Learning to Branch

Maria-Florina Balcan [1] **Travis Dick** [1] **Tuomas Sandholm** [1] **Ellen Vitercik** [1]

Abstract

Tree search algorithms, such as branch-and-bound, are the most widely used tools for solving combinatorial problems. These algorithms recursively partition the search space to find an optimal solution. To keep the tree small, it is crucial to carefully decide, when expanding a tree node, which variable to branch on at that node to partition the remaining space. Many partitioning techniques have been proposed, but no theory describes which is optimal. We show how to use machine learning to determine an optimal weighting of any set of partitioning procedures for the instance distribution at hand using samples. Via theory and experiments, we show that learning to branch is both practical and hugely beneficial.

1. Introduction

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

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

[1]Computer Science Department, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. Correspondence to: Ellen Vitercik <vitercik@cs.cmu.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

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

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

Our learning algorithm adaptively partitions the parameter

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

The sensitivity of tree search algorithms to small changes in their parameters is a key challenge that differentiates our sample complexity analysis from those typically found in machine learning. For many well-understood function classes in machine learning, there is a close connection between the distance in parameter space between two parameter vectors and the distance in function space between the two corresponding functions. Understanding this connection is a necessary prerequisite to analyzing how many significantly different functions there are in the class, and thereby quantifying the class's intrinsic complexity. Intrinsic complexity typically translates to VC dimension, Rademacher complexity, or some other metric which allows us to derive learnability guarantees. Since the tree size of a search algorithm as a function of its parameters does not exhibit this predictable behavior, we must carefully analyze the way in which the parameters influence each step of the procedure in order to derive learning algorithms with strong guarantees. In doing so, we present the first sample complexity guarantees for automated configuration of tree search algorithms. We provide worst-case bounds proving that a surprisingly small number of samples are sufficient for strong learnability guarantees: the sample complexity bound grows quadratically in the size of the problem instance, despite the complexity of the algorithms we study.

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

Related work. Several works have studied the use of machine learning techniques in the context of B&B; for an overview, see the summary by Lodi & Zarpellon (2017).

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

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

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

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

The learning-theoretic model of algorithm configuration that we study in this paper was introduced to the theoretical computer science community by Gupta & Roughgarden

(2017). Under this model, an application domain is modeled as a distribution over problem instances and the goal is to PAC-learn an algorithm that is nearly optimal over the distribution. This model was later studied by Balcan et al. (2017) as well. These papers were purely theoretical. In contrast, we show that the techniques proposed in this paper are practical as well, and provide significant benefit.

See Appendix A for more details.

2. Tree search

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

2.1. Mixed integer linear programs

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

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. B&B then uses a *variable selection policy* (see Section 2.1.1) to choose a variable x_i of the MILP Q to branch on. Specifically, let Q_i^+ (resp., Q_i^-) be the MILP Q except with the additional constraint that $x_i = 1$ (resp., $x_i = 0$). B&B sets the right (resp., left) child of Q in $\mathcal{T}$ to be a node containing the MILP Q_i^+ (resp., Q_i^-). B&B then tries to "fathom" these leafs: the leaf containing Q_i^+ (resp., Q_i^-) is *fathomed* if: 1) The optimal solution to the LP relaxation of Q_i^+ (resp., Q_i^-) satisfies the constraints of the original MILP Q', 2) The relaxation of Q_i^+ (resp., Q_i^-) is infeasible, so Q_i^+ (resp., Q_i^-) must be infeasible as well, or 3) The objective value of the LP relaxation of Q_i^+ (resp., Q_i^-) is smaller than the objective value of the best known feasible solution, so the optimal solution to Q_i^+ (resp., Q_i^-) is no better than the best known feasible solution. B&B terminates when every leaf has been fathomed. It returns the best known feasible solution, which is optimal. See Algorithm 1 in Appendix B for the pseudocode.

The most common node selection policy is the *best bound policy*. Given a B&B tree, it selects the unfathomed leaf containing the MILP Q with the maximum LP relaxation objective value. Another common policy is the *depth-first policy*, which selects the next unfathomed leaf in the tree in depth-first order. See Example B.2 in Appendix B for an example of B&B in action.

2.1.1. Variable selection in MILP tree search

Variable selection policies typically depend on a real-valued *score* per variable x_i. Specifically, let $\texttt{score}$ 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. 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 common definitions of the function $\texttt{score}$ below. We use the notation Q_i^+ (resp., Q_i^-) to denote the MILP Q with the additional constraint that $x_i = 1$ (resp., $x_i = 0$). If Q_i^+ (resp., Q_i^-) is infeasible, then we set $\breve{c}_Q - \breve{c}_{Q_i^+}$ (resp., $\breve{c}_Q - \breve{c}_{Q_i^-}$) to be some large number greater than $||\boldsymbol{c}||_1$.

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

Linear scoring rule (Linderoth & Savelsbergh, 1999). In this case, $\texttt{score}(\mathcal{T}, Q, i) = (1-\mu) \cdot \max\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\} + \mu \cdot \min\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\}$ where $\mu \in [0, 1]$ is a user-specified parameter.

Product scoring rule (Achterberg, 2009). In this case,

$\texttt{score}(\mathcal{T}, Q, i) = \max\{\breve{c}_Q - \breve{c}_{Q_i^-}, \gamma\} \cdot \max\{\breve{c}_Q - \breve{c}_{Q_i^+}, \gamma\}$ where $\gamma = 10^{-6}$.

Entropic lookahead scoring rule (Gilpin & Sandholm, 2011) Let $e(x) = -x \log_2(x) - (1-x)\log_2(1-x)$ if $x \in (0,1)$ and $e(x) = 0$ if $x \in \{0,1\}$. Set $\texttt{score}(\mathcal{T}, Q, i) = -\sum_{j=1}^{n} (1 - \breve{x}_Q[i]) \cdot e\left(\breve{x}_{Q_i^-}[j]\right) + \breve{x}_Q[i] \cdot e\left(\breve{x}_{Q_i^+}[j]\right)$.

Alternative definitions of the linear and product scoring rules. It is often too slow to compute the differences $\breve{c}_Q - \breve{c}_{Q_i^-}$ and $\breve{c}_Q - \breve{c}_{Q_i^+}$ for every variable, since it requires solving as many as $2n$ LPs. A faster option is to partially solve the LP relaxations of Q_i^- and Q_i^+, starting at $\breve{\boldsymbol{x}}_Q$ and running a small number of simplex iterations. Denoting the new objective values as $\tilde{c}_{Q_i^-}$ and $\tilde{c}_{Q_i^+}$, we can revise the linear scoring rule to be $\texttt{score}(\mathcal{T}, Q, i) = (1-\mu) \cdot \max\{\breve{c}_Q - \tilde{c}_{Q_i^+}, \breve{c}_Q - \tilde{c}_{Q_i^-}\} + \mu \cdot \min\{\breve{c}_Q - \tilde{c}_{Q_i^+}, \breve{c}_Q - \tilde{c}_{Q_i^-}\}$ and we can revise the product scoring rule to be $\texttt{score}(\mathcal{T}, Q, i) = \max\{\breve{c}_Q - \tilde{c}_{Q_i^-}, \gamma\} \cdot \max\{\breve{c}_Q - \tilde{c}_{Q_i^+}, \gamma\}$. Other alternatives to computing $\breve{c}_{Q_i^-}$ and $\breve{c}_{Q_i^+}$ that fit within our framework are *pseudo-cost branching* (Bénichou et al., 1971; Gauthier & Ribière, 1977; Linderoth & Savelsbergh, 1999) and *reliability branching* (Achterberg et al., 2005).

3. Guarantees for data-driven branching

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

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

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

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

3.1. Impossibility results for data-independent approaches

In this section, we focus on MILP tree search and prove that it is *impossible* to find a nearly optimal B&B configuration using a data-independent discretization of the parameters. Specifically, suppose $\texttt{score}_1(\mathcal{T}, Q, i) = \min\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\}$ and $\texttt{score}_2(\mathcal{T}, Q, i) = \max\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\}$ and suppose the cost function $\texttt{cost}(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2)$ measures the size of the tree produced by B&B when using a fixed but arbitrary node selection policy. We would like to learn a nearly optimal convex combination $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ of these two rules with respect to $\texttt{cost}$. Gauthier & Ribière (1977) proposed setting $\mu = 1/2$, Bénichou et al. (1971) and Beale (1979) suggested setting $\mu = 1$, and Linderoth & Savelsbergh (1999) found that $\mu = 2/3$ performs well. Achterberg (2009) found that experimentally, $\mu = 5/6$ performed best when comparing among $\mu \in \{0, 1/2, 2/3, 5/6, 1\}$.

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

Theorem 3.1. *Let* $\texttt{score}_1(\mathcal{T}, Q, i) = \min\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\}$, $\texttt{score}_2(\mathcal{T}, Q, i) = \max\{\breve{c}_Q - \breve{c}_{Q_i^+}, \breve{c}_Q - \breve{c}_{Q_i^-}\}$, *and* $\texttt{cost}(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2)$ *be the size of the tree produced by B&B. For every* a, b *such that* $\frac{1}{3} < a < b < \frac{1}{2}$ *and for all even* $n \geq 6$, *there exists an infinite family of distributions* $\mathcal{D}$ *over MILP instances with* n *variables such that if* $\mu \in [0,1] \setminus (a, b)$, *then* $\mathbb{E}_{Q\sim\mathcal{D}}[\texttt{cost}(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2)] =$

$\Omega(2^{(n-9)/4})$ *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: Theorems 3.2 and D.5. In Theorem 3.2, we prove that for all $\mu^* \in (0, 1)$, there exists an infinite family $\mathcal{F}_{n,\mu^*}$ of MILP instances such that for any $Q \in \mathcal{F}_{n,\mu^*}$, if $\mu \in [0, \mu^*)$, then the scoring rule $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ 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 D.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 4 in Appendix D for an illustration. □

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

Theorem 3.2. *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* $\texttt{cost}(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2)$ *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: maximize $\{\boldsymbol{c} \cdot \boldsymbol{x}$ subject to $2\sum_{i=1}^{n'} x_i = n', \boldsymbol{x} \in \{0, 1\}^{n'}\}$.

We build off of this MILP to create the infinite family $\mathcal{F}_{n,\mu^*}$. Each MILP in $\mathcal{F}_{n,\mu^*}$ combines a hard version of Jeroslow's instance on $n-3$ variables $\{x_1, \ldots, x_{n-3}\}$ and an easy version on 3 variables $\{x_{n-2}, x_{n-1}, x_n\}$. Branch-and-bound only needs to determine that one of these problems is infeasible in order to terminate. 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. □

3.2. Sample complexity guarantees

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

The guarantees in this section apply broadly to a class of scoring rules we call *path-wise* scoring rules. Given a node Q in a search tree $\mathcal{T}$, we denote the path from the root of $\mathcal{T}$ to the node Q as $\mathcal{T}_Q$. The path $\mathcal{T}_Q$ includes all nodes and edge labels from the root of $\mathcal{T}$ to Q. See Figures 7b and 7a in Appendix D for an illustration. We now state the definition of path-wise scoring rules.

Definition 3.3 (Path-wise scoring rule)**.** *The function* $\texttt{score}$ *is a path-wise scoring rule if for all search trees* $\mathcal{T}$*, all nodes* Q *in* $\mathcal{T}$*, and all variables* x_i*,* $\texttt{score}(\mathcal{T}, Q, i) = \texttt{score}(\mathcal{T}_Q, Q, i)$ *where* $\mathcal{T}_Q$ *is the path from the root of* $\mathcal{T}$ *to* Q*. See Figure 7 in Appendix D for an illustration.*

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 2.1.1. 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 2.1.1 and as is our approach in our experiments. In fact, these scoring rules depend only on the node in question, rather than the path from the root to the node. We present our sample complexity bounds for the more general class of path-wise scoring rules because this class captures the level of generality the proof holds for.

In order to prove our generalization guarantees, we make use of the following key structure which bounds the number of search trees branch-and-bound will build on a given instance over the entire range of parameters. In essence, this is a bound on the intrinsic complexity of the algorithm class defined by the range of parameters, and this bound on algorithm class's intrinsic complexity implies strong gener-

alization guarantees. The full proof is in Appendix D.

Lemma 3.4. *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* Q *be an arbitrary problem instance over* n *binary variables. There are* $T \leq 2^{n(n-1)/2} n^n$ *intervals* $I_1, \ldots, I_T$ *partitioning* $[0, 1]$ *where for any interval* I_j*, across all* $\mu \in I_j$*, the scoring rule* $\mu\texttt{score}_1 + (1-\mu)\texttt{score}_2$ *results in the same search tree.*

Proof sketch. We first consider the actions of an alternative algorithm A' which runs exactly like B&B, except it only fathoms nodes if they are integral or infeasible. We prove a variation on Lemma 3.4 for A', not B&B. Using the fact that $\texttt{score}_1$ and $\texttt{score}_2$ are path-wise, we relate the behavior of A' to the behavior of B&B to prove the lemma. □

In Theorem 3.5, we use Lemma 3.4 to bound the pseudo-dimension of the class of cost functions $\mathcal{C} = \{\texttt{cost}\left(\cdot, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2\right) : \mu \in [0, 1]\}$. See Definition D.14 in Appendix D for the definition of pseudo-dimension. It is well-known that pseudo-dimension bounds imply generalization guarantees (Pollard, 1984), as summarized by Theorem D.15 in Appendix D. The proof of Theorem 3.5 is in Appendix D.

Theorem 3.5. *Let* $\texttt{cost}$ *be a tree-constant cost function, let* $\texttt{score}_1$ *and* $\texttt{score}_2$ *be two path-wise scoring rules, and let* $\mathcal{C}$ *be the set of functions* $\{\texttt{cost}\left(\cdot, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2\right) : \mu \in [0, 1]\}$. *Then* $\text{Pdim}(\mathcal{C}) = O\left(n^2\right)$.

In Appendix F.2, we provide pseudo-dimension bounds for d-dimensional parameter spaces.

Learning algorithm. For the case where we wish to learn the optimal tradeoff between two scoring rules, we provide an algorithm in Appendix D that finds the empirically optimal parameter $\hat{\mu}$ given a sample of m problem instances. By Theorems 3.5 and D.15, we know that so long as m is sufficiently large, $\hat{\mu}$ is nearly optimal in expectation. Our algorithm stems from the observation that for any tree-constant cost function $\texttt{cost}$ and any problem instance Q, $\texttt{cost}\left(Q, \mu\texttt{score}_1 + (1-\mu)\texttt{score}_2\right)$ is simple: it is a piecewise-constant function of μ with a finite number of pieces. Given a set of m of problem instances, our ERM algorithm constructs all m piecewise-constant functions, takes their average, and finds the minimizer of that function. In practice, we find that the number of pieces making up this piecewise-constant function is small.

4. Experiments

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

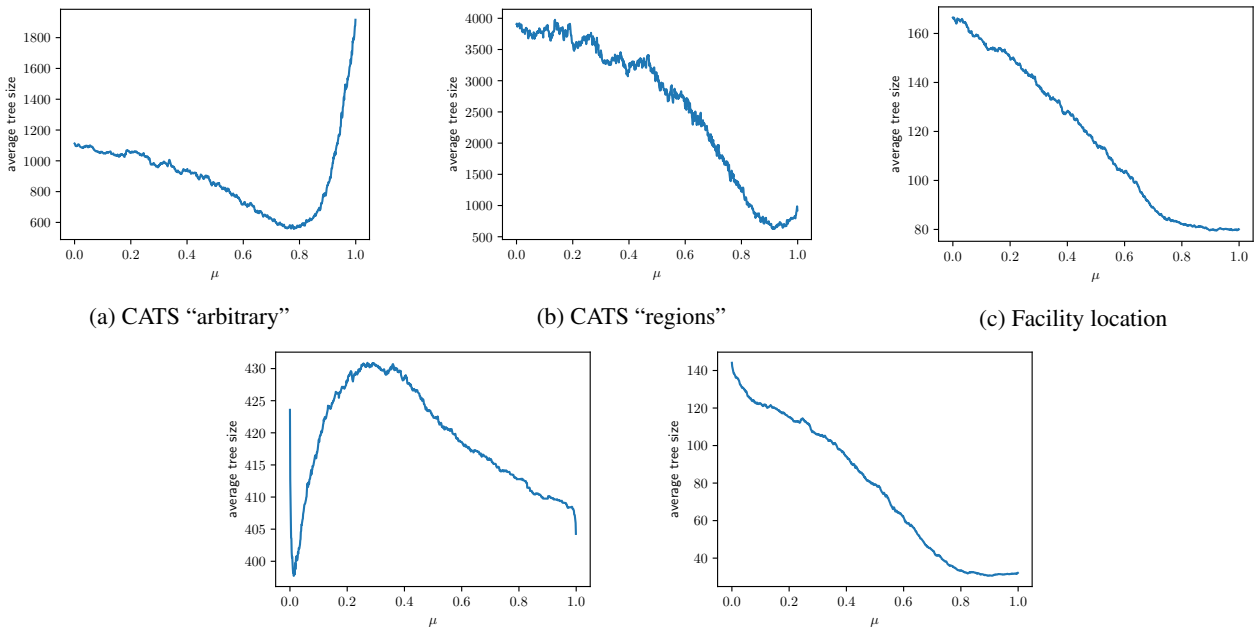
Experimental setup. We use the C API of IBM ILOG CPLEX 12.8.0.0 to override the default variable selection rule using a branch callback. Additionally, our callback performs extra book-keeping to determine a finite set of values for the parameter that give rise to *all* possible B&B trees for a given instance (for the given choice of branching rules that our algorithm is learning to weight). This ensures that there are no good or bad values for the parameter that get skipped; such skipping could be problematic according to our theory in Section 3.1. We run CPLEX exactly once for each possible B&B tree on each instance. Following Khalil et al. (2016); Fischetti & Monaci (2012), and Karzan et al. (2009), we disable CPLEX's cuts and primal heuristics, and we also disable its root-node preprocessing. The CPLEX node selection policy is set to "best bound" (aka. A^* in AI), which is the most typical choice in MILP. All experiments are run on a cluster of 64 c3.large Amazon AWS instances.

For each of the following application domains, Figure 1 shows the average B&B tree size produced for each possible value of the μ parameter for the linear scoring rule, averaged over m independent samples from the distribution.

Combinatorial auctions. We generate $m = 100$ instances of the combinatorial auction winner determination problem under the OR-bidding language (Sandholm, 2002), which makes this problem equivalent to weighted set packing. The problem is NP-complete. We encode each instance as a binary MILP (see Example B.1). We use the Combinatorial Auction Test Suite (CATS) (Leyton-Brown et al., 2000) to generate these instances. We use the "arbitrary" generator with 200 bids and 100 goods and "regions" generator with 400 bids and 200 goods.

Facility location. Suppose there is a set I of customers and a set J of facilities that have not yet been built. The facilities each produce the same good, and each consumer demands one unit of that good. Consumer i can obtain some fraction y_{ij} of the good from facility j, which costs them $d_{ij}y_{ij}$. Moreover, it costs f_j to construct facility j. The goal is to choose a subset of facilities to construct while minimizing total cost. In Appendix E we show how to formulate facility location as a MILP. We generate $m = 500$ instances with 70 facilities and 70 customers each. Each cost d_{ij} is uniformly sampled from $\left[0, 10^4\right]$ and each cost f_j is uniformly sampled from $\left[0, 3 \cdot 10^3\right]$.

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



(a) CATS "arbitrary" (b) CATS "regions" (c) Facility location

(d) Linear separators (e) Clustering

Figure 1. The average tree size produced by B&B when run with the linear scoring rule with parameter μ.

generate $m = 500$ instances with 35 points each and $k = 5$. We set $d(i, i) = 0$ for all i and choose $d(i, j)$ uniformly at random from $[0, 1]$ for $i \neq j$.

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

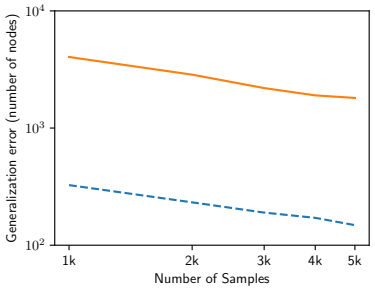
Experimental results. The relationship between the variable selection parameter and the average tree size varies greatly from application to application. This implies that the parameters should be tuned on a per-application basis, and that no parameter value is universally effective. In particular, the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

Data-dependent guarantees. We now explore data-dependent generalization guarantees. To prove our worst-case guarantee Theorem 3.5, we show in Lemma 3.4 that for any MILP instance over n binary variables, there are $T \leq 2^{n(n-1)/2} n^n$ intervals $I_1, \ldots, I_T$ partitioning $[0, 1]$ where for any interval I_j, across all $\mu \in I_j$, the scoring rule $\mu \texttt{score}_1 + (1 - \mu)\texttt{score}_2$ results in the same B&B tree. Our generalization guarantees grow logarithmically in the number of intervals. In practice, we find that the number of intervals partitioning $[0, 1]$ is much smaller than $2^{n(n-1)/2} n^n$. In this section, we take advantage of this data-dependent simplicity to derive stronger generalization guarantees when the number of intervals partitioning $[0, 1]$ is small. To do so, we move from pseudo-dimension to Rademacher complexity (Bartlett & Mendelson, 2002; Koltchinskii, 2001) (see Definition E.1 in Appendix E) since Rademacher complexity implies distribution-dependent guarantees whereas pseudo-dimension implies generalization guarantees that are worst-case over the distribution (see Theorem E.2 in Appendix E). We provide the details in Appendix E. See Figure 2 for a comparison of the worst-case versus data-dependent generalization guarantees. The latter are significantly better.

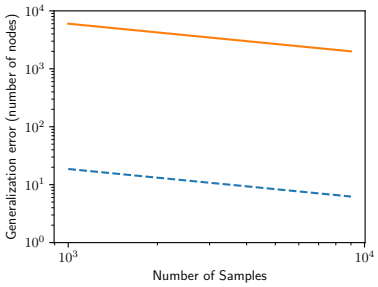
5. Constraint satisfaction problems

In this section, we describe tree search for constraint satisfaction problems. The generalization guarantee from Section 3.2 also applies to tree search in this domain, as we describe in Appendix F.

A constraint satisfaction problem (CSP) is a tuple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of variables,



(a) Linear separators



(b) Clustering

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

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

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.

CSP tree search begins by choosing a variable x_i with domain $D(x_i)$ and building $|D(x_i)|$ branches, each one corresponding to one of the $|D(x_i)|$ possible value assignments of x. Next, a node Q of the tree is chosen, another variable x_j is chosen, and $|D(x_j)|$ branches from Q are build, 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.

As in MIP 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 path-wise policies below.

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

Smallest domain (Haralick & Elliott, 1980). In this case, $\texttt{score}(\mathcal{T}, Q, i) = 1/|D(x_i)|$.

Our theory is for tree search and applies to both MIPs 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.

6. Conclusions and broader applicability

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

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

Acknowledgments

This work was supported in part by the NSF under grants CCF-1422910, CCF-1535967, IIS-1618714, IIS-1718457, IIS-1617590, CCF-1733556, a Microsoft Research Faculty Fellowship, an Amazon Research Award, a NSF Graduate Research Fellowship, and the ARO under award W911NF-17-1-0082.

References

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

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

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

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

Balcan, M.-F., Nagarajan, V., Vitercik, E., and White, C. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. *Conference on Learning Theory (COLT)*, 2017.

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

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

Bénichou, M., Gauthier, J.-M., Girodet, P., Hentges, G., Ribière, G., and Vincent, O. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

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

Buck, R. C. Partition of space. *Amer. Math. Monthly*, 50: 541–544, 1943. ISSN 0002-9890.

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

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

Di Liberto, G., Kadioglu, S., Leo, K., and Malitsky, Y. Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 248(3):943–953, 2016.

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

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

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

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

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

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

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

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

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

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

Kappes, J. H., Speth, M., Reinelt, G., and Schnörr, C. Towards efficient and exact map-inference for large scale discrete computer vision problems via combinatorial optimization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1752–1758. IEEE, 2013.

Karzan, F. K., Nemhauser, G. L., and Savelsbergh, M. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1(4):249–293, 2009.

Khalil, E. B., Bodic, P. L., Song, L., Nemhauser, G., and Dilkina, B. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Khalil, E. B., Dilkina, B., Nemhauser, G., Ahmed, S., and Shao, Y. Learning to run heuristics in tree search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Sandholm, T. Very-large-scale generalized combinatorial multi-attribute auctions: Lessons from conducting $60 billion of sourcing. In Neeman, Z., Roth, A., and Vulkan, N. (eds.), *Handbook of Market Design*. Oxford University Press, 2013.

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

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

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