2}\rfloor$ of the variables to 1, it can set one variable to $\frac{1}{2}$, and it has to set the rest of the variables to 0. Letting $i = \lceil\frac{n-3}{2}\rceil$, this means the LP relaxation will set the first $i-1$ variables $x[1],\cdots x[i-1]$ to zero, it will set $x[i] = \frac{1}{2}$, and it will set $x[i+1],\ldots,x[n-3]$ to 1. In other words,

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

For example, if $n = 8$, then $\check{\boldsymbol{x}}_Q = (0,0,\frac{1}{2},1,1,0,\frac{1}{2},1)$. (See Figure B.1(a).) Therefore, the only candidate variables to branch on are $x[n-1]$ and $x[i]$ where again, $i = \lceil(n-3)/2\rceil$.

To determine when variable B&B will branch on, we need to calculate $\check{\boldsymbol{x}}_{Q_i^-}$ 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

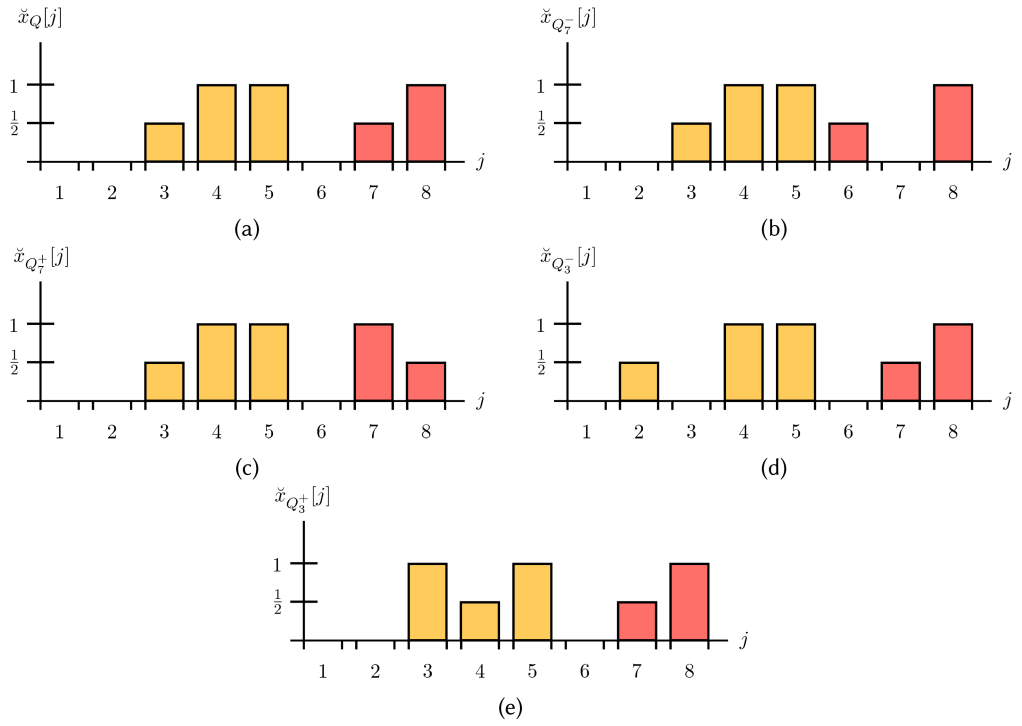


Fig. B.1. Illustrations to accompany the proof of Lemma B.2 when $n = 8$. For each j on the x-axis, the histogram gives the value of either $\check{x}_Q[j]$ (Figure B.1(a)), $\check{x}_{Q_7^-}[j]$ (Figure B.1(b)), $\check{x}_{Q_7^+}[j]$ (Figure B.1(c)), $\check{x}_{Q_3^-}[j]$ (Figure B.1(d)), or $\check{x}_{Q_3^+}[j]$ (Figure B.1(e)). In this case, $i = 3$.

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 $\check{\boldsymbol{x}}_{Q_{n-1}^-}$ and $\check{\boldsymbol{x}}_{Q_{n-1}^+}$. 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 B.1(b).) 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 B.1(c).) In other words, $\check{\boldsymbol{x}}_{Q_{n-1}^-} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{Q_{n-1}^+} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$.

The argument for $\check{\boldsymbol{x}}_{Q_i^-}$ and $\check{\boldsymbol{x}}_{Q_i^-}$ is similar. Recall that, in $\check{\boldsymbol{x}}_Q$, the solution to the LP relaxation of the original MIP Q, we have that $\check{x}_Q[i] = \frac{1}{2}$ since it is the median of the variables $x[1], \ldots, x[n-3]$. For all $j > i$, we have that $\check{x}_Q[j] = 1$ and for all $j < i$, we have that $\check{x}_Q[j] = 0$. 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 B.1(d).) In other words, $\check{\boldsymbol{x}}_{Q_i^-} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_i + \frac{1}{2}\boldsymbol{e}_{i-1}$. 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 $2\sum_{i=1}^{n-3} x[i] = n-3$. Therefore, it will set $x[i+1]$ to $\frac{1}{2}$ and $x[j]$ to 1 for all $j > i+1$. (See Figure B.1(e).) In other words, $\check{\boldsymbol{x}}_{Q_i^+} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{i+1}$.

Therefore,

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

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

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

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

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 B&B will explore is $Q_{n-1,n-2}^{-,+}$. The vector $\check{\boldsymbol{x}}_{Q_{n-1,n-2}^{-,+}}$ 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 $\mu\texttt{score}_1(Q_{n-1,n-2}^{-,+}, n) + (1-\mu)\texttt{score}_2(Q_{n-1,n-2}^{-,+}, n) = B$ whereas $\mu\texttt{score}_1(Q_{n-1,n-2}^{-,+}, i) + (1-\mu)\texttt{score}_2(Q_{n-1,n-2}^{-,+}, i) = \gamma/2$, as before. Therefore, B&B will branch on $x[n]$ and fathom both children.

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

on $x[n]$, $\check{\boldsymbol{x}}_{Q^{+,-}_{n-1,n}} = \check{\boldsymbol{x}}_{Q^{+}_{n-1}} - \frac{1}{2}\boldsymbol{e}_n + \frac{1}{2}\boldsymbol{e}_{n-2}$, so $\check{c}_{Q^{+,-}_{n-1,n}} = \check{c}_{Q^{+}_{n-1}} - \frac{\gamma}{2}(3 - \frac{1}{2\mu^*})$. Meanwhile, $Q^{+,+}_{n-1,n}$ is infeasible, so $\check{c}_{Q^{+}_{n-1}} - \check{c}_{Q^{+,+}_{n-1,n-2}} = B$. Therefore, $\mu\texttt{score}_1(Q^{+}_{n-1}, i) + (1-\mu)\texttt{score}_2(Q^{+}_{n-1}, i) = \gamma/2$ and $\mu\texttt{score}_1(Q^{+}_{n-1}, n) + (1-\mu)\texttt{score}_2(Q^{+}_{n-1}, n) = \frac{\mu\gamma}{2}(3 - \frac{1}{2\mu^*}) + (1-\mu)B > \gamma/2$. Therefore, $x[n]$ will be branched on next.

The next node B&B will explore is $Q^{+,-}_{n-1,n}$. The vector $\check{\boldsymbol{x}}_{Q^{+,-}_{n-1,n}}$ 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

$$\mu\texttt{score}_1\left(Q^{-,+}_{n-1,n}, n-2\right) + (1-\mu)\texttt{score}_2\left(Q^{-,+}_{n-1,n}, n-2\right) = B$$

whereas $\mu\texttt{score}_1(Q^{-,+}_{n-1,n-2}, i) + (1-\mu)\texttt{score}_2(Q^{-,+}_{n-1,n-2}, i) = \gamma/2$, as before. Therefore, B&B will branch on $x[n-2]$ and fathom both children.

At this point, all children have been fathomed, so B&B will terminate. □

Lemma B.3. *Suppose $\mu > \mu^*$. Then, for any MILP $Q_{\gamma,n} \in \mathcal{F}_{n,\mu^*}$, B&B with the scoring rule $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ will create a tree of depth at least $2^{(n-5)/4}$.*

Proof of Lemma B.3. To prove this lemma, we use induction to show that on any path from the root of the B&B 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, B&B will continue down every branch to depth $\lfloor (n-3)/2 \rfloor$, thus creating a tree with $2^{(n-4)/2}$ nodes.

Claim B.4. *On any path from the root of the B&B 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 B.4. 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 B&B 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 B.2, if $\mu > \mu^*$, then B&B 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 B&B 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 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, $\check{c}_{Q'} - \check{c}_{(Q')^{-}_{p_{i'+1}}} = \check{c}_{Q'} - (\check{c}_{Q'} - \gamma p_{i'+1}/2 + \gamma p_{i'}/2) = \frac{\gamma(p_{i'+1} - p_{i'})}{2}$. 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,

$\check{c}_{Q'} - \check{c}_{(Q')^+_{p_{i'+1}}} = \check{c}_{Q'} - (\check{c}_{Q'} + \gamma p_{i'+1}/2 - \gamma p_{i'+2}/2) = \frac{\gamma(p_{i'+2}-p_{i'+1})}{2}$. Suppose that $\check{c}_{Q'} - \check{c}_{(Q')^+_{p_{i'+1}}} > \check{c}_{Q'} - \check{c}_{(Q')^-_{p_{i'+1}}}$. Then

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

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

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

Meanwhile, as in the proof of Lemma B.2, $\mu\mathtt{score}_1(Q', n-1) + (1-\mu)\mathtt{score}_2(Q', n-1) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < \frac{\gamma}{2}$ so long as $\mu > \mu^*$. Thus, B&B will branch next on $x[p_{i'}]$. □

□

□

Theorem B.5. *Let*

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

For all even $n \geq 6$ and all $\mu^* \in (\frac{1}{3}, \frac{2}{3})$, 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 $\Omega(2^{(n-5)/4})$ 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 $\boldsymbol{c}_1 \in \mathbb{R}^{n-3}$ 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 $\boldsymbol{c}_2 = (1, 2, 3)$. Let $\boldsymbol{c} = \gamma(\boldsymbol{c}_1, \boldsymbol{c}_2) \in \mathbb{R}^n$ be the concatenation of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ multiplied with γ. For example, if $\gamma = 1$ and $n = 8$, then $\boldsymbol{c} = (0, 0, 1.5, 3 - \frac{1}{2\mu^*}, 3 - \frac{1}{2\mu^*}, 1, 2, 3)$ 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

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

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 B&B 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 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.

Lemma B.6. *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 B.6. 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:

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

For example, if $n = 8$, then $\check{\boldsymbol{x}}_Q = (0, 0, \frac{1}{2}, 1, 1, 0, \frac{1}{2}, 1)$. 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 $\check{\boldsymbol{x}}_{Q_i^-} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_i + \frac{1}{2}\boldsymbol{e}_{i-1}$ and $\check{\boldsymbol{x}}_{Q_i^+} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{i+1}$. Branching on $x[n-1]$, we have $\check{\boldsymbol{x}}_{Q_{n-1}^-} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{Q_{n-1}^+} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

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

This means that $\check{c}_Q - \check{c}_{Q_i^-} = 3\gamma/4$, $\check{c}_Q - \check{c}_{Q_i^+} = \frac{\gamma}{4}(3 - \frac{1}{\mu^*})$, and $\check{c}_Q - \check{c}_{Q_{n-1}^-} = \check{c}_Q - \check{c}_{Q_{n-1}^+} = \gamma/2$. Therefore, $\mu\texttt{score}_1(Q,i) + (1-\mu)\texttt{score}_2(Q,i) = \frac{\mu\gamma}{4}(3 - \frac{1}{\mu^*}) + \frac{3\gamma(1-\mu)}{4} = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$ 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 B&B will explore is Q_{n-1}^-. The vector $\check{\boldsymbol{x}}_{Q_{n-1}^-}$ has fractional values only in positions i and $n-2$. Branching on i, we again have that $\check{c}_{Q_{n-1}^-} - \check{c}_{Q_{n-1,i}^{-,-}} = 3\gamma/4$ and $\check{c}_{Q_{n-1}^-} - \check{c}_{Q_{n-1,i}^{-,+}} = \frac{\gamma}{4}(3 - \frac{1}{\mu^*})$. Branching on $n-2$, $Q_{n-1,n-2}^{-,-}$ is infeasible, so $\check{c}_{Q_{n-1}^-} - \check{c}_{Q_{n-1,n-2}^{-,-}}$ equals some large number $B \ge ||\boldsymbol{c}||_1$. Next, $\check{\boldsymbol{x}}_{Q_{n-1,n-2}^{-,+}} = \check{\boldsymbol{x}}_{Q_{n-1}^-} + \frac{1}{2}\boldsymbol{e}_{n-2} - \frac{1}{2}\boldsymbol{e}_n$, so $\check{c}_{Q_{n-1,n-2}^{-,+}} = \check{c}_{Q_{n-1}^-} - \gamma$. Therefore, $\mu\texttt{score}_1(Q_{n-1}^-, i) + (1-\mu)\texttt{score}_2(Q_{n-1}^-, i) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*}$ and

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

$$= B - \frac{\mu\gamma}{4\mu^*} + 3\gamma/4 - B$$
$$= \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*},$$

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 B&B will explore is $Q^{-,+}_{n-1,n-2}$. The vector $\check{\boldsymbol{x}}_{Q^{-,+}_{n-1,n-2}}$ 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 $\mu\texttt{score}_1(Q^{-,+}_{n-1,n-2}, n)+(1-\mu)\texttt{score}_2(Q^{-,+}_{n-1,n-2}, n) = B$ whereas $\mu\texttt{score}_1(Q^{-,+}_{n-1,n-2}, i)+(1-\mu)\texttt{score}_2(Q^{-,+}_{n-1,n-2}, i) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < B$. Therefore, B&B will branch on $x[n]$ and fathom both children.

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

The next node B&B will explore is $Q^{+,-}_{n-1,n}$. The vector $\check{\boldsymbol{x}}_{Q^{+,-}_{n-1,n}}$ 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 $\mu\texttt{score}_1(Q^{-,+}_{n-1,n}, n-2) + (1-\mu)\texttt{score}_2(Q^{-,+}_{n-1,n}, n-2) = B$ whereas $\mu\texttt{score}_1(Q^{-,+}_{n-1,n-2}, i) + (1-\mu)\texttt{score}_2(Q^{-,+}_{n-1,n-2}, i) = \frac{3\gamma}{4} - \frac{\mu\gamma}{4\mu^*} < B$, as before. Therefore, B&B will branch on $x[n-2]$ and fathom both children.

At this point, all children have been fathomed, so B&B will terminate. □

Lemma B.7. *Suppose $\mu < \mu^*$. Then for any MILP $Q_{\gamma,n} \in \mathcal{G}_{n,\mu^*}$, B&B with the scoring rule $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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 B.8. *Let j be an even number such that $2 \leq j \leq i-2$ and let $J = \{x[i-j/2], \ldots, x[i+j/2-1]\}$. 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 $n-3-(i+\frac{j}{2}-1) = 2i-1-(i+\frac{j}{2}-1) = i-\frac{j}{2}$ 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 $\check{\boldsymbol{x}}_{Q^-_{i+j/2}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i+j/2} + \frac{1}{2}\boldsymbol{e}_{i-j/2-1}$ and $\check{\boldsymbol{x}}_{Q^+_{i+j/2}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i+j/2} - \frac{1}{2}\boldsymbol{e}_{i+j/2+1}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{Q^-_{n-1}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{Q^+_{n-1}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{Q^-_{i+j/2}} &= \check{c}_Q - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)\\
\check{c}_{Q^+_{i+j/2}} &= \check{c}_Q\\
\check{c}_{Q^-_{n-1}} &= \check{c}_Q - 1 + \frac{1}{2} = \check{c}_Q - \frac{\gamma}{2}\\
\check{c}_{Q^+_{n-1}} &= \check{c}_Q + 1 - \frac{3}{2} = \check{c}_Q - \frac{\gamma}{2}
\end{aligned}$$

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

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

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 $n-3-(i+\frac{j}{2}) = 2i-1-(i+\frac{j}{2}) = i-\frac{j}{2}-1$ 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 $\check{\boldsymbol{x}}_{Q^-_{i-j/2}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i-j/2} + \frac{1}{2}\boldsymbol{e}_{i-j/2-1}$ and $\check{\boldsymbol{x}}_{Q^+_{i-j/2}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i-j/2} - \frac{1}{2}\boldsymbol{e}_{i+j/2+1}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{Q^-_{n-1}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{Q^+_{n-1}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{Q^-_{i-j/2}} &= \check{c}_Q\\
\check{c}_{Q^+_{i-j/2}} &= \check{c}_Q - \frac{\gamma}{2}\left(3 - \frac{1}{2\mu^*}\right)\\
\check{c}_{Q^-_{n-1}} &= \check{c}_Q - \gamma + \frac{\gamma}{2} = \check{c}_Q - \frac{\gamma}{2}\\
\check{c}_{Q^+_{n-1}} &= \check{c}_Q + \gamma - \frac{3\gamma}{2} = \check{c}_Q - \frac{\gamma}{2}
\end{aligned}$$

This means that $\check{c}_Q - \check{c}_{Q^-_{i-j/2}} = \frac{\gamma}{2}(3-\frac{1}{2\mu^*})$, $\check{c}_Q - \check{c}_{Q^+_{i-j/2}} = 0$, and $\check{c}_Q - \check{c}_{Q^-_{n-1}} = \check{c}_Q - \check{c}_{Q^+_{n-1}} = \frac{\gamma}{2}$ as in the previous case, so $x[i-j/2]$ will be branched on next. □

Claim B.9. *Suppose that J is the set of variables branched on along a path of depth $2 \le j \le i-2$ 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 $\check{\boldsymbol{x}}_{Q^-_{i-j'-1}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i-j'-1} + \frac{1}{2}\boldsymbol{e}_{i-j'-2}$ and $\check{\boldsymbol{x}}_{Q^+_{i-j'-1}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i-j'-1} - \frac{1}{2}\boldsymbol{e}_{i+j'+1}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{Q^-_{n-1}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{Q^+_{n-1}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

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

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

$$\begin{aligned}
\mu\mathtt{score}_1(Q, i-j'-1) + (1-\mu)\mathtt{score}_2(Q, i-j'-1) &= \frac{\gamma(1-\mu)}{2}\left(3 - \frac{1}{2\mu^*}\right) \\
&\ge \frac{\gamma\,(1-\mu^*)}{2}\left(3 - \frac{1}{2\mu^*}\right) \\
&\ge \frac{\gamma}{2}.
\end{aligned}$$

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 $\check{\boldsymbol{x}}_{Q^-_{i+j'+1}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{i+j'+1} + \frac{1}{2}\boldsymbol{e}_{i-j'-1}$ and $\check{\boldsymbol{x}}_{Q^+_{i+j'+1}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{i+j'+1} - \frac{1}{2}\boldsymbol{e}_{i+j'+2}$. Branching on

$x[n-1]$, we have that $\check{\boldsymbol{x}}_{Q^-_{n-1}} = \check{\boldsymbol{x}}_Q - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{Q^+_{n-1}} = \check{\boldsymbol{x}}_Q + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

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

This means that $\check{c}_Q - \check{c}_{Q^-_{i+j'+1}} = \frac{\gamma}{2}(3 - \frac{1}{2\mu^*})$, $\check{c}_Q - \check{c}_{Q^+_{i+j'+1}} = 0$, and $\check{c}_Q - \check{c}_{Q^-_{n-1}} = \check{c}_Q - \check{c}_{Q^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\mu\texttt{score}_1(Q, i+j'+1) + (1-\mu)\texttt{score}_2(Q, i+j'+1) = \frac{\gamma(1-\mu)}{2}(3 - \frac{1}{2\mu^*})$ and $\mu\texttt{score}_1(Q, n-1) + (1-\mu)\texttt{score}_2(Q, n-1) = \frac{\gamma}{2}$. As in the previous case, this means that $x[i+j'+1]$ will be branched on next. □

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 B.6 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 B.8, 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 B.8, 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 B.8, 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

$$\begin{aligned}&\{x[i-j'-1], x[i-j'], \ldots, x[i+j']\}\\ =\ &\{x[i-(j+1)/2], x[i-(j+1)/2+1], \ldots, x[i+(j+1)/2-1]\}\end{aligned}$$

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

$$\begin{aligned}&\{x[i-j'], x[i-j'+1], \ldots, x[i+j'], x[i+j'+1]\}\\ =\ &\{x[i-(j+1)/2+1], x[i-(j+1)/2+1], \ldots, x[i+(j+1)/2]\}\end{aligned}$$

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. □

□

Theorem B.10. *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 $\mathcal{T}'$. □

Lemma B.11 (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 3.3. *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 $\mu \in I_j$.*[13]

[13]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 `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 $x[i]$, $\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}[j], Q, i)$.

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 B&B 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

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

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 $k = \text{argmax}_{\ell \in [n]} \{\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2\}$. 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 $k = \text{argmax}_\ell\{\mu\texttt{score}_1(\mathcal{T}_\mu, Q, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_\mu, Q, \ell)\}$, or equivalently, so long as $k = \text{argmax}_\ell\{\mu\texttt{score}_1(\mathcal{T}_Q, Q, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_Q, Q, \ell)\}$. 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.

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 $\mathcal{T}[i] = \mathcal{T}'[i]$. □

Theorem B.12. *Let* $\texttt{score}_1$ *and* $\texttt{score}_2$ *be two path-wise scoring rules and let* C *be the set of functions* $C = \{\texttt{cost}_\mu : \mu \in [0, 1]\}$ *mapping MILPs to* $\mathbb{R}$. *Then* $\text{Pdim}(C) = O(n^2)$.

Proof. Suppose that $\text{Pdim}(C) = m$ and let $\mathcal{S} = \{Q_1, \ldots, Q_m\}$ be a shatterable set of MILPs. We know there exists a set of targets $r_1, \ldots, r_m \in \mathbb{R}$ that witness the shattering of $\mathcal{S}$ by 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 $\texttt{cost}_{\mu_{\mathcal{S}'}}(Q_i) \leq r_i$. Otherwise $\texttt{cost}_{\mu_{\mathcal{S}'}}(Q_i) > r_i$. 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 $\text{Pdim}(C) = m = O(\log(2^{n(n-1)/2}n^n)) = O(n^2)$ (see Lemma B.11 in Appendix B).

To prove that $|M| \leq m2^{n(n-1)/2}n^n + 1$, we rely on Lemma 3.2, 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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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 $Q_i \in \mathcal{S}$, $\texttt{cost}_\mu(Q_i)$ is the same 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. □

Lemma 3.10. *Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules, let $\texttt{cost}_\mu$ be a tree-constant cost function for all $\boldsymbol{\mu} \in [0, 1]^d$ and let Q be an arbitrary MILP over n integer variables. Suppose we limit B&B to producing search trees of size κ. There is a set $\mathcal{H}$ of at most $\kappa n^{\kappa+2}$ 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 $\sum_{j=1}^d \mu_j \texttt{score}_j$ is invariant across all $(\mu_1, \ldots, \mu_d) \in R$. In other words, $\texttt{cost}^*_Q(\boldsymbol{\mu})$ is constant for all $\boldsymbol{\mu} \in R$.*

Proof. The proof has two steps. In Claim B.13, we show that there are at most n^κ different search trees that Algorithm 3 might produce for the MILP Q as we vary the mixing parameter vector $(\mu_1, \ldots, \mu_d)$. In Claim B.14, 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 κn^2 halfspaces. These facts together prove the lemma.

Claim B.13. *There are only n^κ different search trees that can be achieved by varying the parameter vector $(\mu_1, \ldots, \mu_d)$.*

Proof of Claim B.13. Fix any d mixing parameters $(\mu_1, \ldots, \mu_d)$ and let $v_1, \ldots, v_\kappa \in [n]$ be the sequence of branching variables chosen by B&B run with scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$, 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 3 with scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ produces a tree of size $k < \kappa$, 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 κ). 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 B&B with scoring rules of the form $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ is at most n^κ, since there are only n^κ distinct sequences of κ variables $v_1, \ldots, v_\kappa \in [n]$.

Let $(\mu_1, \ldots, \mu_d)$ and $(\mu'_1, \ldots, \mu'_d)$ be two sets of mixing parameters, and suppose running B&B with $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ and $\mu'_1\texttt{score}_1 + \cdots + \mu'_d\texttt{score}_d$ both results in the sequence of branching variable decisions being $v_1, \ldots, v_\kappa$. We prove that the resulting search trees are identical by induction on the iterations of the algorithm, where an iteration corresponds to Lines 4 through 22 of Algorithm 1. 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 $v_1, \ldots, v_\kappa$ is shared. Since the two trees are identical, we will branch using the same constraint $v_t \le a$ for some integer $a \in \mathbb{Z}$ in both trees. 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 B&B 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. □

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

Claim B.14. *For a given search tree $\mathcal{T}$, there are at most κn^2 halfspaces such that Algorithm 3 using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ builds the tree $\mathcal{T}$ if and only if $(\mu_1, \ldots, \mu_d)$ lies in the intersection of those halfspaces.*

Proof of Claim B.14. Let $v_1, \ldots, v_\kappa$ 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 B&B. Let $\mathcal{T}_t$ be the state of B&B after t iterations.

Induction Hypothesis. For a given index $t \in [\kappa]$, there are at most tn^2 halfspaces such that B&B 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 $(\mu_1, \ldots, \mu_d)$ 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 $t < \kappa$ 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 B&B 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 $(\mu_1, \ldots, \mu_d)$ lies in the intersection of those halfspaces. Let Q' be the IP contained in the next node that B&B will branch on given $\mathcal{T}_t$. We know that B&B will choose to branch on variable v_{t+1} at this node if and only if

$$\begin{aligned}&\mu_1\texttt{score}_1(\mathcal{T}_t, Q', v_{t+1}) + \cdots + \mu_d\texttt{score}_d(\mathcal{T}_t, Q', v_{t+1})\\ &> \max_{v' \neq v_{t+1}} \{\mu_1\texttt{score}_1(\mathcal{T}_t, Q', v') + \cdots + \mu_d\texttt{score}_d(\mathcal{T}_t, Q', v')\}\,.\end{aligned}$$

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

□

Theorem 3.11. *Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules and let $\texttt{cost}_{\boldsymbol{\mu}}$ be a tree-constant cost function for all $\boldsymbol{\mu} \in [0,1]^d$. Let C be the set of functions $C = \{\texttt{cost}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in [0,1]^d\}$ mapping MILPs to $[0, H]$. For any set $\mathcal{S} = \{Q_1, \ldots, Q_m\}$ of MILPs,*

$$\widehat{\mathcal{R}}_{\mathcal{S}}(C) = O\left(H\sqrt{\frac{d(\ln(m\kappa) + \kappa \ln n)}{m}}\right).$$

Proof. We will again use Massart's lemma (Lemma 3.7) to prove this lemma. Let $A \subseteq [0,H]^m$ be the following set of vectors:

$$A = \left\{ \begin{pmatrix} \texttt{cost}_{\boldsymbol{\mu}}(Q_1) \\ \vdots \\ \texttt{cost}_{\boldsymbol{\mu}}(Q_m) \end{pmatrix} : \boldsymbol{\mu} \in [0,1]^d \right\}.$$

By definition, this means that

$$\widehat{\mathcal{R}}_{\mathcal{S}}(C) = \frac{1}{m}\mathbb{E}_{\boldsymbol{\sigma}\sim\{-1,1\}^m}\left[\sup_{\boldsymbol{a}\in A}\sum_{i=1}^{m}\sigma_i a[i]\right]$$

Moreover, by definition of the dual class,

$$A = \left\{ \begin{pmatrix} \texttt{cost}^*_{Q_1}(\boldsymbol{\mu}) \\ \vdots \\ \texttt{cost}^*_{Q_m}(\boldsymbol{\mu}) \end{pmatrix} : \boldsymbol{\mu} \in [0,1]^d \right\}.$$

By Lemma 3.10, for each dual function Q_i there is a set $\mathcal{H}_i$ of at most $\kappa n^{\kappa+2}$ hyperplanes such that for any connected component R of $[0,1]^d \setminus \mathcal{H}$, $\texttt{cost}^*_{Q_i}(\boldsymbol{\mu})$ is constant for all $\boldsymbol{\mu} \in R$. All of the $m\kappa n^{\kappa+2}$ hyperplanes in $\mathcal{H}_1 \cup \cdots \cup \mathcal{H}_m$ split $[0,1]^d$ into $O(d(m\kappa n^{\kappa+2})^d)$ regions where in any one region R, the entire vector $\texttt{cost}^*_{Q_1}(\boldsymbol{\mu}), \ldots, \texttt{cost}^*_{Q_m}(\boldsymbol{\mu})$ is fixed across all $\boldsymbol{\mu} \in [0,1]^d$. This means that $|A| = O(d(m\kappa n^{\kappa+2})^d)$. The lemma statement therefore follows from Massart's lemma. □

Theorem B.15. *Let* $\texttt{score}_1, \ldots, \texttt{score}_d$ *be* d *arbitrary scoring rules and let* $\texttt{cost}_{\boldsymbol{\mu}}$ *be a tree-constant cost function for all* $\boldsymbol{\mu} \in [0,1]^d$. *Let* C *be the set of functions* $C = \{\texttt{cost}_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in [0,1]^d\}$ *mapping MILPs to* $\mathbb{R}$. *Then* $\text{Pdim}(C) = O(d\kappa \log n + d \log d)$.

Proof. Suppose that $\text{Pdim}(C) = m$ 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 C. This means that for every $\mathcal{S}' \subseteq \mathcal{S}$, there exists a parameter vector $(\mu_{1,\mathcal{S}'}, \ldots, \mu_{d,\mathcal{S}'})$ such that if $Q_i \in \mathcal{S}'$, then $\texttt{cost}\left(Q_i, \mu_{1,\mathcal{S}'}\texttt{score}_1 + \cdots + \mu_{d,\mathcal{S}'}\texttt{score}_d\right) \leq r_i$. Otherwise

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

Let $M = \left\{(\mu_{1,\mathcal{S}'}, \ldots, \mu_{d,\mathcal{S}'}) : \mathcal{S}' \subseteq \mathcal{S}\right\}$. We will prove that $|M| = O(d(m\kappa n^{\kappa+2})^d)$, and since $2^m = |M|$, this means that $\text{Pdim}(C) = m = O(d\kappa \log n + d \log d)$ (see Lemma B.11 in Appendix B).

To prove our bound on $|M|$, we rely on Lemma 3.10, which tells us that for any problem instance Q, there is a set $\mathcal{H}$ of at most $T \leq \kappa n^{\kappa+2}$ 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 $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ is invariant across all $(\mu_1, \ldots, \mu_d) \in R$. If we merge all T hyperplanes for all samples in $\mathcal{S}$, we are left with a set $\mathcal{H}'$ of $T' \leq m\kappa n^{\kappa+2}$ hyperplanes where for any connected component R of $[0,1]^d \setminus \mathcal{H}'$ and any $Q_i \in \mathcal{S}$, the search tree B&B builds using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ given as input Q_i is invariant across all $(\mu_1, \ldots, \mu_d) \in R$. Therefore, at most one element of M can come from each connected component, of which there are $O(d|\mathcal{H}'|^d) = O(d(m\kappa n^{\kappa+2})^d)$. Therefore, $|M| = O(d(m\kappa n^{\kappa+2})^d)$, as claimed. □

B.1 Proofs from Section 3.3

Theorem 3.17. *Let* $\mathcal{F} = \{f_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ *and* $\mathcal{G} = \{g_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\}$ *consist of functions mapping* $\mathcal{Z}$ *to* $[0,H]$. *For any* $\mathcal{S} \subseteq \mathcal{Z}$, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) \leq \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{G}) + \frac{1}{|\mathcal{S}|}\sum_{z\in\mathcal{S}}\left\|f^*_z - g^*_z\right\|_\infty$.

Proof. Let $\mathcal{S} = \{z_1, \ldots, z_m\}$ be an arbitrary subset of $\mathcal{Z}$. Fix an arbitrary vector $\boldsymbol{\mu} \in \mathcal{R}$ and index $i \in [m]$. Suppose that $\sigma_i = 1$. Since $f^*_{z_i}(\boldsymbol{\mu}) \le g^*_{z_i}(\boldsymbol{\mu}) + \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty$, we have that

$$\sigma_i f^*_{z_i}(\boldsymbol{\mu}) \le \sigma_i g^*_{z_i}(\boldsymbol{\mu}) + \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty. \tag{16}$$

Meanwhile, suppose $\sigma_i = -1$. Since $f^*_{z_i}(\boldsymbol{\mu}) \ge g^*_{z_i}(\boldsymbol{\mu}) - \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty$, we have that

$$\sigma_i f^*_{z_i}(\boldsymbol{\mu}) = -f^*_{z_i}(\boldsymbol{\mu}) \le -g^*_{z_i}(\boldsymbol{\mu}) + \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty = \sigma_i g^*_{z_i}(\boldsymbol{\mu}) + \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty. \tag{17}$$

Combining Equations (16) and (17), we have that

$$\sup_{\boldsymbol{\mu}\in\mathcal{R}} \sum_{i=1}^m \sigma_i g_{\boldsymbol{\mu}}(z_i) \ge \sum_{i=1}^m \sigma_i g^*_{z_i}(\boldsymbol{\mu}) \ge \sum_{i=1}^m \sigma_i f^*_{z_i}(\boldsymbol{\mu}) - \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty. \tag{18}$$

By definition of the supremum, Equation (18) implies that for every $\boldsymbol{\sigma} \in \{-1, 1\}^m$,

$$\sup_{\boldsymbol{\mu}\in\mathcal{R}} \sum_{i=1}^m \sigma_i g_{\boldsymbol{\mu}}(z_i) \ge \sup_{\boldsymbol{\mu}\in\mathcal{R}} \sum_{i=1}^m \sigma_i f_{\boldsymbol{\mu}}(z_i) - \sum_{i=1}^m \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty.$$

Therefore

$$\mathbb{E}_{\boldsymbol{\sigma}\sim\{-1,1\}^m}\left[\sup_{\boldsymbol{\mu}\in\mathcal{R}} \sum_{i=1}^m \sigma_i g_{\boldsymbol{\mu}}(z_i)\right] \ge \mathbb{E}_{\boldsymbol{\sigma}\sim\{-1,1\}^m}\left[\sup_{\boldsymbol{\mu}\in\mathcal{R}} \sum_{i=1}^m \sigma_i f_{\boldsymbol{\mu}}(z_i)\right] - \sum_{i=1}^m \left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty,$$

so the lemma statement holds. □

Algorithm for Finding Approximating Function. We now provide a **dynamic programming (DP)** algorithm (Algorithm 2) which applies in the common case where $\mathcal{R} = \mathbb{R}$ and the dual functions f^*_z are piecewise constant with a large number of pieces. Given an integer k, the algorithm returns a piecewise-constant function g^*_z with at most k pieces such that $\left\|f^*_z - g^*_z\right\|_\infty$ is minimized. Letting t be the number of pieces in the piecewise decomposition of f^*_z, the DP algorithm runs in $O\left(kt^2\right)$ time.

The algorithm takes as input the partition $[a_1, a_2), \ldots, [a_t, a_{t+1})$ of the parameter space $\mathcal{R}$ and values $c_1, \ldots, c_t$ such that for any interval $[a_i, a_{i+1})$, $f^*_z(\mu) = c_i$ for all $\mu \in [a_i, a_{i+1})$. The algorithm begins by calculating upper and lower bounds on the value of the function f^*_z across various subsets of its domain. In particular, for each $i, i' \in [t]$ such that $i \le i'$, the algorithm calculates the lower bound $\ell_{i,i'} = \min\{c_i, c_{i+1}, \ldots, c_{i'}\}$ and the upper bound $u_{i,i'} = \max\{c_i, c_{i+1}, \ldots, c_{i'}\}$. Algorithm 2 performs these calculations in $O(t^2)$ time.

Next, for each $i \in [t]$ and $j \in [k]$, the algorithm calculates a value $C(i, j)$ which equals the smallest ℓ_∞ norm between any piecewise constant function with j pieces and the function f^*_z when restricted to the interval $[a_1, a_{i+1})$. Since $\mathcal{R} = [a_1, a_{t+1})$, we have that $C(t, k)$—the value our algorithm returns—equals $\min_{g\in\mathcal{G}_k} \left\|f^*_z - g\right\|_\infty$, as claimed. For all $i \in [t]$, $C(i, 1) = \frac{u_{1,i} - \ell_{1,i}}{2}$ and for all $j \ge 2$,

$$C(i, j) = \min\left\{C(i, 1), \min_{i'\in[i-1]}\left\{C(i', j-1) + \frac{u_{i'+1,i} - \ell_{i'+1,i}}{2}\right\}\right\}.$$

Algorithm 2 performs these calculations in $O(kt^2)$ time.

Corollary 3.18. *With probability* $1 - \delta$ *over the draw of the set* $\mathcal{S} \sim \mathcal{D}^m$, *for all* $\boldsymbol{\mu} \in \mathcal{R}$ *and all* $j \ge 1$,

$$\left|\frac{1}{m}\sum_{z\in\mathcal{S}} f_{\boldsymbol{\mu}}(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[f_{\boldsymbol{\mu}}(z)\right]\right| = O\left(\frac{1}{m}\sum_{z\in\mathcal{S}} \left\|f^*_z - g^*_{j,z}\right\|_\infty + \widehat{\mathcal{R}}_{\mathcal{S}}\left(\mathcal{G}_j\right) + H\sqrt{\frac{1}{m}\ln\frac{j}{\delta}}\right). \tag{8}$$

ALGORITHM 2: Piecewise-constant function fitting via dynamic programming

Input: Partition $[a_1, a_2), \ldots, [a_t, a_{t+1})$ of $\mathcal{R}$, values $c_1, \ldots, c_t$, and desired number of pieces $k \in \mathbb{N}$.

for $i \in [t]$ **do**
  Set $u_{i,i} = c_i$ and $\ell_{i,i} = c_i$.
  for $i' \in \{i+1, \ldots, t\}$ **do**
    if $c_{i'} < \ell_{i,i'-1}$ **then**
      Set $\ell_{i,i'} = c_{i'}$ and $u_{i,i'} = u_{i,i'-1}$.
    else if $c_{i'} > u_{i,i'-1}$ **then**
      Set $\ell_{i,i'} = \ell_{i,i'-1}$ and $u_{i,i'} = c_{i'}$.
    else
      Set $\ell_{i,i'} = \ell_{i,i'-1}$ and $u_{i,i'} = u_{i,i'-1}$.
  end
end
for $i \in [t]$ **do**
  Set $C(i,1) = \frac{u_{1,i} - \ell_{1,i}}{2}$.
end
for $j \in \{2, \ldots, k\}$ **do**
  for $i \in [t]$ **do**
    Set $C(i,j) = \min\left\{C(i,1), \min_{i' \in [i-1]}\left\{C(i', j-1) + \frac{u_{i'+1,i} - \ell_{i'+1,i}}{2}\right\}\right\}$.
  end
end

Proof. We will prove that with probability at least $1-\delta$ over the draw of the training set $\mathcal{S} = \{z_1, \ldots, z_m\} \sim \mathcal{D}^m$, for all parameter vectors $\boldsymbol{\mu} \in \mathcal{R}$ and all $j \in \mathbb{N}$,

$$\left|\frac{1}{m}\sum_{z\in\mathcal{S}} f_{\boldsymbol{\mu}}(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[f_{\boldsymbol{\mu}}(z)\right]\right| \le \frac{2}{m}\sum_{i=1}^{m}\left\|f^*_{z_i} - g^*_{j,z_i}\right\|_\infty + 2\widehat{\mathcal{R}}\left(\mathcal{G}_j\right) + 3\sqrt{\frac{1}{2m}\ln\frac{(\pi j)^2}{3\delta}}.$$

For each integer $j \ge 1$, let $\delta_j = \frac{6\delta}{(\pi j)^2}$. From Theorems 3.6 and 3.17, we know that with probability at least $1-\delta_j$ over the draw of the training set $\mathcal{S} =\sim \mathcal{D}^m$, for all parameter vectors $\boldsymbol{\mu} \in \mathcal{R}$,

$$\begin{aligned}\left|\frac{1}{m}\sum_{z\in\mathcal{S}} f_{\boldsymbol{\mu}}(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[f_{\boldsymbol{\mu}}(z)\right]\right| &\le \frac{2}{m}\sum_{i=1}^{m}\left\|f^*_{z_i} - g^*_{j,z_i}\right\|_\infty + 2\widehat{\mathcal{R}}\left(\mathcal{G}_j\right) + 3\sqrt{\frac{1}{2m}\ln\frac{2}{\delta_j}} \\ &= \frac{2}{m}\sum_{i=1}^{m}\left\|f^*_{z_i} - g^*_{j,z_i}\right\|_\infty + 2\widehat{\mathcal{R}}\left(\mathcal{G}_j\right) + 3\sqrt{\frac{1}{2m}\ln\frac{(\pi j)^2}{3\delta}}.\end{aligned}$$

Since $\sum_{i=1}^{\infty}\delta_j = \delta$, the corollary follows from a union bound over all $j \ge 1$. □

Corollary B.16. *Let $\mathcal{F} = \{f_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\} \subseteq [0,1]^{\mathcal{Z}}$ be a set of functions mapping $\mathcal{Z}$ to $[0,1]$. Let $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3, \ldots$ be a countable sequence of function classes, where for each $j \in \mathbb{N}$, $\mathcal{G}_j = \{g_{j,\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\} \subseteq [0,1]^{\mathcal{Z}}$ is a set of functions mapping $\mathcal{Z}$ to $[0,1]$, parameterized by vectors $\boldsymbol{\mu} \in \mathcal{R}$. With probability at least $1-\delta$ over the draw of the training set $\mathcal{S} \sim \mathcal{D}^m$, for all parameter vectors $\boldsymbol{\mu} \in \mathcal{R}$ and all $j \in \mathbb{N}$,*

$$\left|\frac{1}{m}\sum_{z\in\mathcal{S}} f_{\boldsymbol{\mu}}(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[f_{\boldsymbol{\mu}}(z)\right]\right| \le 2\widehat{\mathcal{R}}\left(\mathcal{G}_j\right) + 2\mathbb{E}_{z\sim\mathcal{D}}\left[\left\|f^*_z - g^*_{j,z}\right\|_\infty\right] + \sqrt{\frac{2}{m}\ln\frac{2(\pi j)^2}{3\delta}}.$$

Proof. From Theorem 3.17, we know that for every integer $j \geq 1$,

$$\mathbb{E}_{\mathcal{S}' \sim \mathcal{D}^m}\left[\widehat{\mathcal{R}}_{\mathcal{S}'}(\mathcal{F})\right] \leq \mathbb{E}_{\mathcal{S}' \sim \mathcal{D}^m}\left[\widehat{\mathcal{R}}_{\mathcal{S}'}\left(\mathcal{G}_j\right)\right] + \mathbb{E}_{z \sim \mathcal{D}}\left[\left\|f_z^* - g_{j,z}^*\right\|_\infty\right].$$

For each integer $j \geq 1$, let $\delta_j = \frac{6\delta}{(\pi j)^2}$. From Theorem 3.6 and a Hoeffding bound, we know that with probability at least $1 - \delta_j$ over the draw of the training set $\mathcal{S} = \{z_1, \ldots, z_m\} \sim \mathcal{D}^m$, for all parameter vectors $\boldsymbol{\mu} \in \mathcal{R}$,

$$\begin{aligned}\left|\frac{1}{m}\sum_{z \in \mathcal{S}} f_{\boldsymbol{\mu}}(z) - \mathbb{E}_{z \sim \mathcal{D}}\left[f_{\boldsymbol{\mu}}(z)\right]\right| &\leq 2\widehat{\mathcal{R}}\left(\mathcal{G}_j\right) + 2\mathbb{E}_{z \sim \mathcal{D}}\left[\left\|f_z^* - g_{j,z}^*\right\|_\infty\right] + \sqrt{\frac{2}{m}\ln\frac{4}{\delta_j}} \\ &= 2\widehat{\mathcal{R}}\left(\mathcal{G}_j\right) + 2\mathbb{E}_{z \sim \mathcal{D}}\left[\left\|f_z^* - g_{j,z}^*\right\|_\infty\right] + \sqrt{\frac{2}{m}\ln\frac{2(\pi j)^2}{3\delta}}.\end{aligned}$$

Since $\sum_{j=1}^{\infty} \delta_j = \delta$, the corollary follows from a union bound over all $j \geq 1$. □

Theorem B.17 (Hölder's Inequality). *Let p_0 and p_1 be two values in $[1, \infty]$ such that $\frac{1}{p_0} + \frac{1}{p_1} = 1$. Then for all functions u and w, $\|uw\|_1 \leq \|u\|_{p_0} \|w\|_{p_1}$.*

Theorem B.18 (Interpolation). *Let p and q be two values in $(0, \infty]$ and let θ be a value in $(0, 1)$. Let p_θ be defined such that $\frac{1}{p_\theta} = \frac{\theta}{p_1} + \frac{1-\theta}{p_0}$. Then, for all functions u, $\|f\|_{p_\theta} \leq \|f\|_{p_1}^{\theta} \|f\|_{p_0}^{1-\theta}$.*

Lemma B.19. *For any $\gamma \in (0, \frac{1}{4})$ and $p \in [1, \infty)$, let $\mathcal{F}$ and $\mathcal{G}$ be the function classes defined in Theorem 3.20. The dual class $\mathcal{G}^*$ (γ, p)-approximates the dual class $\mathcal{F}^*$.*

Proof. For ease of notation, let $t = \gamma^p$, $a = \frac{1}{2\gamma^p}$, $\mathcal{R} = (0, t]$, and $\mathcal{Z} = [\frac{1}{2\gamma^p}, \infty)$. Throughout this proof, we will use the following inequality:

$$\left\|f_z^* - g_z^*\right\|_2 = \sqrt{\int_0^t (f_z^*(\mu) - g_z^*(\mu))^2 d\mu} = \sqrt{\int_0^t \left(\frac{1}{2}\cos(\mu z)\right)^2 d\mu} = \frac{1}{4}\sqrt{2t + \frac{\sin(2tz)}{z}} \leq \frac{1}{4}\sqrt{2t + \frac{1}{z}}. \tag{19}$$

First, suppose $p = 2$. Since $t = \gamma^2$ and $\frac{1}{z} \leq 2\gamma^2$, Equation (19) implies that $\left\|f_z^* - g_z^*\right\|_2 \leq \frac{1}{4}\sqrt{4\gamma^2} < \gamma$.

Next, suppose $p < 2$. We know that

$$\left\|\left(f_z^* - g_z^*\right)^p\right\|_1 = \int_0^t \left|\left(f_z^*(\mu) - g_z^*(\mu)\right)^p\right| d\mu = \int_0^t \left|f_z^*(\mu) - g_z^*(\mu)\right|^p d\mu = \left\|f_z^* - g_z^*\right\|_p^p. \tag{20}$$

From Equation (20) and Hölder's inequality (Theorem B.17) with $u = \left(f_z^* - g_z^*\right)^p$, w the constant function $w : \mu \mapsto 1$, $p_0 = \frac{2}{p}$, and $p_1 = \frac{2}{2-p}$, we have that

$$\begin{aligned}\left\|f_z^* - g_z^*\right\|_p^p &= \left\|\left(f_z^* - g_z^*\right)^p\right\|_1 \\ &\leq \|w\|_{\frac{2}{2-p}} \left\|\left(f_z^* - g_z^*\right)^p\right\|_{\frac{2}{p}} \\ &= \left(\int_0^t d\mu\right)^{\frac{2-p}{2}} \left\|\left(f_z^* - g_z^*\right)^p\right\|_{\frac{2}{p}} \\ &= t^{\frac{2-p}{2}} \left\|\left(f_z^* - g_z^*\right)^p\right\|_{\frac{2}{p}} \\ &= t^{\frac{2-p}{2}} \left(\int_0^t \left(f_z^*(\mu) - g_z^*(\mu)\right)^2 d\mu\right)^{\frac{p}{2}} \\ &= t^{\frac{2-p}{2}} \left\|f_z^* - g_z^*\right\|_2^p.\end{aligned}$$

Therefore,

$$\begin{aligned}
\left\|f_z^* - g_z^*\right\|_p &\le t^{\frac{1}{p}-\frac{1}{2}} \left\|f_z^* - g_z^*\right\|_2 \\
&\le \frac{t^{\frac{1}{p}-\frac{1}{2}}}{4}\sqrt{2t + \frac{1}{z}} && \text{(Equation (19))} \\
&= \frac{t^{\frac{1}{p}}}{4}\sqrt{2 + \frac{1}{zt}} \\
&= \frac{\gamma}{4}\sqrt{2 + \frac{1}{z\gamma^p}} && \left(t = \gamma^p\right) \\
&< \gamma, && \left(z \ge \frac{1}{2\gamma^p}\right)
\end{aligned}$$

Finally, suppose $p > 2$. Let $\theta = 1 - \frac{2}{p}$, $p_0 = 2$, and $p_1 = \infty$. By Theorem B.18,

$$\begin{aligned}
\left\|f_z^* - g_z^*\right\|_p &\le \left\|f_z^* - g_z^*\right\|_2^{1-\theta} \\
&= \left\|f_z^* - g_z^*\right\|_2^{\frac{2}{p}} \\
&\le \sqrt[p]{\frac{t}{8} + \frac{1}{16z}} && \text{(Equation (19))} \\
&= \sqrt[p]{\frac{\gamma^p}{8} + \frac{1}{16z}} && \left(t = \gamma^p\right) \\
&\le \sqrt[p]{\frac{\gamma^p}{4}} && \left(z \ge \frac{1}{2\gamma^p}\right) \\
&< \gamma.
\end{aligned}$$

Therefore, for all $p \in [1, \infty)$ and all $z \in \mathcal{Z}$, $\left\|f_z^* - g_z^*\right\|_p \le \gamma$, so the dual class $\mathcal{G}^*$ (γ, p)-approximates the dual class $\mathcal{F}^*$. □

In the following lemma, we denote absolute loss using the notation $\ell(z, y, f) = |f(z) - y|$. Given a set of samples $\mathcal{S} = \{(z_1, y_1), \ldots, (z_m, y_m)\} \subseteq \mathcal{Z} \times [0, 1]$, we use the standard notation $L_{\mathcal{S}}(f) = \frac{1}{m}\sum_{i=1}^m |f(z_i) - y_i|$ to denote the average empirical loss of a function $f : \mathcal{Z} \to [0, 1]$ and $L_{\mathcal{D}}(f) = \mathbb{E}_{(z,y)\sim\mathcal{D}}[|f(z) - y|]$ to denote the expected loss of f. The absolute loss function can be naturally incorporated into the definition of Rademacher complexity: $\widehat{\mathcal{R}}(\ell \circ \mathcal{F}) = \frac{1}{m}\mathbb{E}_{\sigma\sim\{-1,1\}^m}[\sup_{f\in\mathcal{F}}\sum_{i=1}^m \sigma_i |f(z_i) - y_i|]$. The *worst-case empirical Rademacher complexity* of a class $\mathcal{F}$ is defined as $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) = \sup_{\mathcal{S}:|\mathcal{S}|=m} \widehat{\mathcal{R}}(\ell \circ \mathcal{F})$.

Lemma B.20. *For any $\gamma \in (0, \frac{1}{4})$ and $p \in [1, \infty)$, let $\mathcal{F} = \{f_\mu \mid \mu \in (0, \gamma^p]\}$ be a class of functions with domain $[\frac{1}{2\gamma^p}, \infty)$ such that for all $\mu \in (0, \gamma^p]$ and $z \in [\frac{1}{2\gamma^p}, \infty)$, $f_\mu(z) = \frac{1}{2}(1 + \cos(\mu z))$. For every $m \ge 1$, $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) = \frac{1}{2}$.*

Proof. This proof is similar to the proof that the VC-dimension of the function class

$$\{z \mapsto \operatorname{sign}(\sin(\mu z)) \mid \mu \in \mathbb{R}\} \subseteq \{-1, 1\}^{\mathbb{R}}$$

is infinite (see, e.g., Lemma 7.2 in the textbook by Anthony and Bartlett [2009]). To prove this lemma, we will show that for every $c \in (0, 1/2)$, $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) \ge c$ (Claim B.21). We also show that $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) \le \frac{1}{2}$ (Claim B.22). Therefore, the lemma statement follows.

Claim B.21. *For every $c \in (0, 1/2)$, $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) \geq c$.*

Proof of Claim B.21. Let m be an arbitrary positive integer. We begin by defining several variables that we will use throughout this proof. Let $\mathcal{R} = (0, \gamma^p]$ and let α be any positive power of $\frac{1}{2}$ smaller than $\min\{\frac{1}{2\pi+1}, \frac{\arccos(2c)}{\pi+\arccos(2c)}\}$. Since $2c \in (0,1)$, $\frac{\arccos(2c)}{\pi+\arccos(2c)}$ is well defined. Also, since $\alpha \leq \frac{\arccos(2c)}{\pi+\arccos(2c)}$, we have that $\frac{\pi\alpha}{1-\alpha} \leq \arccos(2c) < \frac{\pi}{2}$. Finally, since the function cos is decreasing on the interval $[0, \pi/2]$, we have that $\frac{1}{2}\cos\frac{\pi\alpha}{1-\alpha} \geq c$. Let $z_i = \frac{\alpha^{-i}}{2\gamma^p}$ and $y_i = 0$ for $i \in [m]$. Since $\alpha < 1$, we have that $z_i \geq \frac{1}{2\gamma^p}$, so each z_i is an element of the domain $[\frac{1}{2\gamma^p}, \infty)$ of the functions in $\mathcal{F}$.

We will show that for every assignment of the variables $\sigma_1, \ldots, \sigma_m \in \{-1, 1\}$, there exists a parameter $\mu_0 \in (0, \gamma^p]$ such that

$$\frac{1}{m}\sup_{\mu\in(0,\gamma^p]}\sum_{i=1}^{m}\sigma_i f_\mu(z_i) \geq \frac{1}{m}\sum_{i=1}^{m}\sigma_i f_{\mu_0}(z_i) = \frac{1}{2m}\sum_{i=1}^{m}\sigma_i(1+\cos(\mu_0 z_i)) \geq c + \frac{1}{2}\sum_{i=1}^{m}\sigma_i.$$

This means that when $\mathcal{S} = \{(z_1, y_1), \ldots, (z_m, y_m)\}$,

$$\begin{aligned}
\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) &\geq \widehat{\mathcal{R}}(\ell \circ \mathcal{F}) \\
&= \frac{1}{m}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{\mu\in\mathcal{R}}\sum_{i=1}^{m}\sigma_i\left|f_\mu(z_i) - y_i\right|\right] \\
&= \frac{1}{m}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{\mu\in\mathcal{R}}\sum_{i=1}^{m}\sigma_i\left|f_\mu(z_i)\right|\right] && (y_i = 0) \\
&= \frac{1}{m}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{\mu\in\mathcal{R}}\sum_{i=1}^{m}\sigma_i f_\mu(z_i)\right] && \big(f_\mu(z_i) \geq 0\big) \\
&\geq c + \frac{1}{2}\mathbb{E}_{\boldsymbol{\sigma}}\left[\sum_{i=1}^{m}\sigma_i\right] \\
&= c.
\end{aligned}$$

To this end, given an assignment of the variables $\sigma_1, \ldots, \sigma_m \in \{-1, 1\}$, let $(b_1, \ldots, b_m) \in \{0, 1\}^m$ be defined such that

$$b_i = \begin{cases} 0 & \text{if } \sigma_i = 1 \\ 1 & \text{otherwise} \end{cases}$$

and let

$$\mu_0 = 2\pi\gamma^p\left(\sum_{j=1}^{m}\alpha^j b_j + \alpha^{m+1}\right).$$

Since $0 < \mu_0 < 2\pi\gamma^p\sum_{j=1}^{\infty}\alpha^j = \frac{2\pi\gamma^p\alpha}{1-\alpha} \leq \gamma^p$, μ_0 is an element of the parameter space $(0, \gamma^p]$. The inequality $\frac{2\pi\gamma^p\alpha}{1-\alpha} \leq \gamma^p$ holds because $\alpha \leq \frac{1}{2\pi+1}$, so $\frac{2\pi\alpha}{1-\alpha} \leq 1$.

Next, we evaluate $f_{\mu_0}(z_i) = \frac{1}{2}(1 + \cos(\mu_0 z_i))$:

$$\begin{aligned}
\frac{1}{2}(1 + \cos(\mu_0 z_i)) &= \frac{1}{2} + \frac{1}{2}\cos\left(2\pi\gamma^p \left(\sum_{j=1}^{m} \alpha^j b_j + \alpha^{m+1}\right) \frac{\alpha^{-i}}{2\gamma^p}\right) \\
&= \frac{1}{2} + \frac{1}{2}\cos\left(\pi \left(\sum_{j=1}^{m} \alpha^j b_j + \alpha^{m+1}\right) \alpha^{-i}\right) \\
&= \frac{1}{2} + \frac{1}{2}\cos\left(\sum_{j=1}^{i-1} \alpha^{j-i}\pi b_j + \pi b_i + \sum_{j=i+1}^{m} \alpha^{j-i}\pi b_j + \alpha^{m+1-i}\pi\right) \\
&= \frac{1}{2} + \frac{1}{2}\cos\left(\pi\left(b_i + \sum_{j=1}^{m-i} \alpha^j b_{i+j} + \alpha^{m+1-i}\right)\right). \qquad (21)
\end{aligned}$$

The final equality holds because, for every $j < i$, α^{j-i} is a positive power of 2, so $\alpha^{j-i}\pi b_j$ is a multiple of 2π. We will use the following fact: Since

$$0 < \sum_{j=1}^{m-i} \alpha^j b_{i+j} + \alpha^{m+1-i} \le \sum_{j=1}^{m-i+1} \alpha^j < \sum_{j=1}^{\infty} \alpha^j = \frac{\alpha}{1-\alpha},$$

the argument of $\cos(\cdot)$ in Equation (21) lies strictly between πb_i and $\pi b_i + \frac{\pi\alpha}{1-\alpha}$.

Suppose $b_i = 0$. Since $\alpha \le \frac{1}{2}$, we know that $\frac{\pi\alpha}{1-\alpha} \le \pi$. Therefore, $\cos(\cdot)$ is monotone decreasing on the interval $[0, \frac{\pi\alpha}{1-\alpha}]$. Moreover, we know that $\frac{1}{2}\cos\frac{\pi\alpha}{1-\alpha} \ge c$. Therefore, $f_{\mu_0}(z_i) = \frac{1}{2}(1 + \cos(\mu_0 z_i)) \ge \frac{1}{2} + c$. Since $b_i = 0$, it must be that $\sigma_i = 1$, so $\sigma_i f_{\mu_0}(z_i) \ge c + \frac{1}{2} = c + \frac{\sigma_i}{2}$. Meanwhile, suppose $b_i = 1$. The function $\cos(\cdot)$ is monotone increasing on the interval $[\pi, \pi + \frac{\pi\alpha}{1-\alpha}]$. Moreover, $\frac{1}{2}\cos(\pi + \frac{\pi\alpha}{1-\alpha}) = -\frac{1}{2}\cos\frac{\pi\alpha}{1-\alpha} \le -c$. Therefore, $f_{\mu_0}(z_i) = \frac{1}{2}(1 + \cos(\mu_0 z_i)) \le \frac{1}{2} - c$. Since $b_i = 1$, it must be that $\sigma_i = -1$, so $\sigma_i f_{\mu_0}(z_i) \ge c - \frac{1}{2} = c + \frac{\sigma_i}{2}$. Since this is true for any $i \in [m]$, we have that

$$\frac{1}{2m}\sum_{i=1}^{m} \sigma_i (1 + \cos(\mu_0 z_i)) \ge c + \frac{1}{2}\sum_{i=1}^{m} \sigma_i,$$

as claimed. □

We conclude this proof by showing that $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) \le \frac{1}{2}$.

Claim B.22. *For any $m \ge 1$, $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) \le \frac{1}{2}$.*

Proof of Claim B.22. Let $\mathcal{S} = \{(z_1, y_1), \ldots, (z_m, y_m)\} \subset [\frac{1}{2\gamma^p}, \infty) \times [0, 1]$ be an arbitrary set of points. For any assignment of the variables $\sigma_1, \ldots, \sigma_m \in \{-1, 1\}$, since $|f_\mu(z_i) - y_i| \in [0, 1]$,

$$\sup_{\mu \in (0, \gamma^p]} \sum_{i=1}^{m} \sigma_i \left|f_\mu(z_i) - y_i\right| \le \sum_{i=1}^{m} \mathbf{1}_{\{\sigma_i = 1\}}.$$

Therefore,

$$\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) = \sup_{(z_1, y_1), \ldots, (z_m, y_m)} \frac{1}{m}\mathbb{E}_\sigma\left[\sup_{\mu \in (0, \gamma^p]} \sum_{i=1}^{m} \sigma_i \left|f_\mu(z_i) - y_i\right|\right] \le \frac{1}{m}\mathbb{E}_\sigma\left[\sum_{i=1}^{m} \mathbf{1}_{\{\sigma_i = 1\}}\right] = \frac{1}{2},$$

as claimed. □

Together, Claims B.21 and B.22 imply that, for every $m \ge 1$, $\overline{\mathcal{R}}_m(\ell \circ \mathcal{F}) = \frac{1}{2}$. □

Lemma B.23. *Let $\mathcal{G} = \{g_\mu \mid \mu \in \mathbb{R}\} \subseteq [0,1]^{\mathcal{Z}}$ be a set of functions mapping a set $\mathcal{Z}$ to $[0,1]$ parameterized by a single real value $\mu \in \mathbb{R}$. Suppose that every function $g_z^* \in \mathcal{G}^* \subseteq [0,1]^{\mathbb{R}}$ is piecewise-constant with at most j pieces. Then, for any set $\mathcal{S} = \{z_1, \ldots, z_m\} \subseteq \mathcal{Z}$,*

$$\widehat{\mathcal{R}}(\mathcal{G}) = \frac{1}{m}\mathbb{E}_{\sigma\sim\{-1,1\}^m}\left[\sup_{\mu\in\mathbb{R}}\sum_{i=1}^{m}\sigma_i g_\mu(z_i)\right] \leq \sqrt{\frac{2\ln(m(j-1)+1)}{m}}.$$

Proof. We will use Massart's lemma (Lemma 3.7) to prove this lemma. Let $A \subseteq [0,1]^m$ be the following set of vectors:

$$A = \left\{\begin{pmatrix} g_\mu(z_1) \\ \vdots \\ g_\mu(z_m)\end{pmatrix} : \mu \in \mathbb{R}\right\}.$$

By definition of the dual class,

$$A = \left\{\begin{pmatrix} g^*_{z_1}(\mu) \\ \vdots \\ g^*_{z_m}(\mu)\end{pmatrix} : \mu \in \mathbb{R}\right\}.$$

Since each function $g^*_{z_i}$ is piecewise-constant with at most j pieces, $|A| \leq m(j-1)+1$. The lemma statement therefore follows from Massart's lemma. □

Lemma B.24. *Let $\mathcal{F} = \{f_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\} \subseteq [0,1]^{\mathcal{Z}}$ and $\mathcal{G} = \{g_{\boldsymbol{\mu}} \mid \boldsymbol{\mu} \in \mathcal{R}\} \subseteq [0,1]^{\mathcal{Z}}$ be two sets of functions mapping a domain $\mathcal{Z}$ to $[0,1]$. With probability $1-\delta$ over the draw of m samples $z_1, \ldots, z_m \sim \mathcal{D}$,*

$$\mathbb{E}_{z\sim\mathcal{D}}\left[\left\|f_z^* - g_z^*\right\|_\infty\right] \leq \frac{1}{m}\sum_{i=1}^{m}\left\|f^*_{z_i} - g^*_{z_i}\right\|_\infty + \sqrt{\frac{1}{2m}\ln\frac{1}{\delta}}. \tag{22}$$

Proof. Let $h : \mathcal{Z} \to [0,1]$ be defined such that $h(z) = \left\|f_z^* - g_z^*\right\|_\infty$. From Hoeffding's inequality, we know that with probability $1-\delta$ over the draw of m samples $z_1, \ldots, z_m \sim \mathcal{D}$,

$$\mathbb{E}_{z\sim\mathcal{D}}[h(z)] \leq \frac{1}{m}\sum_{i=1}^{m}h(z_i) + \sqrt{\frac{1}{2m}\ln\frac{1}{\delta}},$$

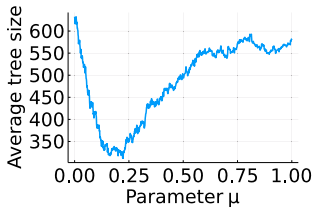
which implies that Equation (22) holds. □

C ADDITIONAL INFORMATION ABOUT EXPERIMENTS

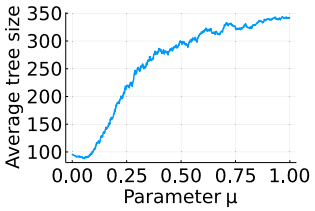
In this appendix, we begin by describing the MILP formulations of several problems from Section 4, where we described our experiments.

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:

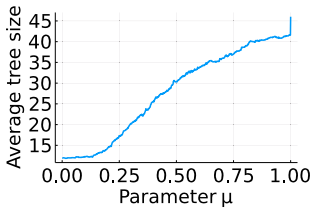
$$\begin{array}{lll}
\min & \sum_{i,j\in[n]} d(p_i,p_j)\, y_{ij} & \\
\text{s.t.} & \sum_{i=1}^{n} x_i = k & \\
& \sum_{j=1}^{n} y_{ij} = 1 & \forall i \in [n] \\
& y_{ij} \leq x_j & \forall i,j \in [n] \\
& x_i \in \{0,1\} & \forall i \in [n] \\
& y_{ij} \in \{0,1\} & \forall i,j \in [n].
\end{array}$$



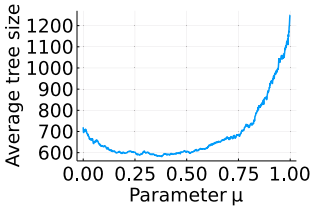
(a) CATS "arbitrary": 200 bids, 100 goods



(b) CATS "regions": 400 bids, 200 goods



(c) Clustering



(d) Linear separators

Fig. C.1. The geometric mean tree size produced by B&B when run with the linear scoring rule with parameter μ (Equation (10)).

Agnostically Learning Linear Separators. We can formulate this problem as a MILP as follows: Let $M > \max \|\boldsymbol{p}_i\|_1$.

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

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.[14]

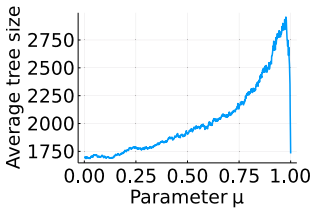
Moreover, for each of the training sets used to compute Figures C.1–C.3, we plot the geometric mean tree size as a function of the mixing parameter μ (whereas Figures 10–12 display the arithmetic mean).

D TREE SEARCH FOR CONSTRAINT SATISFACTION PROBLEMS

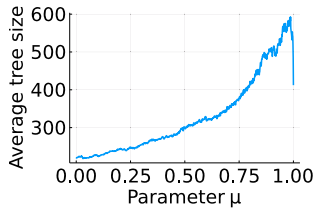
In this section, we describe how our generalization bounds from Section 3.2 generalize to tree search algorithms for solving *constraint satisfaction problems (CSPs)*, which we introduced in Section 3.4. A tree search algorithm takes as input a tuple $\Pi = (X, D, f, g)$, where $X = \{x_1, \ldots, x_n\}$ 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, `localFathom` and

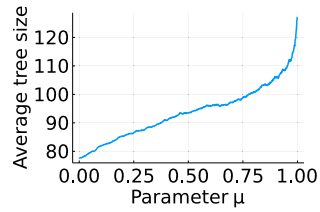
[14]In practice, we implement this constraint by enforcing that $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle \geq -M x_i + \gamma$ for some tiny $\gamma > 0$ since MILP solvers cannot enforce strict inequalities.



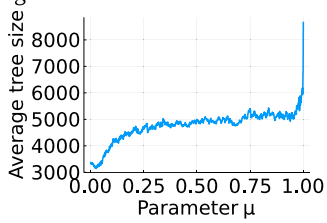
(a) CATS "arbitrary": 150 bids, 100 goods



(b) CATS "regions": 300 bids, 200 goods

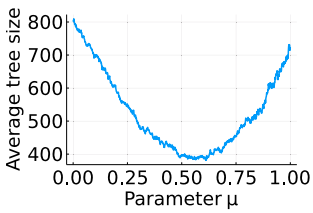


(c) Clustering

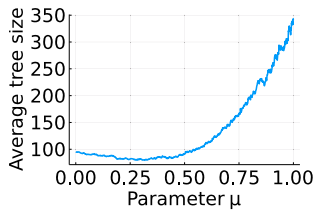


(d) Linear separators

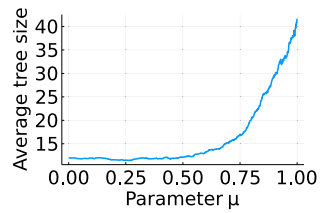
Fig. C.2. The geometric mean tree size produced by B&B when run with the linear scoring rule with parameter μ using pseudocost branching.



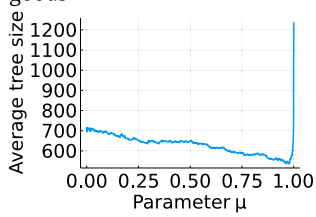
(a) CATS "arbitrary": 200 bids, 100 goods



(b) CATS "regions": 400 bids, 200 goods



(c) Clustering



(d) Linear separators

Fig. C.3. The geometric mean tree size produced by B&B when run with the product scoring rule with parameter μ (Equation (11)) using strong branching.

`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 $\texttt{globalFathom}(\Pi, \boldsymbol{y}, \mathcal{T}) = \text{fathom}$ if $c \leq c^*$.

See Algorithm 3 for the tree search pseudocode.

ALGORITHM 3: Tree search for constraint satisfaction problems

Input: A problem instance $\Pi = (X, D, f, g)$.
Let $\mathcal{T}$ be a tree that consists of a single node containing the empty partial solution $(\top, \ldots, \top)$.
while *there remains an unfathomed leaf in* $\mathcal{T}$ **do**
  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.
  Use a *variable-selection policy* to choose a variable $x_i \in X$ to branch on at that leaf.
  For all $j \in D_i$, let $\boldsymbol{y}_i^{(j)}$ be the partial solution $\boldsymbol{y}$ except with the component $y[i] = j$.
  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.
  for $j \in D_i$ **do**
    if $\texttt{localFathom}(\Pi, \boldsymbol{y}_i^{(j)}) = \text{fathom}$ **then**
      Update $\mathcal{T}'$ so that the leaf containing the partial solution $\boldsymbol{y}_i^{(j)}$ is fathomed.
    else if $\texttt{globalFathom}(\Pi, \boldsymbol{y}_i^{(j)}, \mathcal{T}') = \text{fathom}$ **then**
      Update $\mathcal{T}'$ so that the leaf containing the partial solution $\boldsymbol{y}_i^{(j)}$ is fathomed.
  end
  Set $\mathcal{T} = \mathcal{T}'$.
end
Output: The best known feasible solution $\boldsymbol{y}^*$, if one exists. Otherwise, return `Null`.

D.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 $\texttt{score}_1, \ldots, \texttt{score}_d$ be a set of variable-selection scoring rules, such as those in Section 3.4.2. 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 `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 $\texttt{score}_1, \ldots, \texttt{score}_d$ if for any distribution $\mathcal{D}$, with probability at least $1 - \delta$ over the draw of a sample $\{\Pi_1, \ldots, \Pi_m\} \sim \mathcal{D}^m$, the algorithm returns a convex combination $\texttt{score} = \hat{\mu}_1\texttt{score}_1 + \cdots + \hat{\mu}_d\texttt{score}_d$ such that $\mathbb{E}_{\Pi\sim\mathcal{D}}[\texttt{cost}(\Pi, \texttt{score})] - \mathbb{E}_{\Pi\sim\mathcal{D}}[\texttt{cost}(\Pi, \texttt{score}^*)] \leq \epsilon$, 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.

We assume that the problem instances in the support of $\mathcal{D}$ are over n D-ary variables, for some $n, D \in \mathbb{N}$.[15]

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

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

D.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 D.1 (Score-Based Variable-Selection Policy). Let $\texttt{score}$ 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 ($\texttt{score}(\mathcal{T}, Q, i) \in \mathbb{R}$). 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 $\text{argmax}_{x_j \in N_{\mathcal{T},\boldsymbol{y}}} \{\texttt{score}(\mathcal{T}, \boldsymbol{y}, j)\}$ to branch on at the node Q.

Definition D.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 $\texttt{score}(\mathcal{T}, \boldsymbol{y}, i)$ is a path-wise scoring rule if the value of $\texttt{score}(\mathcal{T}, \boldsymbol{y}, i)$ 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, $\texttt{score}(\mathcal{T}, \boldsymbol{y}, i) = \texttt{score}(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, i)$.

The following lemma parallels Lemma 3.2:

Lemma D.3. *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* 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 3 except it does not use the function $\texttt{globalFathom}$, but rather skips Steps 10 and 11 of Algorithm 3. We then relate the behavior of TS' to the behavior of Algorithm 3 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 D.4. *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 search 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

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

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 $\texttt{localFathom}(\Pi, \boldsymbol{y}) = \text{fathom}$, 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 $\texttt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell)$ and $\texttt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell)$ 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 $\mu\texttt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell) + (1-\mu)\texttt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, \ell)$ 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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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

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

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 $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ and $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. □

Next, we explicitly relate the behavior of Algorithm 3 to TS', proving that the search tree Algorithm 3 builds is a rooted subtree of the search tree TS' builds.

Claim D.5. *Given a parameter $\mu \in [0,1]$, let $\mathcal{T}$ and $\mathcal{T}'$ be the trees Algorithm 3 and TS' build, respectively, using the scoring rule $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$. 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 D.5. 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 $\texttt{localFathom}(\Pi, \boldsymbol{y}') = \text{fathom}$, we know that Algorithm 3 will also fathom this node. However, this is not the case since Algorithm 3 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 3 (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, $\mu\texttt{score}_1(\bar{\mathcal{T}}, \boldsymbol{y}', k) + (1-\mu)\texttt{score}_2(\bar{\mathcal{T}}, \boldsymbol{y}', k) = \mu\texttt{score}_1(\mathcal{T}_{\boldsymbol{y}'}, \boldsymbol{y}', k) + (1-\mu)\texttt{score}_2(\mathcal{T}_{\boldsymbol{y}'}, \boldsymbol{y}', k) = \mu\texttt{score}_1(\bar{\mathcal{T}}', \boldsymbol{y}', k) + (1-\mu)\texttt{score}_2(\bar{\mathcal{T}}', \boldsymbol{y}', k)$. This means that Algorithm 3 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 D.4 and D.5 to prove Lemma D.3. Let $I_1, \ldots, I_T$ be the intervals guaranteed to exist by Claim D.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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ and $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$ result in the same search tree. For a contradiction, suppose that this is not the case. Consider the first iteration where of Algorithm 3 using the scoring rule $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ differs from Algorithm 3 using the scoring rule $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$, where an iteration corresponds to lines 2 through 14 of Algorithm 3. Up until this iteration, Algorithm 3 has built the same partial search tree $\mathcal{T}$. Since the node-selection policy does not depend on μ' or μ'', Algorithm 3 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 3 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 D.4, 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 $\mu'\texttt{score}_1 + (1-\mu')\texttt{score}_2$ (respectively, $\mu''\texttt{score}_1 + (1-\mu'')\texttt{score}_2$) by the time it branches on the node containing the partial solution $\boldsymbol{y}$. By Claims D.4 and D.5, we know that $\mathcal{T}_{\boldsymbol{y}}$ is the path from the root to the node containing the partial solution $\boldsymbol{y}$ of both $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$. By Claim D.4, 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 $i = \text{argmax}_j \left\{\mu'\texttt{score}_1(\bar{\mathcal{T}}', \boldsymbol{y}, j) + (1-\mu')\texttt{score}_2(\bar{\mathcal{T}}', \boldsymbol{y}, j)\right\}$, or equivalently,

$$i = \text{argmax}_j \left\{\mu'\texttt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j) + (1-\mu')\texttt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j)\right\}, \tag{23}$$

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

$$i = \text{argmax}_j \left\{\mu''\texttt{score}_1(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j) + (1-\mu'')\texttt{score}_2(\mathcal{T}_{\boldsymbol{y}}, \boldsymbol{y}, j)\right\}. \tag{24}$$

Returning to the search tree $\mathcal{T}$ that Algorithm 3 is building, Equation (23) implies that

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

and Equation (24) implies that $i = \text{argmax}_j \{\mu''\texttt{score}_1(\mathcal{T}, \boldsymbol{y}, j) + (1-\mu'')\texttt{score}_2(\mathcal{T}, \boldsymbol{y}, j)\}$. Therefore, Algorithm 3 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 3 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 3 are identical. Therefore, the lemma holds. □

D.2.1 Convex Combinations of 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 3.2.2. The proofs in this section are very similar to those from Section 3.2.2; we include them here for the sake of completeness.

Lemma D.6. *Let* `cost` *be a tree-constant cost function, let* $\texttt{score}_1, \ldots, \texttt{score}_d$ *be* d *arbitrary scoring rules, and let* Π *be an arbitrary problem instance over* n *D-ary variables. Suppose we limit Algorithm 3 to producing search trees of size* κ*. There is a set* $\mathcal{H}$ *of at most* $\kappa n^{\kappa+2}$ *hyperplanes such that for any connected component* R *of* $[0,1]^d \setminus \mathcal{H}$*, the search tree Algorithm 3 builds using the scoring rule* $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ *is invariant across all* $(\mu_1, \ldots, \mu_d) \in R$*.*

Proof. The proof has two steps. In Claim D.7, we show that there are at most n^κ different search trees that Algorithm 3 might produce for the instance Π as we vary the mixing parameter vector $(\mu_1, \ldots, \mu_d)$. In Claim D.8, 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 κn^2 halfspaces. These facts together prove the lemma.

Claim D.7. *There are only* n^κ *different search trees that can be achieved by varying the parameter vector* $(\mu_1, \ldots, \mu_d)$*.*

Proof of Claim D.7. Fix any d mixing parameters $(\mu_1, \ldots, \mu_d)$ and let $v_1, \ldots, v_\kappa \in [n]$ be the sequence of branching variables chosen by Algorithm 3 run with scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$, 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 3 with scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ produces a tree of size $k < \kappa$, 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 κ). 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 3 with scoring rules of the form $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ is at most n^κ, since there are only n^κ distinct sequences of κ variables $v_1, \ldots, v_\kappa \in [n]$.

Let $(\mu_1, \ldots, \mu_d)$ and $(\mu'_1, \ldots, \mu'_d)$ be two sets of mixing parameters, and suppose running Algorithm 3 with $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ and $\mu'_1\texttt{score}_1 + \cdots + \mu'_d\texttt{score}_d$ both results in the sequence of branching variable decisions being $v_1, \ldots, v_\kappa$. We prove that the resulting search trees are identical by induction on the iterations of the algorithm, where an iteration corresponds to lines 2 through 14 of Algorithm 3. 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 $v_1, \ldots, v_\kappa$ 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 3 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. □

Next, we argue that for any given search tree $\mathcal{T}$ produced by Algorithm 3, the set of mixing parameters $(\mu_1, \ldots, \mu_d)$ giving rise to $\mathcal{T}$ is defined by the intersection of $n^{\kappa+2}$ halfspaces.

Claim D.8. *For a given search tree $\mathcal{T}$, there are at most κn^2 halfspaces such that Algorithm 3 using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ builds the tree $\mathcal{T}$ if and only if $(\mu_1, \ldots, \mu_d)$ lies in the intersection of those halfspaces.*

Proof of Claim D.8. Let $v_1, \ldots, v_\kappa$ 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 3. Let $\mathcal{T}_t$ be the state of Algorithm 3 after t iterations.

Induction Hypothesis. For a given index $t \in [\kappa]$, there are at most tn^2 halfspaces such that Algorithm 3 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 $(\mu_1, \ldots, \mu_d)$ 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 $t < \kappa$ 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 3 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 $(\mu_1, \ldots, \mu_d)$ lies in the intersection of those halfspaces. Let $\boldsymbol{y}$ be the partial solution contained in the next node that Algorithm 3 will branch on given $\mathcal{T}_t$. We know that Algorithm 3 will choose to branch on variable v_{t+1} at this node if and only if

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

Since these functions are linear in $(\mu_1, \ldots, \mu_d)$, there are at most n^2 halfspaces defining the region where $v_{t+1} = \text{argmax}\{\mu_1\texttt{score}_1(\mathcal{T}, \boldsymbol{y}, v') + \cdots + \mu_d\texttt{score}_d(\mathcal{T}, \boldsymbol{y}, v')\}$. Let $\mathcal{B}'$ be this set of halfspaces. Algorithm 3 using the scoring rule $\mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d$ builds the partial tree $\mathcal{T}_{t+1}$ after $t+1$ iterations if and only if $(\mu_1, \ldots, \mu_d)$ lies in the intersection of the $(t+1)n^2$ halfspaces in the set $\mathcal{B} \cup \mathcal{B}'$. □

□

Theorem D.9. *Let* cost *be a tree-constant cost function and let* $\texttt{score}_1, \ldots, \texttt{score}_d$ *be d arbitrary scoring rules. Suppose we limit Algorithm 3 to producing search trees of size κ. Let C be the set of functions* $\{\texttt{cost}(\cdot, \mu_1\texttt{score}_1 + \cdots + \mu_d\texttt{score}_d) : (\mu_1, \ldots, \mu_d) \in [0,1]^d\}$. *Then* $\text{Pdim}(C) = O(d\kappa \log n + d \log d)$.

Proof. This theorem follows from Lemma D.6 in the exact same way that Theorem B.15 follows from Lemma 3.10. □

D.2.2 Non-linear Combinations of Scoring Rules.

Lemma D.10. *Let* $\texttt{cost}$ *be a tree-constant cost function, let* $\texttt{score}_1, \ldots, \texttt{score}_d$ *be* d *arbitrary scoring rules, and let* Π *be an arbitrary problem instance. Then*

$$\texttt{cost}\left(\Pi, \prod_{i=1}^{d} \texttt{score}_i^{\mu_i}\right) = \texttt{cost}\left(\Pi, \sum_{i=1}^{d} \mu_i \log \texttt{score}_i\right).$$

Proof. We claim that Algorithm 3 builds the same tree using the scoring rule $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$ as it does using the scoring rule $\sum_{i=1}^{d} \mu_i \log \texttt{score}_i$. Therefore, since $\texttt{cost}$ is tree-constant, the lemma statement holds.

For a contradiction, suppose Algorithm 3 does not build the same tree using the scoring rule $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$ as it does using the scoring rule $\sum_{i=1}^{d} \mu_i \log \texttt{score}_i$. Consider the first round of Algorithm 3 where the algorithm's behavior using the scoring rule $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$ differs from its behavior using the scoring rule $\sum_{i=1}^{d} \mu_i \log \texttt{score}_i$. Let $\mathcal{T}'$ be the tree Algorithm 3 has built up until that round and let $\boldsymbol{y}$ be the partial solution contained in the next node that Algorithm 3 will branch on given $\mathcal{T}'$. It must be that at Step 4, the algorithm chose a different variable to branch on depending on the scoring rule. In other words,

$$\operatorname{argmax}_j \left\{ \prod_{i=1}^{d} \texttt{score}_i(\mathcal{T}', \boldsymbol{y}, j)^{\mu_i} \right\} \neq \operatorname{argmax}_j \left\{ \sum_{i=1}^{d} \mu_i \log \texttt{score}_i(\mathcal{T}', \boldsymbol{y}, j) \right\}.$$

However, if $j^* = \operatorname{argmax}_j \left\{ \prod_{i=1}^{d} \texttt{score}_i(\mathcal{T}', \boldsymbol{y}, j)^{\mu_i} \right\}$, then[16]

$$j^* = \operatorname{argmax}_j \left\{ \log \left(\prod_{i=1}^{d} \texttt{score}_i(\mathcal{T}', \boldsymbol{y}, j)^{\mu_i} \right) \right\},$$

which means that $j^* = \operatorname{argmax}_j \{ \sum_{i=1}^{d} \mu_i \log \texttt{score}_i(\mathcal{T}', \boldsymbol{y}, j) \}$, which is a contradiction. Therefore, Algorithm 3 builds the same tree using the scoring rule $\prod_{i=1}^{d} \texttt{score}_i^{\mu_i}$ as it does using the scoring rule $\sum_{i=1}^{d} \mu_i \log \texttt{score}_i$, so the lemma statement holds. □

ACKNOWLEDGMENTS

We thank Kevin Leyton-Brown for a stimulating discussion that inspired us to pursue the research described in Section 3.3.

REFERENCES

Tobias Achterberg. 2007. *Constraint Integer Programming*. Ph.D. Dissertation. Technische Universität Berlin.

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

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

Saba Ahmadi, Hedyeh Beyhaghi, Avrim Blum, and Keziah Naggita. 2022. Setting fair incentives to maximize improvement. *arXiv preprint arXiv:2203.00134* (2022).

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

Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*. Springer-Verlag, 142–157.

[16]For any set $X \subset \mathbb{R}_{>0}$, $\operatorname{argmax}_{x \in X} \{x\} = \operatorname{argmax}_{x \in X} \{\log x\}$. After all, if not, then there are $x_1, x_2 \in X$ such that $x_1 < x_2$ but $\log x_1 > \log x_2$, which is a contradiction.

Martin Anthony and Peter Bartlett. 2009. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.

Maria-Florina Balcan. 2020. Data-driven algorithm design. In *Beyond Worst Case Analysis of Algorithms*, Tim Roughgarden (Ed.). Cambridge University Press. (Forthcoming).

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. 2018a. Learning to branch. In *Proceedings of the International Conference on Machine Learning (ICML)* (2018).

Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. 2018b. Dispersion for data-driven algorithm design, online learning, and private optimization. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS'18)*.

Maria-Florina Balcan, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. 2022. Provably tuning the ElasticNet across instances. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'22)*.

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. 2017. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Proceedings of the Conference on Learning Theory (COLT'17)*.

Maria-Florina Balcan, Travis Dick, and Manuel Lang. 2020a. Learning to link. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*.

Maria-Florina Balcan, Travis Dick, and Wesley Pegden. 2020b. Semi-bandit optimization in the dispersed setting. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'20)*.

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. 2020c. Learning to optimize computational resources: Frugal training with generalization guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'20)*.

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. 2020d. Refined bounds for algorithm configuration: The knife-edge of dual class approximability. In *International Conference on Machine Learning (ICML'20)*.

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. 2021a. Generalization in portfolio-based algorithm selection. In *AAAI Conference on Artificial Intelligence (AAAI'21)*.

Maria-Florina F. Balcan, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. 2021b. Learning-to-learn non-convex piecewise-Lipschitz functions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'21)*.

Maria-Florina F. Balcan and Dravyansh Sharma. 2021. Data driven semi-supervised learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (2021).

Peter Bartlett, Piotr Indyk, and Tal Wagner. 2022. Generalization bounds for data-driven numerical linear algebra. In *Proceedings of the Conference on Learning Theory (COLT'22)*.

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

Yoshua Bengio, Emma Frejinger, Andrea Lodi, Rahul Patel, and Sriram Sankaranarayanan. 2020a. A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs. In *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'20)*.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2020b. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research* (2020).

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

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

Avrim Blum, Chen Dan, and Saeed Seddighin. 2021. Learning complexity of simulated annealing. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'21)*.

Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. 2018. Learning a classification of mixed-integer quadratic programming problems. In *Proceedings of the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'18)*. 595–604.

Vincent Cohen-Addad and Varun Kanade. 2017. Online optimization of smoothed piecewise constant functions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'17)*.

Santanu S. Dey, Yatharth Dubey, and Marco Molinaro. 2021a. Branch-and-bound solves random binary IPS in polytime. In *Proceedings of theAnnual ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*.

Santanu S. Dey, Yatharth Dubey, and Marco Molinaro. 2021b. Lower bounds on the size of general branch-and-bound trees. *arXiv preprint arXiv:2103.09807* (2021).

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

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. 2019. Exact combinatorial optimization with graph convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'19)*. 15554–15566.

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

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

Carla Gomes and Bart Selman. 2001. Algorithm portfolios. *Artificial Intelligence* 126 (2001), 43–62.

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

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

He He, Hal Daume III, and Jason M. Eisner. 2014. Learning to search in branch and bound algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'14)*.

Eric Horvitz, Yongshao Ruan, Carla Gomez, Henry Kautz, Bart Selman, and Max Chickering. 2001. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'01)*.

Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-based frequency estimation algorithms. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2010. Automated configuration of mixed integer programming solvers. In *Proceedings of the International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer, 186–202.

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

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

Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence* 206 (2014), 79–111. https://doi.org/10.1016/j.artint.2013.10.003

IBM ILOG Inc. 2007. CPLEX 11.0 Release Notes.

IBM ILOG Inc. 2017. CPLEX 12.8 Parameters Reference.

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

Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC-instance-specific algorithm configuration. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'10)*.

Elias Boutros Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. 2016. Learning to branch in mixed integer programming. In *Proceedings of theAAAI Conference on Artificial Intelligence (AAAI'16)*.

Elias Boutros Khalil, Bistra Dilkina, George Nemhauser, Shabbir Ahmed, and Yufen Shao. 2017. Learning to run heuristics in tree search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'17)*.

Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. 2017. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'17)*.

Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. 2019. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (2019).

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

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

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

Michail G. Lagoudakis and Michael L. Littman. 2001. Learning to select branching rules in the DPLL procedure for satisfiability. *Electronic Notes in Discrete Mathematics* 9 (2001), 344–359.

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

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

Kevin Leyton-Brown, Paul Milgrom, and Ilya Segal. 2017. Economics and computer science of a radio spectrum reallocation. *Proceedings of the National Academy of Sciences* 114, 28 (2017), 7202–7209.

Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. 2009. Empirical hardness models: Methodology and a case study on combinatorial auctions. *J. ACM* 56, 4 (2009), 1–52.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. 2000. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC'00)*. Minneapolis, MN, 66–76.

Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. 2016. Learning rate based branching heuristic for SAT solvers. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*. Springer, 123–140.

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

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

L. Lobjois and M. Lemaître. 1998. Branch and bound algorithm selection by performance prediction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'98)*. San Jose, CA, 353–358.

Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive caching with machine learned advice. In *Proceedings of the International Conference on Machine Learning (ICML'18)*.

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

Michael Mitzenmacher. 2018. A model for learned bloom filters and optimizing by sandwiching. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'18)*. 464–473.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. *Foundations of Machine Learning*. MIT Press.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving online algorithms via ML predictions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'18)*. 9661–9670.

Yasha Pushak and Holger Hoos. 2018. Algorithm configuration landscapes. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Springer, 271–283.

John R. Rice. 1976. The algorithm selection problem. *Advances in Computers* 15 (1976), 65–118.

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

Shinsaku Sakaue and Taihei Oki. 2022. Sample complexity of learning heuristic functions for greedy-best-first and A* search. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'22)*.

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

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

Tzur Sayag, Shai Fine, and Yishay Mansour. 2006. Combining multiple heuristics. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 242–253.

Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Dravyansh Sharma, Maria-Florina Balcan, and Travis Dick. 2020. Learning piecewise Lipschitz functions in changing environments. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'20)*.

Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. 2020. A general large neighborhood search framework for solving integer programs. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'20)*.

Jialin Song, Ravi Lanka, Albert Zhao, Aadyot Bhatnagar, Yisong Yue, and Masahiro Ono. 2018. Learning to search via retrospective imitation. *arXiv preprint arXiv:1804.00846* (2018).

David Speck, André Biedenkapp, Frank Hutter, Robert Mattmüller, and Marius Lindauer. 2021. Learning heuristic selection with dynamic algorithm configuration. In *International Conference on Automated Planning and Scheduling (ICAPS'21)*, Vol. 31. 597–605.

Daniel A. Spielman and Shang-Hua Teng. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)* 51, 3 (2004), 385–463.

Matthew Streeter and Daniel Golovin. 2009. An online algorithm for maximizing submodular functions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'09)*. 1577–1584.

Matthew Streeter, Daniel Golovin, and Stephen F. Smith. 2007. Combining multiple heuristics online. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'07)*.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. 2020. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the International Conference on Machine Learning (ICML)* (2020).

Gellért Weisz, András György, and Csaba Szepesvári. 2018. LeapsAndBounds: A method for approximately optimal algorithm configuration. In *Proceedings of the International Conference on Machine Learning (ICML'18)*.

Gellért Weisz, András György, and Csaba Szepesvári. 2019. CapsAndRuns: An improved method for approximately optimal algorithm configuration. In *Proceedings of the International Conference on Machine Learning (ICML'19)*.

Wei Xia and Roland Yap. 2018. Learning robust search strategies using a bandit-based approach. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'18)*.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32, 1 (2008), 565–606.

Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'10)*.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proceedings of the RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI'11)*.

Received 10 October 2020; revised 14 March 2023; accepted 28 September 2023