Improved Learning Bounds for Branch-and-Cut

Maria-Florina Balcan[1], Siddharth Prasad[1], Tuomas Sandholm[1,2,3,4], and Ellen Vitercik[5]

[1] School of Computer Science, Carnegie Mellon University
[2] Optimized Markets, Inc.
[3] Strategic Machine, Inc.
[4] Strategy Robot, Inc.
[5] EECS Department, UC Berkeley
{ninamf,sprasad2,sandholm}@cs.cmu.edu, vitercik@berkeley.edu

Abstract. Branch-and-cut is the most widely used algorithm for solving integer programs, employed by commercial solvers like CPLEX and Gurobi. Branch-and-cut has a wide variety of tunable parameters that have a huge impact on the size of the search tree that it builds, but are challenging to tune by hand. An increasingly popular approach is to use machine learning to tune these parameters: using a *training set* of integer programs from the application domain at hand, the goal is to find a configuration with strong predicted performance on future, unseen integer programs from the same domain. If the training set is too small, a configuration may have good performance over the training set but poor performance on future integer programs. In this paper, we prove *sample complexity guarantees* for this procedure, which bound how large the training set should be to ensure that for any configuration, its average performance over the training set is close to its expected future performance. Our guarantees apply to parameters that control the most important aspects of branch-and-cut: node selection, branching constraint selection, and cutting plane selection, and are sharper and more general than those found in prior research [6, 8].

Keywords: Branch-and-cut · Sample complexity · Machine learning.

1 Introduction

Branch-and-cut (B&C) is a powerful algorithmic paradigm that is the backbone of all modern integer-program (IP) solvers. The main components of B&C can be tuned and tweaked in a myriad of different ways. The fastest commercial integer program solvers like CPLEX and Gurobi employ an array of heuristics to make decisions at every stage of B&C to reduce the solving time as much as possible, and give the user freedom to tune the multitude of parameters influencing the search through the space of feasible solutions. However, tuning the parameters that control B&C in a principled way is an inexact science with little to no formal mathematical guidelines. A rapidly growing line of work studies machine-learning approaches to speeding up the various aspects of B&C—in

particular investigating whether high-performing B&C parameter configurations can be learned from a *training set* of typical IPs from the particular application at hand. Complementing the substantial number of experimental approaches using machine learning for B&C, a nascent generalization theory has developed in parallel that aims to provide a rigorous theoretical foundation for how well any B&C configuration learned from training IP data will perform on new unseen IPs [6, 8]. In particular, this line of theoretical research provides *sample complexity guarantees* that bound how large the training set should be to ensure that *no matter how the parameters are configured*, the average performance of branch-and-cut over the training set is close to its expected future performance. With too small a training set, a configuration may have strong average performance over the training set but terrible expected performance on future IPs. In this paper, we expand and improve upon this theory to develop a wider and sharper handle on the learnability of the various key components of B&C.

1.1 Summary of main contributions

Our main contribution is a formalization of a general model of tree search, presented in Section 2.1, that allows us to improve and generalize prior results on the sample complexity of tuning B&C. In this model, the algorithm repeatedly chooses a leaf node of the search tree, performs a series of actions (for example, a cutting plane to apply and a constraint to branch on), and adds children to that leaf in the search tree. The algorithm will also fathom nodes when applicable. The node and action selection are governed by *scoring rules*, which assign a real-valued score to each node and possible action. For example, a node-selection scoring rule might equal the objective value of the node's LP relaxation. We focus on general tree search with *path-wise* scoring rules. At a high level, a score of a node or action is path-wise if its value only depends on information contained along the path between the root and that node, as is often the case in B&C. Many commonly used scoring rules are path-wise.

In Section 3 we prove our main structural result: for any IP, the tree search parameter space can be partitioned into a finite number of regions such that in any one region, the resulting search tree is fixed. By analyzing the complexity of this partition, we prove our sample complexity bound. In particular, we relate the complexity of these partitions to the *pseudo-dimension* of the set of functions that measure the performance of B&C as a function of the input IP, given any fixed parameter configuration. Pseudo-dimension (defined in Section 3 is a combinatorial notion from machine learning theory that measures the *intrinsic complexity* of a set of functions. At a high level, it measures the ability of functions in a class to match complex patterns. Classic results from learning theory then allow us to translate our pseudo-dimension bound into a sample complexity guarantee [3], capturing the intuition that the more complex patterns one can fit (i.e., the larger the pseudo-dimension is), the more samples we need to generalize.

Finally, in Section 4, we show how this general model of tree search captures a wide array of B&C components—including node selection, general branching constraint selection, and cutting plane selection, simultaneously—and present the implications of our sample complexity analysis.

Our model significantly generalizes over that of Balcan et al. [6], who only studied path-wise scoring rules for single-variable selection for branching. In contrast, we are able to handle node selection, general branching constraint selection, and cutting plane selection. Our results also improve over those of subsequent research by Balcan et al. [8] for the case of path-wise scores. While their techniques apply as broadly as ours, their analysis is very general, not taking advantage of any inherent tree structure or using the path-wise assumption, thus leading to large sample complexity bounds.

1.2 Additional related research

A growing body of research has studied how machine learning can be used to speed up the time it takes to solve integer programs, primarily from an empirical perspective, whereas we study this problem from a theoretical perspective. This line of research has included general parameter tuning procedures [e.g., 20, 21, 23, 31], which are not restricted to any one aspect of B&C. Researchers have also honed in on specific aspects of tree search and worked towards improving those using machine learning. These include variable selection [2, 6, 11, 13, 16, 25], general branching constraint selection [35], cut selection [8, 19, 31, 33], node selection [18, 30], and heuristic scheduling [9, 26]. Machine learning approaches to large neighborhood search have also been used to speed up solver runtimes [32].

This paper contributes to a line of research that provides sample complexity guarantees for algorithm configuration, often by using structure exhibited by the algorithm's performance as a function of its parameters [e.g., 4–8, 17]. This line of research has studied algorithms for clustering [e.g., 7], computational biology [5], and integer programming [6, 8], among other computational problems. The main contribution of this paper is to provide a sharp yet general analysis of the performance of tree search as a function of its parameters.

A related line of research provides algorithm configuration procedures with provable guarantees that are agnostic to the specific algorithm that is being configured [e.g., 27, 34] and are particularly well-suited for algorithms with a finite number of possible configurations (though they can be applied to algorithms with infinite parameter spaces by randomly sampling a finite set of configurations).

2 Main tree search model

In this section we present our general tree search model and situate it within the framework of sample complexity. Balcan et al. [8] studied the sample complexity of a much more general formulation of a tunable search algorithm without any inherent tree structure. Our formulation explicitly builds a tree.

Algorithm 1 Tree search

```
Input: Root node $Q$, depth limit $\Delta$
1: Initialize $\mathcal{T} = Q$.
2: while $\mathcal{T}$ contains an unfathomed leaf do
3:     Select a leaf $Q$ of $\mathcal{T}$ that maximizes $\texttt{nscore}(\mathcal{T}, Q)$.
4:     if $\texttt{depth}(Q) = \Delta$ or $\texttt{fathom}(\mathcal{T}, Q, \texttt{None})$ then
5:         Fathom $Q$.
6:     else
7:         Select an action $A \in \texttt{actions}(\mathcal{T}, Q)$ that maximizes $\texttt{ascore}(\mathcal{T}, Q, A)$.
8:         if $\texttt{fathom}(\mathcal{T}, Q, A)$ then
9:             Fathom $Q$.
10:        else if $\texttt{children}(\mathcal{T}, Q, A) = \emptyset$ then
11:            Fathom $Q$.
12:        else
13:            Add all nodes in $\texttt{children}(\mathcal{T}, Q, A)$ to $\mathcal{T}$ as children of $Q$.
```

2.1 General model of tree search

Tree search starts with a root node. In each round of tree search, a leaf node Q is selected. At this node, one of three things may occur: (1) Q is fathomed, meaning it is never visited again, (2) some action is taken at Q, and then it is fathomed, or (3) some action is taken at Q, and then some number of children nodes of Q are added to the tree. (For example, an action might represent a decision about which variable to branch on.) This process repeats until the tree has no unfathomed leaves. More formally, there are functions `actions`, `children`, and `fathom` prescribing how the search proceeds. Given a partial tree $\mathcal{T}$ and a leaf Q of $\mathcal{T}$, $\texttt{actions}(\mathcal{T}, Q)$ outputs a set of actions available at Q. Given a partial tree $\mathcal{T}$, a leaf Q of $\mathcal{T}$, and an action $A \in \texttt{actions}(\mathcal{T}, Q)$: $\texttt{fathom}(\mathcal{T}, Q, A) \in \{\texttt{true}, \texttt{false}\}$ is a Boolean function used to determine when to fathom a leaf Q of $\mathcal{T}$ given that action $A \in \texttt{actions}(\mathcal{T}, Q) \cup \{\texttt{None}\}$ was taken at Q, and $\texttt{children}(\mathcal{T}, Q, A)$ outputs a (potentially empty) list of nodes representing the children of Q to be added to the search tree given that action A was taken at Q. Finally, $\texttt{nscore}(\mathcal{T}, Q)$ is a node-selection score that outputs a real-valued score for each leaf of $\mathcal{T}$, and $\texttt{ascore}(\mathcal{T}, Q, A)$ is an action-selection score that outputs a real-valued score for each action $A \in \texttt{actions}(\mathcal{T}, Q)$. These scores are heuristics that are meant to indicate the quality of exploring a node or performing an action. Many aspects of B&C are governed by scoring rules [1]. For example, $\texttt{nscore}(\mathcal{T}, Q)$ might equal the objective value of the LP relaxation of the IP represented by the node Q. If A is a cutting plane, then $\texttt{ascore}(\mathcal{T}, Q, A)$ might equal the distance between A and the optimal solution to the LP relaxation. Algorithm 1 is a formal description of tree search using these functions.

The key condition that enables us to derive stronger sample complexity bounds compared to prior research is the notion of a *path-wise* function, which was also used in prior research but only in the context of variable selection [6].

Definition 1 (Path-wise functions). *A function f on tree-leaf pairs is path-wise if for all $\mathcal{T}$ and $Q \in \mathcal{T}$, $f(\mathcal{T}, Q) = f(\mathcal{T}_Q, Q)$, where $\mathcal{T}_Q$ is the path from* *the root of $\mathcal{T}$ to Q. A function g on tree-leaf-action triples is path-wise if for all A, the function $f_A(\mathcal{T}, Q) := g(\mathcal{T}, Q, A)$ is path-wise.*

Many commonly-used scoring rules are path-wise. For example, scoring rules are often functions of the LP relaxation of the IP represented by a given node, and these scoring rules are path-wise. We assume that $\texttt{actions}, \texttt{ascore}, \texttt{nscore}$ and $\texttt{children}$ are path-wise, though $\texttt{fathom}$ is not necessarily path-wise.

No one scoring rule is optimal across all application domains, and prior research on variable selection has shown that it can be advantageous to adapt the scoring rule to the application domain at hand [6]. To this end, Algorithm 1 can be tuned by two parameters $\mu \in [0, 1]$ and $\lambda \in [0, 1]$ that control action selection and node selection, respectively. Given two fixed path-wise action-selection scores $\texttt{ascore}_1$ and $\texttt{ascore}_2$, we may define a new score by $\texttt{ascore}_\mu(\mathcal{T}, Q) = \mu \cdot \texttt{ascore}_1(\mathcal{T}, Q) + (1-\mu) \cdot \texttt{ascore}_2(\mathcal{T}, Q)$. Similarly, given two fixed path-wise node-selection scores $\texttt{nscore}_1$ and $\texttt{nscore}_2$, we may define a new score by $\texttt{nscore}_\lambda(\mathcal{T}, Q, A) = \lambda \cdot \texttt{nscore}_1(\mathcal{T}, Q, A) + (1 - \lambda) \cdot \texttt{nscore}_2(\mathcal{T}, Q, A)$. Then, if $\texttt{nscore}_\lambda$ and $\texttt{ascore}_\mu$ are used as the scores in Algorithm 1, we can view the behavior of tree search as a function of μ and λ.

Finally, we assume there exists $b, k \in \mathbb{N}$ such that $|\texttt{actions}(\mathcal{T}, Q)| \le b$ for any $Q \in \mathcal{T}$, and $|\texttt{children}(\mathcal{T}, Q, A)| \le k$ for all Q, A.

2.2 Problem formulation

We now define the notion of a *sample complexity bound* more formally. Let $\mathcal{Q}$ denote the domain of possible input root nodes Q to Algorithm 1 (for example, the set of all IPs with n variables and m constraints). We assume there is some unknown distribution $\mathcal{D}$ over $\mathcal{Q}$. In the IP setting, $\mathcal{D}$ could represent, for example, typical scheduling IP instances solved by an airline company. The *sample complexity* of a class of real valued functions $\mathcal{F} = \{f : \mathcal{Q} \to \mathbb{R}\}$ is the minimum number of independent samples required from $\mathcal{D}$ so that with high probability over the samples, the empirical value of f on the samples is a good approximation of the expected value of f over $\mathcal{D}$, uniformly over all $f \in \mathcal{F}$. Formally, given an error parameter ε and confidence parameter δ, the sample complexity $N_{\mathcal{F}}(\varepsilon, \delta)$ is the minimum $N_0 \in \mathbb{N}$ such that for any $N \ge N_0$,

$$\Pr_{Q_1,\ldots,Q_N \sim \mathcal{D}} \left(\sup_{f \in \mathcal{F}} \left| \tfrac{1}{N} \sum_{i=1}^{N} f(Q_i) - \mathbb{E}_{Q \sim \mathcal{D}}[f(Q)] \right| \le \varepsilon \right) \ge 1 - \delta$$

for all distributions $\mathcal{D}$ supported on $\mathcal{Q}$.

In the context of Algorithm 1, we study families of *tree-constant* cost functions. A cost function $\texttt{cost} : \mathcal{Q} \to \mathbb{R}$ is tree constant if $\texttt{cost}(Q)$ only depends on the tree built by Algorithm 1 on input Q (an example is tree size). Let $\texttt{cost}_{\mu,\lambda}(Q)$ denote this cost when Algorithm 1 is run using the scores $\texttt{ascore}_\mu = \mu \cdot \texttt{ascore}_1 + (1 - \mu) \cdot \texttt{ascore}_2$ and $\texttt{nscore}_\lambda = \lambda \cdot \texttt{nscore}_1 + (1 - \lambda) \cdot \texttt{nscore}_2$. We study the sample complexity of $\mathcal{F} = \{\texttt{cost}_{\mu,\lambda} : \mu, \lambda \in [0, 1]\}$.

A strength of these guarantees is that they apply no matter how the parameters are tuned: optimally or suboptimally, manually or automatically. For *any* configuration, these guarantees bound the difference between average performance over the training set and expected future performance on unseen IPs.

3 Sample complexity bounds for tree search

In order to derive our sample complexity guarantees, we first prove a key structural property: the behavior of Algorithm 1 is piecewise constant as a function of the node-selection score parameter λ and the action-selection score parameter μ. We give a high-level outline of our approach. We first assume that the conditional checks whether $\texttt{fathom}(\mathcal{T}, Q, \cdot) = \texttt{true}$ (lines 4 and 8) are suppressed. Let $\mathcal{A}'$ denote Algorithm 1 without these checks (so $\mathcal{A}'$ fathoms a node if and only if the depth limit is reached or if the node has no children). The behavior of $\mathcal{A}'$ as a function of μ and λ can be shown to be piecewise constant using the same argument as in Claim 3.4 of Balcan et al. [6]. Given this, our first main technical contribution (Lemma 1) is a generalization of Claim 3.5 of Balcan et al. [6] that relates the behavior of $\mathcal{A}'$ to Algorithm 1. The argument in Balcan et al. [6] is specific to branching, but by extracting the main ideas in their argument we are able to extend their result to our much more general setting. Our second main technical contribution (Lemma 3) is to establish piecewise structure when the node-selection score is controlled by $\lambda \in [0, 1]$. The main reason for this auxiliary step of analyzing $\mathcal{A}'$ is due to the fact that `fathom` is *not* necessarily a path-wise function, and can depend on the state of the entire tree.

Lemma 1. *Fix $\mu \in [0, 1]$. Let $\mathcal{T}$ and $\mathcal{T}'$ be the trees built by Algorithm 1 and $\mathcal{A}'$, respectively, using the action-selection score $\mu \cdot \texttt{ascore}_1 + (1 - \mu) \cdot \texttt{ascore}_2$. Let Q be any node in $\mathcal{T}$, and let $\mathcal{T}_Q$ be the path from the root of $\mathcal{T}$ to Q. Then, $\mathcal{T}_Q$ is a rooted subtree of $\mathcal{T}'$, no matter what node selection policy is used.*

Proof. Let t denote the length of the path $\mathcal{T}_Q$. Let $\mathcal{T}_Q$ be comprised of the sequence of nodes $(Q_1, \ldots, Q_t)$ such that Q_1 is the root of $\mathcal{T}$, $Q_t = Q$, and for $\tau \in \{1, \ldots, t-1\}$ $Q_{\tau+1} \in \texttt{children}(\mathcal{T}_{Q_\tau}, Q_\tau, A_\tau)$ where $A_\tau \in \texttt{actions}(\mathcal{T}_{Q_\tau}, Q_\tau)$ is the action selected by Algorithm 1 at node Q_τ. We show that $(Q_1, \ldots, Q_t)$ is a rooted path in $\mathcal{T}'$ as well.

Suppose for the sake of contradiction that this is not the case. Let $\tau \in \{2, \ldots, t\}$ be the minimal index such that $(Q_1, \ldots, Q_{\tau-1})$ is a rooted path in $\mathcal{T}'$, but there is no edge in $\mathcal{T}'$ from $Q_{\tau-1}$ to node Q_τ. There are two possible cases:

Case 1. $Q_{\tau-1}$ was fathomed by $\mathcal{A}'$. This case is trivially not possible since whenever $\mathcal{A}'$ fathoms a node, so does Algorithm 1 (recall $\mathcal{A}'$ was defined by suppressing fathoming conditions of Algorithm 1).

Case 2. $Q_\tau \notin \texttt{children}(\mathcal{T}', Q_{\tau-1}, A'_{\tau-1})$ where $A'_{\tau-1}$ is the action taken by $\mathcal{A}'$ at node $Q_{\tau-1}$. In this case, if $\texttt{children}(\mathcal{T}', Q_{\tau-1}, A'_{\tau-1}) = \emptyset$, then $Q_{\tau-1}$ would be fathomed by $\mathcal{A}'$, which cannot happen by the first case. Otherwise, if $\texttt{children}(\mathcal{T}', Q_{\tau-1}, A'_{\tau-1}) \neq \emptyset$, we show that we arrive at a contradiction due to the fact that the scoring rules, action-set functions, and children functions are all path-wise. Let $A'_{\tau-1}$ denote the action taken by $\mathcal{A}'$ at $Q_{\tau-1}$, and let

$A_{\tau-1}$ denote the action taken by Algorithm 1 at $Q_{\tau-1}$. Since `actions` is path-wise, $\texttt{actions}(\mathcal{T}, Q_{\tau-1}) = \texttt{actions}(\mathcal{T}_{Q_{\tau-1}}, Q_{\tau-1}) = \texttt{actions}(\mathcal{T}', Q_{\tau-1})$. Since $\texttt{ascore}_1$ and $\texttt{ascore}_2$ are path-wise, we have

$$\begin{aligned}&\mu \cdot \texttt{ascore}_1(\mathcal{T}, Q_{\tau-1}, A) + (1-\mu)\cdot \texttt{ascore}_2(\mathcal{T}, Q_{\tau-1}, A)\\ &\quad= \mu \cdot \texttt{ascore}_1(\mathcal{T}_{Q_{\tau-1}}, Q_{\tau-1}, A) + (1-\mu)\cdot \texttt{ascore}_2(\mathcal{T}_{Q_{\tau-1}}, Q_{\tau-1}, A)\\ &\quad= \mu \cdot \texttt{ascore}_1(\mathcal{T}', Q_{\tau-1}, A) + (1-\mu)\cdot \texttt{ascore}_2(\mathcal{T}', Q_{\tau-1}, A).\end{aligned}$$

for all actions $A \in \texttt{actions}(\mathcal{T}_{Q_{\tau-1}}, Q_{\tau-1})$. Therefore Algorithm 1 and $\mathcal{A}'$ choose the same action at node Q_{t-1}, that is, $A_{\tau-1} = A'_{\tau-1}$. Finally, since `children` is path-wise, we have $\texttt{children}(\mathcal{T}, Q_{\tau-1}, A_{\tau-1}) = \texttt{children}(\mathcal{T}_{Q_{\tau-1}}, Q_{\tau-1}, A_{\tau-1}) = \texttt{children}(\mathcal{T}', Q_{\tau-1}, A_{\tau-1})$. Since $Q_\tau \in \texttt{children}(\mathcal{T}, Q_{\tau-1}, A_{\tau-1})$, this is a contradiction, which completes the proof. □

We use the following generalization of Claim 3.4 of Balcan et al. [6] that shows the behavior of $\mathcal{A}'$ is piecewise constant. While their argument only applies to single-variable branching, our key insight is that the same reasoning can be readily adapted to handle any actions (including general branching constraints and cutting planes). The structure of our proof (which we defer to the full version of the paper) is identical, but is modified to work in our more general setting.

Lemma 2. *Let $\texttt{ascore}_1$ and $\texttt{ascore}_2$ be two path-wise action-selection scores. Fix the input root node Q. There are $T \le k^{\Delta(\Delta-1)/2} b^{\Delta}$ subintervals $I_1, \dots, I_T$ partitioning $[0,1]$ where for any subinterval I_t, the action-selection score $\mu \cdot \texttt{ascore}_1 + (1-\mu)\cdot \texttt{ascore}_2$ results in the same tree built by $\mathcal{A}'$ for all $\mu \in I_t$, no matter what node selection policy is used.*

We now prove our main structural result for Algorithm 1.

Lemma 3. *Let $\texttt{ascore}_1$ and $\texttt{ascore}_2$ be path-wise action-selection scores and let $\texttt{nscore}_1$ and $\texttt{nscore}_2$ be path-wise node-selection scores. Fix the input root node Q. There are $T \le k^{\Delta(9+\Delta)} b^{\Delta}$ rectangles partitioning $[0,1]^2$ such that for any rectangle R_t, the node-selection score $\lambda \cdot \texttt{nscore}_1 + (1-\lambda)\cdot \texttt{nscore}_2$ and the action-selection score $\mu \cdot \texttt{ascore}_1 + (1-\mu)\cdot \texttt{ascore}_2$ result in the same tree built by Algorithm 1 for all $(\mu, \lambda) \in R_t$.*

Proof. By Lemma 2, there is a partition of $[0,1]$ into subintervals $I_1 \cup \cdots \cup I_T$ such that for all μ within a given subinterval, the tree built by $\mathcal{A}'$ is invariant (independent of the node-selection score). Fix a subinterval I_t of this partition. Let $\mathcal{T}$ denote the tree built by Algorithm 1. For each node $Q \in \mathcal{T}$, let $\mathcal{T}_Q$ denote the path from the root to Q in $\mathcal{T}$. Since $\texttt{nscore}_1$ is path-wise, for any tree $\mathcal{T}'$ containing $\mathcal{T}_Q$ as a rooted path, $\texttt{nscore}_1(\mathcal{T}', Q) = \texttt{nscore}_1(\mathcal{T}_Q, Q)$. The same holds for $\texttt{nscore}_2$. For every pair of nodes $Q_1, Q_2 \in \mathcal{T}$, let $\lambda(Q_1, Q_2) \in [0,1]$ denote the unique solution to

$$\begin{aligned}&\lambda \cdot \texttt{nscore}_1(\mathcal{T}_{Q_1}, Q_1) + (1-\lambda)\cdot \texttt{nscore}_2(\mathcal{T}_{Q_1}, Q_1)\\ &\qquad\qquad= \lambda \cdot \texttt{nscore}_1(\mathcal{T}_{Q_2}, Q_2) + (1-\lambda)\cdot \texttt{nscore}_2(\mathcal{T}_{Q_2}, Q_2),\end{aligned}$$

if it exists (if there are either (1) no solutions or (2) infinitely many solutions, set $\lambda(Q_1, Q_2) = 0$). The thresholds $\lambda(Q_1, Q_2)$ for every pair of nodes $Q_1, Q_2 \in \mathcal{T}$ partition $[0, 1]$ into subintervals such that for all λ within a given subinterval, the total order over the nodes of $\mathcal{T}$ induced by $\texttt{nscore}_\lambda$ is invariant. In particular, this means that the node selected by each iteration of Algorithm 1 is invariant. Let $J_1 \cup \cdots \cup J_S$ denote these subintervals induced by the thresholds over all subinterval $I_t \in \{I_1, \ldots, I_T\}$ established in Lemma 2.

We now show that this implies that the tree built by Algorithm 1 is invariant over all (μ, λ) within a given rectangle $I_t \times J_s$. Fix some rectangle $I_t \times J_s$. We proceed by induction on the iterations (of the while loop) of Algorithm 1. For the base case (iteration 0, before entering the while loop), the tree consists of only the root, so the hypothesis trivially holds. Now, suppose the statement holds up until the jth iteration, for some j. We analyze each line of Algorithm 1 to show that the behavior of the $j+1$st iteration is independent of $(\mu, \lambda) \in I_t \times J_s$. First, since J_s determines the node selected at each iteration (as argued above), the node selected on the $j + 1$st iteration (line 3) is fixed, independent of $(\mu, \lambda) \in I_t \times J_s$. Denote this node by Q. Thus, whether $\texttt{depth}(Q) = \varDelta$ is independent of $(\mu, \lambda) \in I_t \times J_s$, and similarly whether $\texttt{fathom}(\mathcal{T}, Q, \texttt{None}) = \texttt{true}$ is independent of $(\mu, \lambda) \in I_t \times J_s$ (line 4). This implies that whether or not Q is fathomed at this stage is independent of $(\mu, \lambda) \in I_t \times J_s$. If Q was fathomed, we are done. Otherwise, we argue that the action selected at line 7 is invariant over $(\mu, \lambda) \in I_t \times J_s$. By Lemma 2, $\mathcal{A}'$ builds the same tree for all $\mu \in I_t$. Let $\mathcal{T}_Q$ denote the path from the root to Q in this tree. By Lemma 1, $\mathcal{T}_Q$ is the path from the root to Q in the tree built by Algorithm 1 as well. The action selected at Q by $\mathcal{A}'$ is invariant over $\mu \in I_t$ (by Lemma 2). Therefore, since $\texttt{actions}$, $\texttt{ascore}_1$, and $\texttt{ascore}_2$ are path-wise, the action A selected by Algorithm 1 at Q is invariant over $\mu \in I_t$. Finally, $\texttt{fathom}(\mathcal{T}, Q, A)$ and $\texttt{children}(\mathcal{T}, Q, A)$ are completely determined, so the execution of the remaining conditional statement (line 8 to line 13) is invariant over $(\mu, \lambda) \in I_t \times J_s$. Thus, the entire iteration of Algorithm 1 is invariant over $(\mu, \lambda) \in I_t \times J_s$, which completes the induction.

Finally, we count the total number of rectangles in our partition of $[0, 1]^2$. For each interval I_t in the partition established in Lemma 2, we obtained a partition of $I_t \times [0, 1]$ into rectangles induced by at most $\binom{|\mathcal{T}|}{2}$ thresholds, which consists of at most at most

$$1 + \binom{(k^{\varDelta+1} - 1)/(k - 1)}{2} \leq 1 + \left(\frac{k^{\varDelta+1} - 1}{k - 1}\right)^2 \leq k^{5\varDelta}$$

subintervals. Accounting for every interval $I_t \in \{I_1, \ldots, I_T\}$ in the partition from Lemma 2, we get a total of $Tk^{5\varDelta} \leq k^{\varDelta(9+\varDelta)/2} b^{\varDelta}$ rectangles, as desired. □

We now bound the sample complexity of the collection $\mathcal{F} = \{\texttt{cost}_{\mu,\lambda} : (\mu, \lambda) \in [0, 1]^2\}$ where $\texttt{cost}$ is any tree-constant function, such as tree size. We do this by bounding the *pseudo-dimension* of $\mathcal{F}$, which is a combinatorial measure of intrinsic complexity of a class of real valued functions. The pseudo-dimension of $\mathcal{F}$, denoted by $\mathrm{Pdim}(\mathcal{F})$, is the largest positive integer N such that there exist N nodes $Q_1, \ldots, Q_N \in \mathcal{Q}$ and N thresholds $r_1, \ldots, r_N \in \mathbb{R}$ such that

$|\{(\mathrm{sign}(f(Q_1)-r_1),\dots,\mathrm{sign}(f(Q_N)-r_N)) : f \in \mathcal{F}\}| = 2^N$. A well-known result in learning theory states that if functions in $\mathcal{F}$ have bounded range $[-H,H]$, then $N_{\mathcal{F}}(\varepsilon,\delta) = O(\frac{H^2}{\varepsilon^2}(\mathrm{Pdim}(\mathcal{F})+\ln(\frac{1}{\delta})))$ [3]. When each function in $\mathcal{F}$ maps to $\{0,1\}$, the pseudo-dimension is more commonly referred to as the *VC dimension*.

Bounding the pseudo-dimension is a simple instantiation of the general framework provided by Balcan et al. [5] with the piecewise structure established in Lemma 3. Balcan et al.'s [5] main result gives pseudo-dimension bounds for families of piecewise structured functions in terms of the VC dimension of the class of 0/1 classifiers defining the boundaries of the functions, the number of classifiers defining the boundaries, and the pseudo-dimension of the family of functions when restricted to each piece. (Strictly, this result is in terms of the dual classes of the boundary and piece functions. However, since the dual class of all linear separators is the set of all linear separators, we omit this detail for simplicity.)

Theorem 1. *Let $\texttt{cost}(Q)$ be any tree-constant cost function, and let $\texttt{cost}_{\mu,\lambda}(Q)$ be the cost of the tree built by Algorithm 1 on input root node Q using action-selection score parameterized by μ and node-selection score parameterized by λ. Then,* $\mathrm{Pdim}(\{\texttt{cost}_{\mu,\lambda}\}) = O(\Delta^2 \log k + \Delta \log b)$.

Proof. By Lemma 3, there are at most $T = k^{\Delta(9+\Delta)}b^{\Delta}$ rectangles partitioning $[0,1]^2$ such that for a fixed input node Q, $\texttt{cost}_{\mu,\lambda}(Q)$ is constant over each rectangle as a function of μ,λ. These T rectangles can be defined by T thresholds on $[0,1]$ corresponding to μ and T thresholds on $[0,1]$ corresponding to λ. Thus, the T rectangles can be identified by $T^2 = k^{2\Delta(9+\Delta)}b^{2\Delta}$ linear separators in $\mathbb{R}^2$. The VC dimension of linear separators in $\mathbb{R}^2$ is $O(1)$. The pseudo-dimension of the set of constant functions is also $O(1)$. Plugging these quantities into the main theorem of Balcan et al. [5] yields the theorem statement. □

3.1 Multiple actions

Theorem 1 can be easily generalized to the case where there are multiple actions of different types taken at each node of Algorithm 1. Specifically, there are now d path-wise action-set functions $\texttt{actions}_1,\dots,\texttt{actions}_d$, and at line 7 of Algorithm 1 we take one action of each type, that is, we select action $A_1 \in \texttt{actions}_1(\mathcal{T},Q)$, $A_2 \in \texttt{actions}_2(\mathcal{T},Q)$, and so on. The functions `fathom` and `children` then depend on all d actions taken at node Q. We assume that there are two scoring rules $\texttt{score}_1^i$ and $\texttt{score}_2^i$ for each action type $i = 1,\dots,d$. Algorithm 1 can then be parameterized by $(\boldsymbol{\mu},\lambda)$, where $\boldsymbol{\mu} \in \mathbb{R}^d$ is a vector of parameters controlling each action, so the ith action is selected to maximize $\mu_i \cdot \texttt{ascore}_1^i + (1-\mu_i)\cdot \texttt{ascore}_2^i$. Then, as long as $d = O(1)$, we get the same pseudo-dimension bound. We assume b is a uniform upper bound on the size of $\texttt{actions}_i$ for any i. The proof is nearly identical, and we defer it to the full version of the paper.

Theorem 2. *Let $\texttt{cost}(Q)$ be any tree-constant cost function, and let $\texttt{cost}_{\boldsymbol{\mu},\lambda}(Q)$ be the cost of the tree built by Algorithm 1 on input root node Q using action-selection scores parameterized by $\boldsymbol{\mu} \in \mathbb{R}^d$, where $d = O(1)$, and node-selection score parameterized by λ. Then,* $\mathrm{Pdim}(\{\texttt{cost}_{\boldsymbol{\mu},\lambda}\}) = O(\Delta^2 \log k + \Delta \log b)$.

4 Branch-and-cut for integer programming

We now instantiate our main results with the three main components of the B&C algorithm: branching, cutting planes, and node selection, used to solve IPs $\max\{\boldsymbol{c}^T\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0, \boldsymbol{x} \in \mathbb{Z}^n\}$ where $\boldsymbol{c} \in \mathbb{R}^n$, $A \in \mathbb{Z}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Z}^m$. The function $\texttt{fathom}(\mathcal{T}, Q, A)$ outputs $\texttt{true}$ if after having taken action A the LP relaxation at Q is integral, infeasible, or worse than the best integral solution found so far in $\mathcal{T}$. The function $\texttt{children}(\mathcal{T}, Q, A)$ outputs the two subproblems generated by the branching procedure on the IP at Q after having taken action A. For simplicity we refer only to IPs, but everything in our discussion applies to mixed IPs as well. In our model of tree search, node selection is controlled by λ. Cutting planes and branching are types of actions and controlled by μ.

4.1 Branching

Our result generalizes the sample complexity bound of Balcan et al. [6] that only applied for scoring rules to choose single variables to branch on. In that setting, $\texttt{actions}(\mathcal{T}, Q) = \{1, \dots, n\}$ and $\texttt{children}(\mathcal{T}, Q, i)$ consists of the two subproblems derived by adding the constraints $\boldsymbol{x}[i] \leq \lfloor \boldsymbol{x}^*_{\mathsf{LP}}[i] \rfloor$ and $\boldsymbol{x}[i] \geq \lceil \boldsymbol{x}^*_{\mathsf{LP}}[i] \rceil$ to Q, respectively, where $\boldsymbol{x}^*_{\mathsf{LP}}$ is the solution to the LP relaxation at Q. Throughout this section we assume $\Delta = O(n)$, as is the case with single-variable branching.

Multivariable branching constraints. It is well known that allowing for more general generation of branching constraints can result in smaller B&C trees. Gilpin and Sandholm [14] studied multivariable branches of the form $\sum_{i\in S} \boldsymbol{x}[i] \leq \left\lfloor \sum_{i\in S} \boldsymbol{x}^*_{\mathsf{LP}}[i] \right\rfloor$, $\sum_{i\in S} \boldsymbol{x}[i] \geq \left\lceil \sum_{i\in S} \boldsymbol{x}^*_{\mathsf{LP}}[i] \right\rceil$ where S is a subset of the integer variables such that $\sum_{i\in S} \boldsymbol{x}^*_{\mathsf{LP}}[i] \notin \mathbb{Z}$. Here, $\texttt{actions}(\mathcal{T}, Q) = 2^{[n]}$, so, $\mathrm{Pdim}(\{\texttt{cost}_{\mu,\lambda}\}) = O(n^2)$. So our sample complexity bound for multivariable branching constraints is, surprisingly, only a constant factor worse than the bound for single-variable branching constraints.

We give a simple example where B&C using only single variable branches builds a tree of exponential size, while a single branch on the entire set of variables at the root yields two infeasible subproblems (and a B&C tree of size 3).

Theorem 3. *For any n, there is an IP with two constraints and n variables such that with only single variable branches, B&C builds a tree of size $2^{(n-1)/2}$, while with a suitable multivariable branch, B&C builds a tree of size three.*

Proof. Let n be an odd positive integer. Consider the infeasible IP $\max\{\sum_{i=1}^n x[i] : 2\sum_{i=1}^n x[i] = n, \boldsymbol{x} \in \{0,1\}^n\}$. Jeroslow [22] proved that with only single-variable branches, B&C builds a tree with $2^{(n-1)/2}$ nodes to determine infeasibility. However, with a suitable multivariable branch, B&C will build a tree of constant size. The optimal solution to the LP relaxation of the IP is attained when all variables are set to $1/2$. A multivariable branch on all n variables produces the two subproblems with constraints $\sum_{i=1}^n x[i] \leq \lfloor n/2 \rfloor$ and $\sum_{i=1}^n x[i] \geq \lceil n/2 \rceil$, respectively. Since n is odd, $\lfloor n/2 \rfloor < n/2$ and $\lceil n/2 \rceil > n/2$, so the LP relaxations of both subproblems are infeasible. Thus, B&C builds a tree with three nodes. □

Yang et al. [36] provide more examples of situations where multivariable branching yields dramatic improvements in tree size over single variable branching. They also perform a computational evaluation of a few different strategies for generating multivariable branching constraints. Yang et al. [35] explore gradient-boosting for learning to mimic strong branching for multiple variables.

Branching on general disjunctions Branching constraints can be even more general than multivariable branches. Given any integer vector $\boldsymbol{\pi} \in \mathbb{Z}^n$ and any integer $\pi_0 \in \mathbb{Z}$ (jointly referred to as a *disjunction*), the constraints $\boldsymbol{\pi}^T \boldsymbol{x} \leq \pi_0$ or $\boldsymbol{\pi}^T \boldsymbol{x} \geq \pi_0 + 1$ represent a valid partition of the feasible region into subproblems. Owen and Mehrotra [29] ran the first experiments demonstrating that branching on general disjunctions can lead to significantly smaller tree sizes. Subsequent works have posed different heuristics to select disjunctions to branch on [12, 28].

In practice it is known that additional IP constraints should not have coefficients that are too large. If C is a bound on the magnitude of the coefficient of any disjunction, then $\texttt{actions}(\mathcal{T}, Q) = \{-C, \dots, C\}^{n+1}$, so $\text{Pdim}(\{\texttt{cost}_{\mu,\lambda}\}) = O(n^2 \log C)$. Karamanov and Cornuéjols [24] conduct a computational evaluation of disjunctions derived from Gomory mixed-integer cuts. In this setting, $\texttt{actions}(\mathcal{T}, Q)$ is the set of m or fewer disjunctions corresponding to the m or fewer Gomory mixed-integer cuts derived from the simplex tableau from solving the LP relaxation of Q. In this case, $\text{Pdim}(\{\texttt{cost}_{\mu,\lambda}\}) = O(n^2 + n \log m)$.

4.2 Cutting planes

The action set can also correspond to cutting planes used to refine the feasible region of the IP at any stage of B&C. Here, $\texttt{actions}(\mathcal{T}, Q)$ is any set of cutting planes derived solely using the path from the root to the IP at Q. Examples include the set of Gomory mixed-integer cuts derived from the simplex tableau and (in the pure-integer case) the set of Chvátal-Gomory (CG) cuts derived from the simplex tableau [10, 15]. The set $\texttt{actions}(\mathcal{T}, Q)$ can also consist of sequences of cutting planes, representing adding several cutting planes to the IP in waves. For example, the set of all sequences of w cuts generated from the simplex tableau for an IP with m constraints has size at most m^w (regardless of whether the LP is resolved after each cut). The number of such cutting planes provided by the LP tableau at any node in the tree is at most $O(m + nw)$ (the original IP has m constraints, and after at most n branches there are an additional n branching constraints and at most nw cutting planes), which means that $|\texttt{actions}(\mathcal{T}, Q)| \leq O(m + nw)^w$. Thus, $\text{Pdim}(\{\texttt{cost}_{\mu,\lambda}\}) = O(n^2 + nw \log(m + nw))$.

We can also handle arbitrary CG cuts (not just ones from the LP tableau). Balcan et al. [8] proved that given an IP with feasible region $\{\boldsymbol{x} \in \mathbb{Z}^n : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$, even though there are infinitely many CG cut parameters, there are effectively only $O(w 2^w \|A\|_{1,1} + 2^w \|\boldsymbol{b}\|_1 + nw)^{1+mw}$ distinct sequences of cutting planes that w CG cut parameters can produce. At any node in the B&C tree, the number of constraints is at most $O(m + nw)$. So, on the domain of IPs with $\|A\|_{1,1} \leq \alpha$ and $\|\boldsymbol{b}\|_1 \leq \beta$, $|\texttt{actions}(\mathcal{T}, Q)| \leq O(w 2^w \alpha + 2^w \beta + nw)^{1 + w \cdot O(m+nw)}$. Thus, $\text{Pdim}(\{\texttt{cost}_{\mu,\lambda}\}) = O(n^2 w^3 m \log(\alpha + \beta + n))$.

4.3 Improved bounds for branch-and-cut

To allow node selection, branching, and cutting-plane selection to be tuned simultaneously, we apply Theorem 2. This allows us to bound the pseudo-dimension of the family of functions $\{\texttt{cost}_{\mu_1,\mu_2,\lambda}\}$, where μ_1 controls branching, μ_2 controls cutting-plane selection, and λ controls node selection. Let $\texttt{actions}_1(\mathcal{T}, Q)$ denote the set of branching actions available at Q, and let $\texttt{actions}_2(\mathcal{T}, Q)$ denote the set of cutting planes available at Q. Let $b_1, b_2 \in \mathbb{N}$ be such that $\texttt{actions}_1(\mathcal{T}, Q) \leq b_1$ and $\texttt{actions}_2(\mathcal{T}, Q) \leq b_2$ for all $\mathcal{T}$ and all $Q \in \mathcal{T}$. Fix two branching scores $\texttt{ascore}_1^1, \texttt{ascore}_2^1$, fix two cutting-plane selection scores $\texttt{ascore}_1^2, \texttt{ascore}_2^2$, and fix two node-selection scores $\texttt{nscore}_1, \texttt{nscore}_2$.

Theorem 4. *Let $\texttt{cost}(Q)$ be any tree-constant cost function, and let $\texttt{cost}_{\mu_1,\mu_2,\lambda}$ be the cost of the tree built by B&C using branching score $\mu_1 \cdot \texttt{ascore}_1^1 + (1 - \mu_1) \cdot \texttt{ascore}_2^1$, cutting-plane selection score $\mu_2 \cdot \texttt{ascore}_1^2 + (1 - \mu_2) \cdot \texttt{ascore}_2^2$, and node-selection score $\lambda \cdot \texttt{nscore}_1 + (1 - \lambda) \cdot \texttt{nscore}_2$. Then, with $\Delta = O(n)$, $\mathrm{Pdim}(\{\texttt{cost}_{\mu_1,\mu_2,\lambda}\}) = O(n^2 + n \log(b_1 + b_2))$.*

Comparison to existing bounds. Balcan et al. [8] give a pseudo-dimension bound for tree search with a linear dependence on a cap κ on the number of nodes allowed in any tree. Their pseudo-dimension bound in our setting is $\mathrm{Pdim}(\{\texttt{cost}_{\mu_1,\mu_2,\lambda}\}) = O(\kappa \log \kappa + \kappa \log b_1 + \kappa \log b_2)$. While κ is treated as a constant, it can be a prohibitively large quantity. In fact, without explicitly enforcing a limit on the number of nodes expanded by B&C, Balcan et al. [8] obtain a pseudo-dimension bound of $O(2^n(\log b_1 + \log b_2))$.

Balcan et al. [6] use the path-wise property to prove that $\mathrm{Pdim}(\{\texttt{cost}_\mu\}) = O(n^2)$ for single-variable branching, but for the case where branching is the only tunable component of B&C (and node selection is fixed).

5 Conclusions and future research

We presented a general model of tree search and proved sample complexity guarantees for this model that improve and generalize upon the recent sample complexity theory for configuring branch-and-cut. There are many interesting and open directions for future research. One compelling open question is to obtain pseudo-dimension bounds when action sets are infinite. Balcan et al. [8] alluded to this question in the case of cutting planes, and neither the techniques of their work nor the techniques of the present work can handle, for example, important infinite cutting-plane families such as the class of Gomory mixed-integer cuts, or the infinitely many valid disjunctions that could be branched on. Beyond integer programming, our model of tree search could potentially be applied to completely different problem domains that exhibit tree structure.

Bibliography

[1] Achterberg, T.: Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin (2007)

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

[3] Anthony, M., Bartlett, P.: Neural Network Learning: Theoretical Foundations. Cambridge University Press (2009)

[4] Balcan, M.F.: Data-driven algorithm design. In: Roughgarden, T. (ed.) Beyond Worst Case Analysis of Algorithms. Cambridge University Press (2020)

[5] Balcan, M.F., DeBlasio, D., Dick, T., Kingsford, C., Sandholm, T., Vitercik, E.: How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. In: Annual Symposium on Theory of Computing (STOC) (2021)

[6] Balcan, M.F., Dick, T., Sandholm, T., Vitercik, E.: Learning to branch. In: International Conference on Machine Learning (ICML) (2018)

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

[8] Balcan, M.F., Prasad, S., Sandholm, T., Vitercik, E.: Sample complexity of tree search configuration: Cutting planes and beyond. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2021)

[9] Chmiela, A., Khalil, E.B., Gleixner, A., Lodi, A., Pokutta, S.: Learning to schedule heuristics in branch-and-bound. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2021)

[10] Chvátal, V.: Edmonds polytopes and a hierarchy of combinatorial problems. Discrete mathematics **4**(4), 305–337 (1973)

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

[12] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**, 23–47 (2002)

[13] Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Annual Conference on Neural Information Processing Systems (NeurIPS). pp. 15554–15566 (2019)

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

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

[16] Gupta, P., Gasse, M., Khalil, E.B., Kumar, M.P., Lodi, A., Bengio, Y.: Hybrid models for learning to branch. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2020)

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

[18] He, H., Daume III, H., Eisner, J.M.: Learning to search in branch and bound algorithms. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2014)

[19] Huang, Z., Wang, K., Liu, F., Zhen, H.l., Zhang, W., Yuan, M., Hao, J., Yu, Y., Wang, J.: Learning to select cuts for efficient mixed-integer programming. arXiv preprint arXiv:2105.13645 (2021)

[20] Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization (LION). pp. 507–523 (2011)

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

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

[23] Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC—instance-specific algorithm configuration. In: European Conference on Artificial Intelligence (ECAI) (2010)

[24] Karamanov, M., Cornuéjols, G.: Branching on general disjunctions. Mathematical Programming **128**(1-2), 403–436 (2011)

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

[26] Khalil, E., Dilkina, B., Nemhauser, G., Ahmed, S., Shao, Y.: Learning to run heuristics in tree search. In: International Joint Conference on Artificial Intelligence (IJCAI) (2017)

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

[28] Mahajan, A., Ralphs, T.K.: Experiments with branching using general disjunctions. In: Operations Research and Cyber-Infrastructure, pp. 101–118. Springer (2009)

[29] Owen, J.H., Mehrotra, S.: Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. Computational Optimization and Applications **20**(2), 159–170 (November 2001)

[30] Sabharwal, A., Samulowitz, H., Reddy, C.: Guiding combinatorial optimization with UCT. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer (2012)

[31] Sandholm, T.: Very-large-scale generalized combinatorial multi-attribute auctions: Lessons from conducting $60 billion of sourcing. In: Neeman, Z.,

Roth, A., Vulkan, N. (eds.) Handbook of Market Design. Oxford University Press (2013)

[32] Song, J., Lanka, R., Yue, Y., Dilkina, B.: A general large neighborhood search framework for solving integer programs. In: Annual Conference on Neural Information Processing Systems (NeurIPS) (2020)

[33] Tang, Y., Agrawal, S., Faenza, Y.: Reinforcement learning for integer programming: Learning to cut. International Conference on Machine Learning (ICML) (2020)

[34] Weisz, G., György, A., Szepesvári, C.: LeapsAndBounds: A method for approximately optimal algorithm configuration. In: International Conference on Machine Learning (ICML) (2018)

[35] Yang, Y., Boland, N., Dilkina, B., Savelsbergh, M.: Learning generalized strong branching for set covering, set packing, and 0-1 knapsack problems. Tech. rep., Technical Report, 2020. (2020)

[36] Yang, Y., Boland, N., Savelsbergh, M.: Multivariable branching: A 0-1 knapsack problem case study. INFORMS Journal on Computing (2021)