Sample Complexity of Tree Search Configuration: Cutting Planes and Beyond

Anonymous Author(s)
Affiliation
Address
email

Abstract

Cutting-plane methods have enabled remarkable successes in integer programming over the last few decades. State-of-the-art solvers integrate a myriad of cutting-plane techniques to speed up the underlying tree-search algorithm used to find optimal solutions. In this paper we prove the first guarantees for learning high-performing cut-selection policies tailored to the instance distribution at hand using samples. We first bound the sample complexity of learning cutting planes from the canonical family of Chvátal-Gomory cuts. Our bounds handle any number of waves of any number of cuts and are fine tuned to the magnitudes of the constraint coefficients. Next, we prove sample complexity bounds for more sophisticated cut selection policies that use a combination of scoring rules to choose from a family of cuts. Finally, beyond the realm of cutting planes for integer programming, we develop a general abstraction of tree search that captures key components such as node selection and variable selection. For this abstraction, we bound the sample complexity of learning a good policy for building the search tree.

1 Introduction

Integer programming is one of the most broadly-applicable tools in computer science, used to formulate problems from operations research (such as routing, scheduling, and pricing), machine learning (such as adversarially-robust learning, MAP estimation, and clustering), and beyond. *Branch-and-cut (B&C)* is the most widely-used algorithm for solving integer programs (IPs). B&C is highly configurable, and with a deft configuration, it can be used to solve computationally challenging problems. Finding a good configuration, however, is a notoriously difficult problem.

We study machine learning approaches to configuring policies for selecting *cutting planes*, which have an enormous impact on B&C's performance. At a high level, B&C works by recursively partitioning the IP's feasible region, searching for the locally optimal solution within each set of the partition, until it can verify that it has found the globally optimal solution. An IP's feasible region is defined by a set of linear inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$ and integer constraints $\boldsymbol{x} \in \mathbb{Z}^n$, where n is the number of variables. By dropping the integrality constraints, we obtain the *linear programming (LP) relaxation* of the IP, which can be solved efficiently. A cutting plane is a carefully-chosen linear inequality $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta$ which refines the LP relaxation's feasible region without separating any integral point. Intuitively, a well-chosen cutting plane will remove a large portion of the LP relaxation's feasible region, speeding up the time it takes B&C to find the optimal solution to the original IP. Cutting plane selection is a crucial task, yet it is challenging because many cutting planes and cut-selection policies have tunable parameters, and the best configuration depends intimately on the application domain.

We provide the first provable guarantees for learning high-performing cutting planes and cut-selection policies, tailored to the application at hand. We model the application domain via an unknown, application-specific distribution over IPs, as is standard in the literature on using machine learning

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

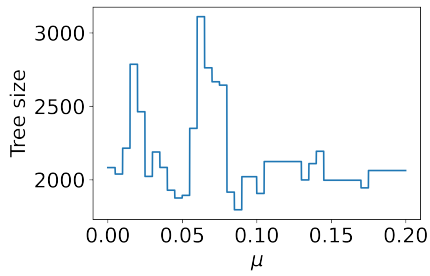


Figure 1: Tree size as a function of a SCIP cut-selection parameter μ (the directed cutoff distance weight, defined in Section 2) on an IP generated from the Combinatorial Auctions Test Suite [28] (the "regions" generator with 600 bids and 600 goods). SCIP [15] is the leading open-source IP solver.

for integer programming [e.g., 19, 21, 29, 34, 41]. For example, this could be a distribution over the routing IPs that a shipping company must solve day after day. The learning algorithm's input is a training set sampled from this distribution. The goal is to use this training set to learn cutting planes and cut-selection policies with strong future performance on problems from the same application but which are not already in the training set—or more formally, strong expected performance.

1.1 Summary of main contributions and overview of techniques

As our first main contribution, we bound the *sample complexity* of learning high-performing cutting planes. Fixing a family of cutting planes, these guarantees bound the number of samples sufficient to ensure that for any sequence of cutting planes from the family, its average performance over the samples is close to its expected performance. We measure performance in terms of the size of the search tree B&C builds. Our guarantees apply to the parameterized family of *Chvátal-Gomory (CG) cuts* [10, 16], one of the most widely-used families of cutting planes.

The overriding challenge is that to provide guarantees, we must analyze how the tree size changes as a function of the cut parameters. This is a sensitive function—slightly shifting the parameters can cause the tree size to shift from constant to exponential in the number of variables. Our key technical insight is that as the parameters vary, the entries of the cut (i.e., the vector $\boldsymbol{\alpha}$ and offset β of the cut $\boldsymbol{\alpha}^T \boldsymbol{x} \leq \beta$) are multivariate polynomials of bounded degree. The number of terms defining the polynomials is exponential in the number of parameters, but we show that the polynomials can be embedded in a space with dimension sublinear in the number of parameters. This insight allows us to better understand tree size as a function of the parameters. We then leverage results by Balcan et al. [8] that show how to use structure exhibited by dual functions (measuring an algorithm's performance, such as its tree size, as a function of its parameters) to derive sample complexity bounds.

Our second main contribution is a sample complexity bound for learning cut-selection policies, which allow B&C to adaptively select cuts as it solves the input IP. These cut-selection policies assign a number of real-valued scores to a set of cutting planes and then apply the cut that has the maximum weighted sum of scores. Tree size is a volatile function of these weights, though we prove that it is piecewise constant, as illustrated in Figure 1, which allows us to prove our sample complexity bound.

Finally, as our third main contribution, we provide guarantees for tuning weighted combinations of scoring rules for other aspects of tree search beyond cut selection, including node and variable selection. We prove that there is a set of hyperplanes splitting the parameter space into regions such that if tree search uses any configuration from a single region, it will take the same sequence of actions. This structure allows us to prove our sample complexity bound. This is the first paper to provide guarantees for tree search configuration that apply simultaneously to multiple different aspects of the algorithm—prior research was specific to variable selection [5].

1.2 Related work

Applied research on tree search configuration. Over the past decade, a substantial literature has developed on the use of machine learning for integer programming and tree search [e.g., 2, 7, 9,

12, 18, 21–23, 27, 29–31, 33, 34, 39–41]. This has included research that improves specific aspects of B&C such as variable selection [2, 12, 22, 27, 30, 39], node selection [18, 33], and heuristic scheduling [23]. These papers are applied, whereas we focus on providing theoretical guarantees.

With respect to cutting plane selection, the focus of this paper, Sandholm [34] uses machine learning techniques to customize B&C for combinatorial auction winner determination, including cutting plane selection. Tang et al. [35] study machine learning approaches to cutting plane selection. They formulate this problem as a reinforcement learning problem and show that their approach can outperform human-designed heuristics for a variety of tasks. Meanwhile, the focus of our paper is to provide the first provable guarantees for cutting plane selection via machine learning.

Ferber et al. [14] study a problem where the IP objective vector $\boldsymbol{c}$ is unknown, but an estimate $\hat{\boldsymbol{c}}$ can be obtained from data. Their goal is to optimize the quality of the solutions obtained by solving the IP defined by $\hat{\boldsymbol{c}}$, with respect to the true vector $\boldsymbol{c}$. They do so by formulating the IP as a differentiable layer in a neural network. The nonconvex nature of the IP does not allow for straightforward gradient computation for the backward pass, so they obtain a continuous surrogate using cutting planes.

Provable guarantees for algorithm configuration. Gupta and Roughgarden [17] initiated the study of sample complexity bounds for algorithm configuration. In research most related to ours, Balcan et al. [5] provide sample complexity bounds for learning tree search *variable selection policies (VSPs)*. They prove their bounds by showing that for any IP, hyperplanes partition the VSP parameter space into regions where the B&C tree size is a constant function of the parameters. The analysis in this paper requires new techniques because although we prove that the B&C tree size is a piecewise-constant function of the CG cutting plane parameters, the boundaries between pieces are far more complex than hyperplanes: they are hypersurfaces defined by multivariate polynomials.

Kleinberg et al. [24, 25] and Weisz et al. [36, 37] design configuration procedures for runtime minimization that come with theoretical guarantees. Their algorithms are designed for the case where there are a finitely-many parameter settings to choose from (although they are still able to provide guarantees for infinite parameter spaces by running their procedure on a finite sample of configurations; Balcan et al. [5, 6] analyze when discretization approaches can and cannot be gainfully employed). In contrast, our guarantees are designed for infinite parameter spaces.

2 Problem formulation

In this section we give a more detailed technical overview of branch-and-cut, as well as an overview of the tools from learning theory we use to prove sample complexity guarantees.

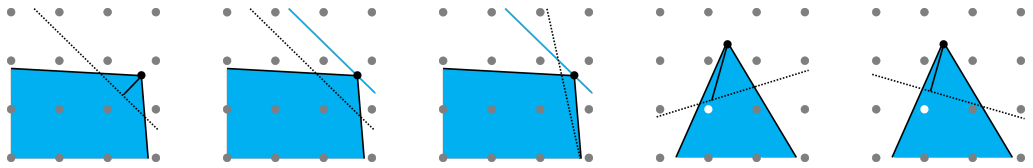
2.1 Branch-and-cut

We study integer programs (IPs) in canonical form given by

$$\max \left\{ \boldsymbol{c}^T \boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0, \boldsymbol{x} \in \mathbb{Z}^n \right\}, \tag{1}$$

where $A \in \mathbb{Z}^{m \times n}$, $\boldsymbol{b} \in \mathbb{Z}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$. Branch-and-cut (B&C) works by recursively partitioning the input IP's feasible region, searching for the locally optimal solution within each set of the partition until it can verify that it has found the globally optimal solution. It organizes this partition as a search tree, with the input IP stored at the root. It begins by solving the LP relaxation of the input IP; we denote the solution as $\boldsymbol{x}^*_{\mathsf{LP}} \in \mathbb{R}^n$. If $\boldsymbol{x}^*_{\mathsf{LP}}$ satisfies the IP's integrality constraints ($\boldsymbol{x}^*_{\mathsf{LP}} \in \mathbb{Z}^n$), then the procedure terminates—$\boldsymbol{x}^*_{\mathsf{LP}}$ is the globally optimal solution. Otherwise, it uses a *variable selection policy* to choose a variable $x[i]$. In the left child of the root, it stores the original IP with the additional constraint that $x[i] \leq \lfloor x^*_{\mathsf{LP}}[i] \rfloor$, and in the right child, with the additional constraint that $x[i] \geq \lceil x^*_{\mathsf{LP}}[i] \rceil$. It then uses a *node selection policy* to select a leaf of the tree and repeats this procedure—solving the LP relaxation and branching on a variable. B&C can *fathom* a node, meaning that it will stop searching along that branch, if 1) the LP relaxation satisfies the IP's integrality constraints, 2) the LP relaxation is infeasible, or 3) the objective value of the LP relaxation's solution is no better than the best integral solution found thus far. We assume there is a bound κ on the size of the tree we allow B&C to build before we terminate, as is common in prior research [5, 19, 24, 25].

Cutting planes are a means of ensuring that at each iteration of B&C, the solution to the LP relaxation is as close to the optimal integral solution as possible. Formally, let $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$



(a) Efficacy (b) Better objective parallelism (c) Worse objective parallelism (d) Better directed cutoff distance (e) Worse directed cutoff distance

Figure 2: Illustration of scoring rules. In each figure, the blue region is the feasible region, the black dotted line is the cut in question, the blue solid line is orthogonal to the objective $\boldsymbol{c}$, the black dot is the LP optimal solution, and the white dot is the incumbent IP solution. Figure 2a illustrates efficacy, which is the length of the black solid line between the cut and the LP optimal solution. The cut in Figure 2b has better objective parallelism than the cut in Figure 2c. The cut in Figure 2d has a better directed cutoff distance than the cut in Figure 2e, but both have the same efficacy.

denote the feasible region obtained by taking the LP relaxation of IP (1). Let $\mathcal{P}_I = \text{conv}(\mathcal{P} \cap \mathbb{Z}^n)$ denote the integer hull of $\mathcal{P}$. A *valid* cutting plane is any hyperplane $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \boldsymbol{\beta}$ such that if $\boldsymbol{x}$ is in the integer hull ($\boldsymbol{x} \in \mathcal{P}_I$), then $\boldsymbol{x}$ satisfies the inequality $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \boldsymbol{\beta}$. In other words, a valid cut does not remove any integral point from the LP relaxation's feasible region. A valid cutting plane *separates* $\boldsymbol{x} \in \mathcal{P} \setminus \mathcal{P}_I$ if it does not satisfy the inequality, or in other words, $\boldsymbol{\alpha}^T\boldsymbol{x} > \beta$. At any node of the search tree, B&C can add valid cutting planes that separate the optimal solution to the node's LP relaxation, thus improving the solution estimates used to prune the search tree. However, adding too many cuts will increase the time it takes to solve the LP relaxation at each node. Therefore, solvers such as SCIP [15], the leading open-source solver, bound the number of cuts that will be applied.

A famous class of cutting planes is the family of *Chvátal-Gomory (CG) cuts*[1] [10, 16], which are parameterized by vectors $\boldsymbol{u} \in \mathbb{R}^m$. The CG cut defined by $\boldsymbol{u} \in \mathbb{R}^m$ is the hyperplane $\lfloor \boldsymbol{u}^T A \rfloor \boldsymbol{x} \leq \lfloor \boldsymbol{u}^T \boldsymbol{b} \rfloor$, which is guaranteed to be valid. Throughout this paper we primarily restrict our attention to $\boldsymbol{u} \in [0,1)^m$. This is without loss of generality, since the facets of $\mathcal{P} \cap \{\boldsymbol{x} \in \mathbb{R}^n : \lfloor \boldsymbol{u}^T A \rfloor \boldsymbol{x} \leq \lfloor \boldsymbol{u}^T \boldsymbol{b} \rfloor \, \forall \boldsymbol{u} \in \mathbb{R}^m\}$ can be described by the finitely many $\boldsymbol{u} \in [0,1)^m$ such that $\boldsymbol{u}^T A \in \mathbb{Z}^n$ [10].

Some IP solvers such as SCIP use *scoring rules* to select among cutting planes, which are meant to measure the quality of a cut. Some commonly-used scoring rules include *efficacy* [4] ($\texttt{score}_1$), *objective parallelism* [1] ($\texttt{score}_2$), *directed cutoff distance* [15] ($\texttt{score}_3$), and *integral support* [38] ($\texttt{score}_4$) (defined in Appendix A). *Efficacy* measures the distance between the cut $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta$ and $\boldsymbol{x}^*_{\mathsf{LP}}$: $\texttt{score}_1(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = (\boldsymbol{\alpha}^T\boldsymbol{x}^*_{\mathsf{LP}} - \beta) / \|\boldsymbol{\alpha}\|_2$, as illustrated in Figure 2a. *Objective parallelism* measures the angle between the objective $\boldsymbol{c}$ and the cut's normal vector $\boldsymbol{\alpha}$: $\texttt{score}_2(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = \left|\boldsymbol{c}^T\boldsymbol{\alpha}\right| / (\|\boldsymbol{\alpha}\|_2 \|\boldsymbol{c}\|_2)$, as illustrated in Figures 2b and 2c. *Directed cutoff distance* measures the distance between the LP optimal solution and the cut in a more relevant direction than the efficacy scoring rule. Specifically, let $\overline{\boldsymbol{x}}$ be the *incumbent solution*, which is the best-known feasible solution to the input IP. The directed cutoff distance is the distance between the hyperplane $(\boldsymbol{\alpha}, \beta)$ and the current LP solution $\boldsymbol{x}^*_{\mathsf{LP}}$ along the direction of the incumbent $\overline{\boldsymbol{x}}$, as illustrated in Figures 2d and 2e: $\texttt{score}_3(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = \|\overline{\boldsymbol{x}} - \boldsymbol{x}^*_{\mathsf{LP}}\|_2 \cdot (\boldsymbol{\alpha}^T\boldsymbol{x}^*_{\mathsf{LP}} - \beta) / \left|\boldsymbol{\alpha}^T(\overline{\boldsymbol{x}} - \boldsymbol{x}^*_{\mathsf{LP}})\right|$. SCIP uses the scoring rule $\frac{3}{5}\texttt{score}_1 + \frac{1}{10}\texttt{score}_2 + \frac{1}{2}\texttt{score}_3 + \frac{1}{10}\texttt{score}_4$ [15].

2.2 Learning theory background

The goal of this paper is to learn cut-selection policies using samples in order to guarantee, with high probability, that B&C builds a small tree in expectation on unseen IPs. To this end, we rely on the notion of *pseudo-dimension* [32], a well-known measure of a function class's *intrinsic complexity*. The pseudo-dimension of a function class $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{Y}}$, denoted $\text{Pdim}(\mathcal{F})$, is the largest integer N for which there exist N inputs $y_1, \ldots, y_N \in \mathcal{Y}$ and N thresholds $r_1, \ldots, r_N \in \mathbb{R}$ such that for every $(\sigma_1, \ldots, \sigma_N) \in \{0,1\}^N$, there exists $f \in \mathcal{F}$ such that $f(y_i) \geq r_i$ if and only if $\sigma_i = 1$. Function classes with bounded pseudo-dimension satisfy the following uniform convergence guarantee [3, 32]. Let $[-\kappa, \kappa]$ be the range of the functions in $\mathcal{F}$, let $N_{\mathcal{F}}(\varepsilon, \delta) = O(\frac{\kappa^2}{\varepsilon^2}\text{Pdim}(\mathcal{F})\ln(\frac{1}{\delta}))$, and let $N \geq N_{\mathcal{F}}(\varepsilon, \delta)$. For all distributions $\mathcal{D}$ on $\mathcal{Y}$, with probability $1-\delta$ over the draw of $y_1, \ldots, y_N \sim \mathcal{D}$,

[1] The set of CG cuts is equivalent to the set of Gomory (fractional) cuts [11], another commonly studied family of cutting planes with a slightly different parameterization.

for every function $f \in \mathcal{F}$, the average value of f over the samples is within ε of its expected value: $|\frac{1}{N}\sum_{i=1}^{N} f(y_i) - \mathbb{E}_{y\sim\mathcal{D}}[f(y)]| \leq \varepsilon$. The quantity $N_{\mathcal{F}}(\varepsilon, \delta)$ is the *sample complexity* of $\mathcal{F}$.

3 Learning Chvátal-Gomory cuts

In this section we bound the sample complexity of learning CG cuts at the root node of the B&C search tree. We warm up by analyzing the case where a single CG cut is added at the root (Section 3.1), and then build on this analysis to handle W sequential *waves* of k simultaneous CG cuts (Section 3.3). This means that all k cuts in the first wave are added simultaneously, the new (larger) LP relaxation is solved, all k cuts in the second wave are added to the new problem simultaneously, and so on. B&C adds cuts in waves because otherwise, the angles between cuts would become obtuse, leading to numerical instability. Moreover, many commercial IP solvers only add cuts at the root because those cuts can be leveraged throughout the tree. However, in Section 5, we also provide guarantees for applying cuts throughout the tree. In this section, we assume that all aspects of B&C (such as node selection and variable selection) are fixed except for the cuts applied at the root of the search tree.

3.1 Learning a single cut

To provide sample complexity bounds, as per Section 2.2, we bound the pseudo-dimension of the set of functions $f_{\boldsymbol{u}}$ for $\boldsymbol{u} \in [0,1]^m$, where $f_{\boldsymbol{u}}(\boldsymbol{c}, A, \boldsymbol{b})$ is the size of the tree B&C builds when it applies the CG cut defined by $\boldsymbol{u}$ at the root. To do so, we take advantage of structure exhibited by the class of *dual* functions, each of which is defined by a fixed IP $(\boldsymbol{c}, A, \boldsymbol{b})$ and measures tree size as a function of the parameters $\boldsymbol{u}$. In other words, each dual function $f^*_{\boldsymbol{c},A,\boldsymbol{b}} : [0,1]^m \to \mathbb{R}$ is defined as $f^*_{\boldsymbol{c},A,\boldsymbol{b}}(\boldsymbol{u}) = f_{\boldsymbol{u}}(\boldsymbol{c}, A, \boldsymbol{b})$. Our main result in this section is a proof that the dual functions are well-structured (Lemma 3.2), which then allows us to apply a result by Balcan et al. [8] to bound $\text{Pdim}(\{f_{\boldsymbol{u}} : \boldsymbol{u} \in [0,1]^m\})$ (Theorem 3.3). Proving that the dual functions are well-structured is challenging because they are volatile: slightly perturbing $\boldsymbol{u}$ can cause the tree size to shift from constant to exponential in n, as we prove in the following theorem. The full proof is in Appendix B.

Theorem 3.1. *For any integer n, there exists an integer program $(\boldsymbol{c}, A, \boldsymbol{b})$ with two constraints and n variables such that if $\frac{1}{2} \leq u[1] - u[2] < \frac{n+1}{2n}$, then applying the CG cut defined by $\boldsymbol{u}$ at the root causes B&C to terminate immediately. Meanwhile, if $\frac{n+1}{2n} \leq u[1] - u[2] < 1$, then applying the CG cut defined by $\boldsymbol{u}$ at the root causes B&C to build a tree of size at least $2^{(n-1)/2}$.*

Proof sketch. Without loss of generality, assume that n is odd. Consider an IP with constraints $2(x[1] + \cdots + x[n]) \leq n$, $-2(x[1] + \cdots + x[n]) \leq -n$, $\boldsymbol{x} \in \{0,1\}^n$, and any objective. This IP is infeasible because n is odd. Jeroslow [20] proved that without the use of cutting planes or heuristics, B&C will build a tree of size $2^{(n-1)/2}$ before it terminates. We prove that when $\frac{1}{2} \leq u[1] - u[2] < \frac{n+1}{2n}$, the CG cut halfspace defined by $\boldsymbol{u} = (u[1], u[2])$ has an empty intersection with the feasible region of the IP, causing B&C to terminate immediately. On the other hand, we show that if $\frac{n+1}{2n} \leq u[1] - u[2] < 1$, then the CG cut halfspace defined by $\boldsymbol{u}$ contains the feasible region of the IP, and thus leaves the feasible region unchanged. In this case, due to Jeroslow [20], applying this CG cut at the root will cause B&C to build a tree of size at least $2^{(n-1)/2}$ before it terminates. □

This theorem shows that the dual tree-size functions can be extremely sensitive to perturbations in the CG cut parameters. However, we are able to prove that the dual functions are piecewise-constant.

Lemma 3.2. *For any IP $(\boldsymbol{c}, A, \boldsymbol{b})$, there are $O(\|A\|_{1,1} + \|\boldsymbol{b}\|_{1,1} + n)$ hyperplanes that partition $[0,1]^m$ into regions where in any one region R, the dual function $f^*_{\boldsymbol{c},A,\boldsymbol{b}}(\boldsymbol{u})$ is constant for all $\boldsymbol{u} \in R$.*

Proof. Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n \in \mathbb{R}^m$ be the columns of A. Let $A_i = \|\boldsymbol{a}_i\|_1$ and $B = \|\boldsymbol{b}\|_1$, so for any $\boldsymbol{u} \in [0,1]^m$, $\lfloor \boldsymbol{u}^T \boldsymbol{a}_i \rfloor \in [-A_i, A_i]$ and $\lfloor \boldsymbol{u}^T \boldsymbol{b} \rfloor \in [-B, B]$. For each integer $k_i \in [-A_i, A_i]$, we have $\lfloor \boldsymbol{u}^T \boldsymbol{a}_i \rfloor = k_i \iff k_i \leq \boldsymbol{u}^T \boldsymbol{a}_i < k_i + 1$. There are $\sum_{i=1}^{n} 2A_i + 1 = O(\|A\|_{1,1} + n)$ such halfspaces, plus an additional $2B + 1$ halfspaces of the form $k_{n+1} \leq \boldsymbol{u}^T \boldsymbol{b} < k_{n+1} + 1$ for each $k_{n+1} \in \{-B, \ldots, B\}$. In any region R defined by the intersection of these halfspaces, the vector $(\lfloor \boldsymbol{u}^T \boldsymbol{a}_1 \rfloor, \ldots, \lfloor \boldsymbol{u}^T \boldsymbol{a}_n \rfloor, \lfloor \boldsymbol{u}^T \boldsymbol{b} \rfloor)$ is constant for all $\boldsymbol{u} \in R$, and thus so is the resulting cut. □

Combined with the main result of Balcan et al. [8], this lemma implies the following bound.

Theorem 3.3. *Let $\mathcal{F}_{\alpha,\beta}$ denote the set of all functions $f_{\boldsymbol{u}}$ for $\boldsymbol{u} \in [0,1]^m$ defined on the domain of IPs $(\boldsymbol{c}, A, \boldsymbol{b})$ with $\|A\|_{1,1} \leq \alpha$ and $\|\boldsymbol{b}\|_1 \leq \beta$. Then,* $\text{Pdim}(\mathcal{F}_{\alpha,\beta}) = O(m \log(m(\alpha+\beta+n)))$.

This theorem implies that $\widetilde{O}(\kappa^2 m/\varepsilon^2)$ samples are sufficient to ensure that with high probability, for every CG cut, the average size of the tree B&C builds upon applying the cutting plane is within ϵ of the expected size of the tree it builds (the $\widetilde{O}$ notation suppresses logarithmic terms).

3.2 Learning a sequence of cuts

We now determine the sample complexity of making W sequential CG cuts at the root. The first cut is defined by m parameters $\boldsymbol{u}_1 \in [0,1]^m$ for each of the m constraints. Its application leads to the addition of the row $\lfloor \boldsymbol{u}_1^T A \rfloor \boldsymbol{x} \leq \lfloor \boldsymbol{u}_1^T \boldsymbol{b} \rfloor$ to the constraint matrix. The next cut is then be defined by $m+1$ parameters $\boldsymbol{u}_2 \in [0,1]^{m+1}$ since there are now $m+1$ constraints. Continuing in this fashion, the Wth cut is be defined by $m+W-1$ parameters $\boldsymbol{u}_W \in [0,1]^{m+W-1}$. Let $f_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_W}(\boldsymbol{c}, A, \boldsymbol{b})$ be the size of the tree B&C builds when it applies the CG cut defined by $\boldsymbol{u}_1$, then applies the CG cut defined by $\boldsymbol{u}_2$ to the new IP, and so on, all at the root of the search tree.

As in Section 3.1, we bound the pseudo-dimension of the functions $f_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_W}$ by analyzing the structure of the dual functions $f^*_{\boldsymbol{c},A,\boldsymbol{b}}$, which measure tree size as a function of the parameters $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_W$. Specifically, $f^*_{\boldsymbol{c},A,\boldsymbol{b}} : [0,1]^m \times \cdots \times [0,1]^{m+W-1} \to \mathbb{R}$, where $f^*_{\boldsymbol{c},A,\boldsymbol{b}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_W) = f_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_W}(\boldsymbol{c}, A, \boldsymbol{b})$. The analysis in this section is more complex because the wth cut (with $w \in \{2, \ldots, W\}$) depends not only on the parameters $\boldsymbol{u}_w$ but also on $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{w-1}$. We prove that the dual functions are again piecewise-constant, but in this case, the boundaries between pieces are defined by multivariate polynomials rather than hyperplanes. The full proof is in Appendix B.

Lemma 3.4. *For any IP $(\boldsymbol{c}, A, \boldsymbol{b})$, there are $O(W 2^W \|A\|_{1,1} + 2^W \|\boldsymbol{b}\|_1 + nW)$ multivariate polynomials in $\leq W^2 + mW$ variables of degree $\leq W$ that partition $[0,1]^m \times \cdots \times [0,1]^{m+W-1}$ into regions where in any one region R, $f^*_{\boldsymbol{c},A,\boldsymbol{b}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_W)$ is constant for all $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_W) \in R$.*

Proof sketch. Let $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n \in \mathbb{R}^m$ be the columns of A. For $\boldsymbol{u}_1 \in [0,1]^m, \ldots, \boldsymbol{u}_W \in [0,1]^{m+W-1}$, define $\widetilde{\boldsymbol{a}}_i^1 \in [0,1]^m, \ldots, \widetilde{\boldsymbol{a}}_i^W \in [0,1]^{m+W-1}$ for each $i \in [n]$ such that $\widetilde{\boldsymbol{a}}_i^w$ is the ith column of the constraint matrix after applying cuts $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{w-1}$. Similarly, define $\widetilde{\boldsymbol{b}}^w$ to be the constraint vector after applying the first $w-1$ cuts. In other words, we have the recurrence relation

$$\widetilde{\boldsymbol{a}}_i^1 = \boldsymbol{a}_i \qquad \widetilde{\boldsymbol{b}}^1 = \boldsymbol{b}$$

$$\widetilde{\boldsymbol{a}}_i^w = \begin{bmatrix} \widetilde{\boldsymbol{a}}_i^{w-1} \\ \boldsymbol{u}_{w-1}^T \widetilde{\boldsymbol{a}}_i^{w-1} \end{bmatrix} \qquad \widetilde{\boldsymbol{b}}^w = \begin{bmatrix} \widetilde{\boldsymbol{b}}^{w-1} \\ \boldsymbol{u}_{w-1}^T \widetilde{\boldsymbol{b}}^{w-1} \end{bmatrix}$$

for $w = 2, \ldots, W$. We prove that $\lfloor \boldsymbol{u}_w^T \widetilde{\boldsymbol{a}}_i^w \rfloor \in [-2^{w-1} \|\boldsymbol{a}_i\|_1, 2^{w-1} \|\boldsymbol{a}_i\|_1]$. For each integer k_i in this interval, $\lfloor \boldsymbol{u}_w^T \widetilde{\boldsymbol{a}}_i^w \rfloor = k_i \iff k_i \leq \boldsymbol{u}_w^T \widetilde{\boldsymbol{a}}_i^w < k_i + 1$. The boundaries of these surfaces are defined by polynomials over $\boldsymbol{u}_w$ in $\leq mw + w^2$ variables with degree $\leq w$. Counting the total number of such hypersurfaces yields the lemma statement. □

We now use this structure to provide a pseudo-dimension bound. The full proof is in Appendix B.

Theorem 3.5. *Let $\mathcal{F}_{\alpha,\beta}$ denote the set of all functions $f_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_W}$ for $\boldsymbol{u}_1 \in [0,1]^m, \ldots, \boldsymbol{u}_W \in [0,1]^{m+W-1}$ defined on the domain of integer programs $(\boldsymbol{c}, A, \boldsymbol{b})$ with $\|A\|_{1,1} \leq \alpha$ and $\|\boldsymbol{b}\|_1 \leq \beta$. Then,* $\text{Pdim}(\mathcal{F}_{\alpha,\beta}) = O(mW^2 \log(mW(\alpha+\beta+n)))$.

Proof sketch. The space of $0/1$ classifiers induced by the set of degree $\leq W$ multivariate polynomials in $W^2 + mW$ variables has VC dimension $O((W^2 + mW) \log W)$ [3]. However, we more carefully examine the structure of the polynomials considered in Lemma 3.4 to give an improved VC dimension bound of $1 + mW$. For each $j = 1, \ldots, m$ define $\widetilde{\boldsymbol{u}}_1[j], \ldots, \widetilde{\boldsymbol{u}}_W[j]$ recursively as

$$\widetilde{\boldsymbol{u}}_1[j] = \boldsymbol{u}_1[j]$$

$$\widetilde{\boldsymbol{u}}_w[j] = \boldsymbol{u}_w[j] + \sum_{\ell=1}^{w-1} \boldsymbol{u}_w[m+\ell] \widetilde{\boldsymbol{u}}_\ell[j] \qquad \text{for } w = 2, \ldots, W$$

The space of polynomials induced by the wth cut is contained in $\text{span}\{1, \widetilde{\boldsymbol{u}}_w[1], \ldots, \widetilde{\boldsymbol{u}}_w[m]\}$. The intuition for this is as follows: consider the additional term added by the wth cut to the constraint matrix, that is, $\boldsymbol{u}_w^T \widetilde{\boldsymbol{a}}_i^w$. The first m coordinates $(\boldsymbol{u}_w[1], \ldots, \boldsymbol{u}_w[m])$ interact only with $\boldsymbol{a}_i$—so $\boldsymbol{u}_w[j]$ collects a coefficient of $\boldsymbol{a}_i[j]$. Each subsequent coordinate $\boldsymbol{u}_w[m+\ell]$ interacts with all coordinates of $\widetilde{\boldsymbol{a}}_i^w$ arising from the first ℓ cuts. The term that collects a coefficient of $\boldsymbol{a}_i[j]$ is precisely $\boldsymbol{u}_w[m+\ell]$ times the sum of all terms from the first ℓ cuts with a coefficient of $\boldsymbol{a}_i[j]$. Using standard facts about the VC dimension of vector spaces and their duals in conjunction with Lemma 3.4 and the framework of Balcan et al. [8] yields the theorem statement. □

The sample complexity (defined in Section 2.2) of learning W sequential cuts is thus $\widetilde{O}(\kappa^2 m W^2/\epsilon^2)$.

3.3 Learning waves of simultaneous cuts

We now determine the sample complexity of making W sequential waves of cuts at the root, each wave consisting of k simultaneous CG cuts. Given vectors $\boldsymbol{u}_1^1, \ldots, \boldsymbol{u}_1^k \in [0,1]^m, \boldsymbol{u}_2^1, \ldots, \boldsymbol{u}_2^k \in [0,1]^{m+k}, \ldots, \boldsymbol{u}_W^1, \ldots, \boldsymbol{u}_W^k \in [0,1]^{m+k(W-1)}$, let $f_{\boldsymbol{u}_1^1,\ldots,\boldsymbol{u}_1^k,\ldots,\boldsymbol{u}_W^1,\ldots,\boldsymbol{u}_W^k}(\boldsymbol{c}, A, \boldsymbol{b})$ be the size of the tree B&C builds when it applies the CG cuts defined by $\boldsymbol{u}_1^1, \ldots, \boldsymbol{u}_1^k$, then applies the CG cuts defined by $\boldsymbol{u}_2^1, \ldots, \boldsymbol{u}_2^k$ to the new IP, and so on, all at the root of the search tree. The full proof of the following theorem is in Appendix B, and follows from the observation that W waves of k simultaneous cuts can be viewed as making kW sequential cuts with the restriction that cuts within each wave assign nonzero weight only to constraints from previous waves.

Theorem 3.6. *Let $\mathcal{F}_{\alpha,\beta}$ be the set of all functions $f_{\boldsymbol{u}_1^1,\ldots,\boldsymbol{u}_1^k,\ldots,\boldsymbol{u}_W^1,\ldots,\boldsymbol{u}_W^k}$ for $\boldsymbol{u}_1^1, \ldots, \boldsymbol{u}_1^k \in [0,1]^m, \ldots, \boldsymbol{u}_W^1, \ldots, \boldsymbol{u}_W^k \in [0,1]^{m+k(W-1)}$ defined on the domain of integer programs $(\boldsymbol{c}, A, \boldsymbol{b})$ with $\|A\|_{1,1} \leq \alpha$ and $\|\boldsymbol{b}\|_1 \leq \beta$. Then,* $\text{Pdim}(\mathcal{F}_{\alpha,\beta}) = O(mk^2W^2 \log(mkW(\alpha+\beta+n)))$.

This result implies that the sample complexity of learning W waves of k cuts is $\widetilde{O}(\kappa^2 m k^2 W^2/\epsilon^2)$.

3.4 Data-dependent guarantees

So far, our guarantees have depended on the maximum possible norms of the constraint matrix and vector in the domain of IPs under consideration. The uniform convergence result in Section 2.2 for $\mathcal{F}_{\alpha,\beta}$ only holds for distributions over A and $\boldsymbol{b}$ with norms bounded by α and β, respectively. In Appendix B.1, we show how to convert these into more broadly applicable data-dependent guarantees that leverage properties of the distribution over IPs. These guarantees hold without assumptions on the distribution's support, and depend on $\mathbb{E}[\max_i \|A_i\|_{1,1}]$ and $\mathbb{E}[\max_i \|\boldsymbol{b}_i\|_1]$ (where the expectation is over N samples), thus giving a sharper sample complexity guarantee that is tuned to the distribution.

4 Learning cut selection policies

In Section 3, we studied the sample complexity of learning waves of specific cut parameters. In this section, we bound the sample complexity of learning *cut-selection policies* at the root, that is, functions that take as input an IP and output a candidate cut. Using scoring rules is a more nuanced way of choosing cuts since it allows for the cut parameters to depend on the input IP.

Formally, let $\mathcal{I}_m$ be the set of IPs with m constraints (the number of variables is always fixed at n) and let $\mathcal{H}_m$ be the set of all hyperplanes in $\mathbb{R}^m$. A *scoring rule* is a function $\texttt{score} : \cup_m(\mathcal{H}_m \times \mathcal{I}_m) \to \mathbb{R}_{\geq 0}$. The real value $\texttt{score}(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta, (\boldsymbol{c}, A, \boldsymbol{b}))$ is a measure of the quality of the cutting plane $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta$ for the IP $(\boldsymbol{c}, A, \boldsymbol{b})$. Examples include the scoring rules discussed in Section 2.1.

Suppose $\texttt{score}_1, \ldots, \texttt{score}_d$ are d different scoring rules. We now bound the sample complexity of learning a combination of these scoring rules that guarantee a low expected tree size.

Theorem 4.1. *Let $\mathcal{C}$ be a set of cutting-plane parameters such that for every IP $(\boldsymbol{c}, A, \boldsymbol{b})$, there is a decomposition of $\mathcal{C}$ into $\leq r$ regions such that the cuts generated by any two vectors in the same region are the same. Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d scoring rules. For $\boldsymbol{\mu} \in \mathbb{R}^d$, let $f_{\boldsymbol{\mu}}(\boldsymbol{c}, A, \boldsymbol{b})$ be the size of the tree B&C builds when it chooses a cut from $\mathcal{C}$ to maximize $\mu[1]\texttt{score}_1(\cdot, (\boldsymbol{c}, A, \boldsymbol{b})) + \cdots + \mu[d]\texttt{score}_d(\cdot, (\boldsymbol{c}, A, \boldsymbol{b}))$. Then,* $\text{Pdim}(\{f_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathbb{R}^d\}) = O(d \log(rd))$.

Proof. Fix an integer program $(\boldsymbol{c}, A, \boldsymbol{b})$. Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r \in \mathcal{C}$ be representative cut parameters for each of the r regions. Consider the hyperplanes $\sum_{i=1}^{d} \mu[i]\texttt{score}_i(\boldsymbol{u}_s) = \sum_{i=1}^{d} \mu[i]\texttt{score}_i(\boldsymbol{u}_t)$ for each $s \neq t \in \{1, \ldots, r\}$ (suppressing the dependence on $\boldsymbol{c}, A, \boldsymbol{b}$). These $O(r^2)$ hyperplanes partition $\mathbb{R}^d$ into regions such that as $\boldsymbol{\mu}$ varies in a given region, the cut chosen from $\mathcal{C}$ is invariant. The desired pseudo-dimension bound follows from the main result of Balcan et al. [8]. □

Theorem 4.1 can be directly instantiated with the class of CG cuts. Combining Lemma 3.2 with the basic combinatorial fact that k hyperplanes partition $\mathbb{R}^m$ into at most k^m regions, we get that the pseudo-dimension of $\{f_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathbb{R}^d\}$ defined on IPs with $\|A\|_{1,1} \leq \alpha$ and $\|\boldsymbol{b}\|_1 \leq \beta$ is $O(dm \log(d(\alpha + \beta + n)))$. Instantiating Theorem 4.1 with the set of all sequences of W CG cuts requires the following extension of scoring rules to sequences of cutting planes. A *sequential scoring rule* is a function that takes as input an IP $(\boldsymbol{c}, A, \boldsymbol{b})$ and a sequence of cutting planes $h_1, \ldots, h_W$, where each cut lives in one higher dimension than the previous. It measures the quality of this sequence of cutting planes when applied one after the other to the original IP. Every scoring rule $\texttt{score}$ can be naturally extended to a sequential scoring rule $\overline{\texttt{score}}$ defined by $\overline{\texttt{score}}(h_1, \ldots, h_W, (\boldsymbol{c}^0, A^0, \boldsymbol{b}^0)) = \sum_{i=0}^{d-1} \texttt{score}(h_{i+1}, (\boldsymbol{c}^i, A^i, \boldsymbol{b}^i))$, where $(\boldsymbol{c}^i, A^i, \boldsymbol{b}^i)$ is the IP after applying cuts $h_1, \ldots, h_{i-1}$.

Corollary 4.2. *Let $\mathcal{C} = [0,1]^m \times \cdots \times [0,1]^{m+W-1}$ denote the set of possible sequences of W Chvátal-Gomory cut parameters. Let $\texttt{score}_1, \ldots, \texttt{score}_d : \mathcal{C} \times \mathcal{I}_m \times \cdots \times \mathcal{I}_{m+W-1} \to \mathbb{R}$ be d sequential scoring rules and let $f_{\boldsymbol{\mu}}(\boldsymbol{c}, A, \boldsymbol{b})$ be as in Theorem 4.1 for the class $\mathcal{C}$. Then,* $\text{Pdim}(\{f_{\boldsymbol{\mu}}^W : \boldsymbol{\mu} \in \mathbb{R}^d\}) = O(dmW^2 \log(dW(\alpha + \beta + n)))$.

Proof. In Lemma 3.4 and Theorem 3.5 we showed that there are $O(W2^W\alpha + 2^W\beta + nW)$ multivariate polynomials that belong to a family of polynomials $\mathcal{G}$ with $\text{VCdim}(\mathcal{G}^*) \leq 1 + mW$ ($\mathcal{G}^*$ denotes the dual of $\mathcal{G}$) that partition $\mathcal{C}$ into regions such that resulting sequence of cuts is invariant in each region. By Claim 3.5 by Balcan et al. [8], the number of regions is $O(W2^W\alpha + 2^W\beta + nW)^{\text{VCdim}(\mathcal{G}^*)} \leq O(W2^W\alpha + 2^W\beta + nW)^{1+mW}$. The corollary then follows from Theorem 4.1. □

These results bound the sample complexity of learning cut-selection policies based on scoring rules, which allow the cuts B&C that selects to depend on the input IP.

5 Sample complexity of generic tree search

In this section, we study the sample complexity of selecting high-performing parameters for generic tree-based algorithms, which are a generalization of B&C. This abstraction allows us to provide guarantees for simultaneously optimizing key aspects of tree search beyond cut selection, including node selection and branching variable selection. We also generalize the previous sections by allowing actions (such as cut selection) to be taken at any stage of the tree search—not just at the root.

Tree search algorithms take place over a series of κ *rounds* (analogous to the B&B tree-size cap κ in the previous sections). There is a sequence of t *steps* that the algorithm takes on each round. For example, in B&C, these steps include node selection, cut selection, and variable selection. The specific *action* the algorithm takes during each step (for example, which node to select, which cut to include, or which variable to branch on) typically depends on a *scoring rule*. This scoring rule weights each possible action and the algorithm performs the action with the highest weight. These actions (deterministically) transition the algorithm from one *state* to another. This high-level description of tree search is summarized by Algorithm 1. For each step $j \in [t]$, the number of possible actions is $T_j \in \mathbb{N}$. There is a scoring rule $\texttt{score}_j$, where $\texttt{score}_j(k, s) \in \mathbb{R}$ is the weight associated with the action $k \in [T_j]$ when the algorithm is in the state s.

There are often several scoring rules one could use, and it is not clear which to use in which scenarios. As in Section 4, we provide guarantees for learning combinations of these scoring rules for the particular application at hand. More formally, for each step $j \in [t]$, rather than just a single scoring rule $\texttt{score}_j$ as in Step 4, there are d_j scoring rules $\texttt{score}_{j,1}, \ldots, \texttt{score}_{j,d_j}$. Given parameters $\boldsymbol{\mu}_j = (\mu_j[1], \ldots, \mu_j[d_j]) \in \mathbb{R}^{d_j}$, the algorithm takes the action $k \in [T_j]$ that maximizes $\sum_{i=1}^{d_j} \mu_j[i]\texttt{score}_{j,i}(k, s)$. There is a distribution $\mathcal{D}$ over inputs x to Algorithm 1. For example, when this framework is instantiated for branch-and-cut, x is an integer program $(\boldsymbol{c}, A, \boldsymbol{b})$. There is a utility function $f_{\boldsymbol{\mu}}(x) \in [-H, H]$ that measures the utility of the algorithm parameterized by

Algorithm 1 Tree search

Input: Problem instance, t scoring rules $\texttt{score}_1, \ldots, \texttt{score}_t$, number of rounds κ.

$s_{1,1} \leftarrow$ Initial state of algorithm
for each round $i \in [\kappa]$ **do**
  for each step $j \in [t]$ **do**
    Perform the action $k \in [T_j]$ that maximizes $\texttt{score}_j\left(s_{i,j}, k\right)$
    $s_{i,j+1} \leftarrow$ New state of algorithm
  $s_{i+1,1} \leftarrow s_{i,t+1}$ ▷ State at beginning of next round equals state at end of this round

Output: Incumbent solution in state $s_{\kappa,t+1}$, if one exists.

$\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_t)$ on input x. For example, this utility function might measure the size of the search tree that the algorithm builds. We assume that this utility function is *final-state-constant*:

Definition 5.1. Let $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_t)$ and $\boldsymbol{\mu}' = (\boldsymbol{\mu}'_1, \ldots, \boldsymbol{\mu}'_t)$ be two parameter vectors. Suppose that we run Algorithm 1 on input x once using the scoring rule $\texttt{score}_j = \sum_{i=1}^{d_j} \mu_j[i]\texttt{score}_{j,i}$ and once using the scoring rule $\texttt{score}_j = \sum_{i=1}^{d_j} \mu'_j[i]\texttt{score}_{j,i}$. Suppose that on each run, we obtain the same final state $s_{\kappa,t+1}$. The utility function is *final-state-constant* if $f_{\boldsymbol{\mu}}(x) = f_{\boldsymbol{\mu}'}(x)$.

We provide a sample complexity bound for learning the parameters $\boldsymbol{\mu}$. The full proof is in Appendix C.

Theorem 5.2. *Let $d = \sum_{j=1}^{t} d_j$ denote the total number of tunable parameters of tree search. Then,*

$$\text{Pdim}\left(\left\{f_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathbb{R}^d\right\}\right) = O\left(d\kappa \sum_{j=1}^{t} \log T_j + d \log d\right).$$

Proof sketch. We prove that there is a set of hyperplanes splitting the parameter space into regions such that if tree search uses any parameter setting from a single region, it will always take the same sequence of actions (including node, variable, and cut selection). The main subtlety is an induction argument to count these hyperplanes that depends on the current step of the tree-search algorithm. □

In the context of integer programming, Theorem 5.2 not only recovers the main result of Balcan et al. [5] for learning variable selection policies, but also yields a more general bound that simultaneously incorporates cutting plane selection, variable selection, and node selection. In B&C, the first action of each round is to select a node. Since there are at most $2^{n+1} - 1$ nodes, $T_1 \leq 2^{n+1} - 1$. The second action is to choose a cutting plane. As in Theorem 4.1, let $\mathcal{C}$ be a family of cutting planes such that for every IP $(\boldsymbol{c}, A, \boldsymbol{b})$, there is a decomposition of the parameter space into $\leq r$ regions such that the cuts generated by any two parameters in the same region are the same. So $T_2 \leq r$. The last action is to choose a variable to branch on at that node, so $T_3 = n$. Applying Theorem 5.2, $\text{Pdim}(\{f_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathbb{R}^d\}) = O(d\kappa n + d\kappa \log r + d \log d)$. Ignoring T_1 and T_2, thereby only learning the variable selection policy, recovers the $O(d\kappa \log n + d \log d)$ bound of Balcan et al. [5].

6 Conclusions and future research

We provided the first provable guarantees for using machine learning to configure cutting planes and cut-selection policies. We analyzed the sample complexity of learning cutting planes from the popular family of Chvátal-Gomory (CG) cuts. We then provided sample complexity guarantees for learning parameterized cut-selection policies, which allow the branch-and-cut algorithm to adaptively apply cuts as it builds the search tree. We showed that this analysis can be generalized to simultaneously capture various key aspects of tree search beyond cut selection, such as node and variable selection.

This paper opens up a variety questions for future research. For example, which other cut families can we learn over with low sample complexity? Section 3 focused on learning within the family of CG cuts (Sections 4 and 5 applied more generally). There are many other families, such as *Gomory mixed-integer cuts* and *lift-and-project cuts*, and a sample complexity analysis of these is an interesting direction for future research (and would call for new techniques). In addition, can we use machine learning to design improved scoring rules and heuristics for cut selection?

References

[1] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

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

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

[4] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996.

[5] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning (ICML)*, 2018.

[6] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Learning to optimize computational resources: Frugal training with generalization guarantees. In *AAAI Conference on Artificial Intelligence*, 2020.

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

[8] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. 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.

[9] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

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

[11] Gérard Cornuéjols and Yanjun Li. Elementary closures for integer programs. *Operations Research Letters*, 28(1):1–8, 2001.

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

[13] Richard Dudley. Universal Donsker classes and metric entropy. *The Annals of Probability*, 15 (4):1306–1326, 1987.

[14] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. MIPaaL: Mixed integer program as a layer. In *AAAI Conference on Artificial Intelligence*, 2020.

[15] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL `http://www.optimization-online.org/DB_HTML/2020/03/7705.html`.

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

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

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

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

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

[21] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC—instance-specific algorithm configuration. In *European Conference on Artificial Intelligence (ECAI)*, 2010.

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

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

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

[25] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

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

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

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

[29] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4):1–52, 2009. ISSN 0004-5411.

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

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

[32] David Pollard. *Convergence of Stochastic Processes*. Springer, 1984.

[33] Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with UCT. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2012.

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

[35] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. *International Conference on Machine Learning (ICML)*, 2020.

[36] Gellért Weisz, András György, and Csaba Szepesvári. LEAPSANDBOUNDS: A method for approximately optimal algorithm configuration. In *International Conference on Machine Learning (ICML)*, 2018.

[37] Gellért Weisz, András György, and Csaba Szepesvári. CAPSANDRUNS: An improved method for approximately optimal algorithm configuration. *International Conference on Machine Learning (ICML)*, 2019.

[38] Franz Wesselmann and Uwe Suhl. Implementing cutting plane management and selection techniques. Technical report, University of Paderborn, 2012.

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

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

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

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We discussed the limitations of our work in our Conclusions and future research section.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] In Section 2 we go through all the technical background required to state our results. We formally lay out the foundations of the cutting planes we study, as well as the learning theory background required. We provide detailed, and precise, formal statements of all our results in the paper.
   (b) Did you include complete proofs of all theoretical results? [Yes] We provide full, detailed proofs of all results. Due to space constraints, the main body includes proof sketches of some of the results, and the full proofs of these results are in the appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]