Learning to Optimize Computational Resources: Frugal Training with Generalization Guarantees

Maria-Florina Balcan
Carnegie Mellon University
ninamf@cs.cmu.edu

Tuomas Sandholm
Carnegie Mellon University
Optimized Markets, Inc.
Strategic Machine, Inc.
Strategy Robot, Inc.
sandholm@cs.cmu.edu

Ellen Vitercik
Carnegie Mellon University
vitercik@cs.cmu.edu

September 10, 2019

Abstract

Algorithms typically come with tunable parameters that have a considerable impact on the computational resources they consume. Too often, practitioners must hand-tune the parameters, a tedious and error-prone task. A recent line of research provides algorithms that return nearly-optimal parameters from within a finite set. These algorithms can be used when the parameter space is infinite by providing as input a random sample of parameters. This data-independent discretization, however, might miss pockets of nearly-optimal parameters: prior research has presented scenarios where the only viable parameters lie within an arbitrarily small region. We provide an algorithm that learns a finite set of promising parameters from within an infinite set. Our algorithm can help compile a configuration portfolio, or it can be used to select the input to a configuration algorithm for finite parameter spaces. Our approach applies to any configuration problem that satisfies a simple yet ubiquitous structure: the algorithm's performance is a piecewise constant function of its parameters. Prior research has exhibited this structure in domains from integer programming to clustering.

1 Introduction

Similar combinatorial problems often arise in seemingly unrelated disciplines. Integer programs, for example, model problems in fields ranging from computational biology to economics. To facilitate customization, algorithms often come with tunable parameters that significantly impact the computational resources they consume, such as runtime. Hand-tuning parameters can be time consuming and may lead to sub-optimal results. In this work, we develop the foundations of automated algorithm configuration via machine learning. A key challenge we face is that in order to evaluate a configuration's requisite computational resources, the learning algorithm itself must expend those resources.

To frame algorithm configuration as a machine learning problem, we assume sample access to an unknown distribution over problem instances, such as the integer programs an airline solves day to day. The learning algorithm uses samples to determine parameters that, ideally, will have strong expected performance. Researchers have studied this configuration model for decades, leading to advances in artificial intelligence [Xu et al., 2008] and computational biology [DeBlasio and Kececioglu, 2018]. This approach has also been used in industry for tens of billions of dollars of combinatorial auctions [Sandholm, 2013].

1

Recently, two lines of research have emerged that explore the theoretical underpinnings of algorithm configuration. One provides sample complexity guarantees, bounding the number of samples sufficient to ensure that an algorithm's performance on average over the samples generalizes to its expected performance on the distribution [Gupta and Roughgarden, 2017, Balcan et al., 2017, 2018a]. These sample complexity bounds apply no matter how the learning algorithm operates, and these papers do not include learning algorithms that extend beyond exhaustive search.

The second line of research provides algorithms for finding nearly-optimal configurations from a finite set [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019]. These algorithms can also be used when the parameter space is infinite: for any $\gamma \in (0,1)$, first sample $\tilde{\Omega}(1/\gamma)$ configurations, and then run the algorithm over this finite set. The authors guarantee that the output configuration will be within the top γ-quantile. If there is only a small region of high-performing parameters, however, the uniform sample might completely miss all good parameters. Algorithm configuration problems with only tiny pockets of high-performing parameters do indeed exist: Balcan et al. [2018a] present distributions over integer programs where the optimal parameters lie within an arbitrarily small region of the parameter space. For any parameter within that region, branch-and-bound—the most widely-used integer programming algorithm—terminates instantaneously. Using any other parameter, branch-and-bound takes an exponential number of steps. This region of optimal parameters can be made so small that any random sampling technique would require an arbitrarily large sample of parameters to hit that region. We discuss this example in more detail in Section 5.

This paper marries these two lines of research. We present an algorithm that identifies a finite set of promising parameters within an infinite set, given sample access to a distribution over problem instances. We prove that this set contains a nearly optimal parameter with high probability. The set can serve as the input to a configuration algorithm for finite parameter spaces [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019], which we prove will then return a nearly optimal parameter from the infinite set.

An obstacle in our approach is that the loss function measuring an algorithm's performance as a function of its parameters often exhibits jump discontinuities: a nudge to the parameters can trigger substantial changes in the algorithm's behavior. As a result, we cannot apply existing techniques with provable guarantees for tuning continuous parameters. For example, configuration algorithms based on Bayesian optimization typically require smooth loss functions, as quantified by its Lipschitz constant [Brochu et al., 2010] or its RKHS norm [Berkenkamp et al., 2019]. Instead, we must tease out useful structure in the configuration problems we study.

The structure we identify is simple yet ubiquitous in combinatorial domains: our approach applies to any configuration problem where the algorithm's performance as a function of its parameters is piecewise constant. Prior research has demonstrated that algorithm configuration problems from diverse domains exhibit this structure. For example, Balcan et al. [2018a] uncovered this structure for branch-and-bound algorithm configuration. Many corporations must regularly solve reams of integer programs, and therefore require highly customized solvers. For example, integer programs are a part of many mesh processing pipelines in computer graphics [Bommes et al., 2009]. Animation studios with thousands of meshes require carefully tuned solvers which, thus far, domain experts have handcrafted [Bommes et al., 2010]. Our algorithm can be used to find configurations that minimize the branch-and-bound tree size. Balcan et al. [2017] also exhibit this piecewise-constant structure in the context of linkage-based hierarchical clustering algorithms. The algorithm families they study interpolate between the classic single-, complete-, and average-linkage procedures. Building the cluster hierarchy is expensive: the best-known algorithm's runtime is $\tilde{O}(n^2)$ given n datapoints [Manning et al., 2010]. As with branch-and-bound, our algorithm finds configurations that return satisfactory clusterings while minimizing the hierarchy tree size.

We now describe our algorithm at a high level. Let ℓ be a loss function where $\ell(\boldsymbol{\rho}, j)$ measures the computational resources (running time, for example) required to solve problem instance j using the algorithm parameterized by the vector $\boldsymbol{\rho}$. Let OPT be the smallest expected loss[1] $\mathbb{E}_{j \sim \Gamma}[\ell(\boldsymbol{\rho}, j)]$ of any parameter $\boldsymbol{\rho}$, where Γ is an unknown distribution over problem instances. Our algorithm maintains upper confidence bound on OPT, initially set to ∞. On each round t, the algorithm begins by drawing a set $\mathcal{S}_t$ of sample problem instances. It computes the partition of the parameter space into regions where for each problem instance in $\mathcal{S}_t$, the loss ℓ, capped at 2^t, is a constant function of the parameters. On a given region of this partition, if the average capped loss is sufficiently low, the algorithm chooses an arbitrary parameter from that region and deems it "good." Once the cap 2^t has grown sufficiently large compared to the upper confidence bound on OPT, the algorithm returns the set of good parameters. We summarize our guarantees informally below.

Theorem 1.1 (Informal)**.** *The following guarantees hold:*

1. *The set of output parameters contains a nearly-optimal parameter with high probability.*

2. *Given accuracy parameters ϵ and δ, the algorithm terminates after $O\left(\ln\left(\sqrt[4]{1+\epsilon} \cdot OPT/\delta\right)\right)$ rounds.*

3. *On the algorithm's final round, let P be the size of the partition the algorithm computes. The number of parameters it outputs is $O\left(P \cdot \ln\left(\sqrt[4]{1+\epsilon} \cdot OPT/\delta\right)\right)$.*

4. *The algorithm's sample complexity on each round t is polynomial in 2^t (which scales linearly with OPT), $\log P$, the parameter space dimension, $\frac{1}{\delta}$, and $\frac{1}{\epsilon}$.*

In Section 5, we prove that our algorithm's sample complexity can be exponentially better than that of the best-known uniform convergence bound. Moreover, it can find strong configurations in scenarios where uniformly sampling configurations will fail.

In Appendix C.1, we survey additional related research.

2 Problem definition

The algorithm configuration model we adopt is a generalization of the model from prior research by Kleinberg et al. [2017, 2019] and Weisz et al. [2018, 2019]. There is a set Π of problem instances and an unknown distribution Γ over Π. For example, this distribution might represent the integer programs an airline solves day to day. Each algorithm is parameterized by a vector $\boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^d$. At a high level, we assume we can set a budget on the computational resources the algorithm consumes, which we quantify using an integer $\tau \in \mathbb{Z}_{\geq 0}$. For example, τ might measure the maximum running time we allow the algorithm. There is a utility function $u : \mathcal{P} \times \Pi \times \mathbb{Z}_{\geq 0} \to \{0, 1\}$, where $u(\boldsymbol{\rho}, j, \tau) = 1$ if and only if the algorithm parameterized by $\boldsymbol{\rho}$ returns a solution to the instance j given a budget of τ. We make the natural assumption that the algorithm is more likely to find a solution the higher its budget: $u(\boldsymbol{\rho}, j, \tau) \geq u(\boldsymbol{\rho}, j, \tau')$ for $\tau \geq \tau'$. Finally, there is a loss function $\ell : \mathcal{P} \times \Pi \to \mathbb{Z}_{\geq 0}$ which measures the minimum budget the algorithm requires to find a solution. Specifically, $\ell(\boldsymbol{\rho}, j) = \infty$ if $u(\boldsymbol{\rho}, j, \tau) = 0$ for all τ, and otherwise, $\ell(\boldsymbol{\rho}, j) = \operatorname{argmin}\{\tau : u(\boldsymbol{\rho}, j, \tau) = 1\}$. In Section 2.1, we provide several examples of this problem definition instantiated for combinatorial problems.

---

[1]As we describe in Section 2, we compete with a slightly more nuanced benchmark than OPT, in line with prior research.
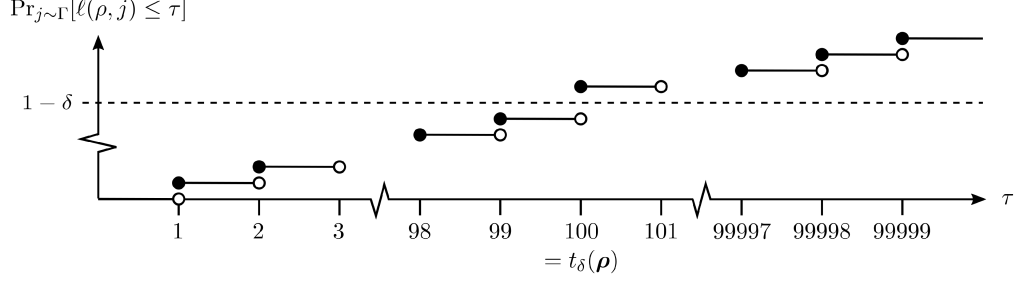
Figure 1: Fix a parameter vector $\boldsymbol{\rho}$. The figure is a hypothetical illustration of the cumulative density function of $\ell(\boldsymbol{\rho}, j)$ when j is sampled from Γ. For each value τ along the x-axis, the solid line equals $\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}, j) \leq \tau]$. The dotted line equals the constant function $1 - \delta$. Since 100 is the largest integer such that $\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}, j) \geq 100] \geq \delta$, we have that $t_\delta(\boldsymbol{\rho}) = 100$.

The distribution Γ over problem instances is unknown, so we use samples from Γ to find a parameter vector $\hat{\boldsymbol{\rho}} \in \mathcal{P}$ with small expected loss. Ideally, we could guarantee that

$$\mathbb{E}_{j\sim\Gamma}\left[\ell\left(\hat{\boldsymbol{\rho}}, j\right)\right] \leq (1 + \epsilon) \inf_{\boldsymbol{\rho}\in\mathcal{P}}\left\{\mathbb{E}_{j\sim\Gamma}\left[\ell\left(\boldsymbol{\rho}, j\right)\right]\right\}. \tag{1}$$

Unfortunately, this ideal goal is impossible to achieve with a finite number of samples, even in the extremely simple case where there are only two configurations, as illustrated below.

Example 2.1. [Weisz et al. [2019]] Let $\mathcal{P} = \{1, 2\}$ be a set of two configurations. Suppose that the loss of the first configuration is 2 for all problem instances: $\ell(1, j) = 2$ for all $j \in \Pi$. Meanwhile, suppose that $\ell(2, j) = \infty$ with probability δ for some $\delta \in (0, 1)$ and $\ell(2, j) = 1$ with probability $1 - \delta$. In this case, $\mathbb{E}_{j\sim\Gamma}[\ell(1, j)] = 2$ and $\mathbb{E}_{j\sim\Gamma}[\ell(2, j)] = \infty$. In order for any algorithm to verify that the first configuration's expected loss is substantially better than the second's, it must sample at least one problem instance j such that $\ell(2, j) = \infty$. Therefore, it must sample $\Omega(1/\delta)$ problem instances, a lower bound that approaches infinity as δ shrinks. As a result, it is impossible to give a finite bound on the number of samples sufficient to find a parameter $\hat{\boldsymbol{\rho}}$ that satisfies Equation (1).

The obstacle that this example exposes is that some configurations might have an enormous loss on a few rare problem instances. To deal with this impossibility result, Weisz et al. [2018, 2019], building off of work by Kleinberg et al. [2017, 2019], propose a relaxed notion of approximate optimality. To describe this relaxation, we introduce the following notation. Given $\delta \in (0, 1)$ and a parameter vector $\boldsymbol{\rho} \in \mathcal{P}$, let $t_\delta(\boldsymbol{\rho})$ be the largest cutoff $\tau \in \mathbb{Z}_{\geq 0}$ such that the probability $\ell(\boldsymbol{\rho}, j)$ is greater than τ is at least δ. Mathematically, $t_\delta(\boldsymbol{\rho}) = \operatorname{argmax}_{\tau\in\mathbb{Z}}\{\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}, j) \geq \tau] \geq \delta\}$. The value $t_\delta(\boldsymbol{\rho})$ can be thought of as the beginning of the loss function's "δ-tail." We illustrate the definition of $t_\delta(\boldsymbol{\rho})$ in Figure 1. We now define the relaxed notion of approximate optimality by Weisz et al. [2018].

Definition 2.1 (($\epsilon, \delta, \mathcal{P}$)-optimality). A parameter vector $\hat{\boldsymbol{\rho}}$ is ($\epsilon, \delta, \mathcal{P}$)-*optimal* if

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\hat{\boldsymbol{\rho}}, j\right), t_\delta\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \leq (1 + \epsilon)\inf_{\boldsymbol{\rho}\in\mathcal{P}}\left\{\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}, j\right), t_{\delta/2}\left(\boldsymbol{\rho}\right)\right\}\right]\right\}.$$

In other words, a parameter vector $\hat{\boldsymbol{\rho}}$ is ($\epsilon, \delta, \mathcal{P}$)-*optimal* if its δ-capped expected loss is within a $(1 + \epsilon)$-factor of the optimal $\delta/2$-capped expected loss.[2] To condense notation, we write $OPT_{c\delta} :=$

---

[2]The fraction $\delta/2$ can be replaced with any $c\delta$ for $c \in (0, 1)$. Ideally, we would replace $\delta/2$ with δ, but the resulting property would be impossible to verify with high probability [Weisz et al., 2018].

$\inf_{\boldsymbol{\rho} \in \mathcal{P}} \{\mathbb{E}_{j \sim \Gamma} [\min \{\ell(\boldsymbol{\rho}, j), t_{c\delta}(\boldsymbol{\rho})\}]\}$. If an algorithm returns an $(\epsilon, \delta, \bar{\mathcal{P}})$-optimal parameter from within a finite set $\bar{\mathcal{P}}$, we call it a *configuration algorithm for finite parameter spaces*. Weisz et al. [2019] provide one such algorithm, CAPSANDRUNS.

2.1 Example applications

In this section, we provide several instantiations of our problem definition in combinatorial domains.

Tree search. Tree search algorithms, such as branch-and-bound, are the most widely-used tools for solving combinatorial problems, such as (mixed) integer programs and constraint satisfaction problems. These algorithms recursively partition the search space to find an optimal solution, organizing this partition as a tree. Commercial solvers such as CPLEX, which use tree search under the hood, come with hundreds of tunable parameters. Researchers have developed machine learning algorithms for tuning these parameters [Hutter et al., 2009, Dickerson and Sandholm, 2013, He et al., 2014, Balafrej et al., 2015, Khalil et al., 2016, 2017, Kruber et al., 2017, Xia and Yap, 2018, Balcan et al., 2018a]. Given parameters $\boldsymbol{\rho}$ and a problem instance j, we might define the budget τ to cap the size of the tree the algorithm builds. In that case, the utility function is defined such that $u(\boldsymbol{\rho}, j, \tau) = 1$ if and only if the algorithm terminates, having found the optimal solution, after building a tree of size τ. The loss $\ell(\boldsymbol{\rho}, j)$ equals the size of the entire tree built by the algorithm parameterized by $\boldsymbol{\rho}$ given the instance j as input.

Clustering. Given a set of datapoints and the distances between each point, the goal in clustering is to partition the points into subsets so that points within any set are "similar." Clustering algorithms are used to group proteins by function, classify images by subject, and myriad other applications. Typically, the quality of a clustering is measured by an objective function, such as the classic k-means, k-median, or k-center objectives. Unfortunately, it is NP-hard to determine the clustering that minimizes any of these objectives. As a result, researchers have developed a wealth of approximation and heuristic clustering algorithms. However, no one algorithm is optimal across all applications.

Balcan et al. [2017] provide sample complexity guarantees for clustering algorithm configuration. Each problem instance is a set of datapoints and there is a distribution over clustering problem instances. They analyze several infinite classes of clustering algorithms. Each of these algorithms begins with a linkage-based step and concludes with a dynamic programming step. The linkage-based routine constructs a hierarchical tree of clusters. At the beginning of the process, each datapoint is in a cluster of its own. The algorithm sequentially merges the clusters into larger clusters until all elements are in the same cluster. There are many ways to build this tree: merge the clusters that are closest in terms of their two closest points (*single-linkage*), their two farthest points (*complete-linkage*), or on average over all pairs of points (*average-linkage*). These linkage procedures are commonly used in practice [Awasthi et al., 2017, Saeed et al., 2003, White et al., 2010] and come with theoretical guarantees. Balcan et al. [2017] study an infinite parameterization, *ρ-linkage*, that interpolates between single-, average-, and complete-linkage. After building the cluster tree, the dynamic programming step returns the pruning of this tree that minimizes a fixed objective function, such as the k-means, k-median, or k-center objectives.

Building the full hierarchy is expensive because the best-known algorithm's runtime is $O(n^2 \log n)$, where n is the number of datapoints [Manning et al., 2010]. It is not always necessary, however, to build the entire tree: the algorithm can preemptively terminate the linkage step after τ merges, then use dynamic programming to recover the best pruning of the cluster forest. We refer to this variation as *τ-capped ρ-linkage*. To evaluate the resulting clustering, we assume there is a cost function

$c : \mathcal{P} \times \Pi \times \mathbb{Z} \to \mathbb{R}$ where $c(\rho, j, \tau)$ measures the quality of the clustering τ-capped ρ-linkage returns, given the instance j as input. We assume there is a threshold θ_j where the clustering is admissible if and only if $c(\rho, j, \tau) \leq \theta_j$, which means the utility function is defined as $u(\rho, j, \tau) = \mathbf{1}_{\{c(\rho, j, \tau) \leq \theta_j\}}$. For example, $c(\rho, j, \tau)$ might measure the clustering's k-means objective value, and θ_j might equal the optimal k-means objective value (obtained only for the training instances via an expensive computation) plus an error term.

3 Data-dependent discretizations of infinite parameter spaces

We begin this section by proving an intuitive fact: given a finite subset $\bar{\mathcal{P}} \subset \mathcal{P}$ of parameters that contains at least one "sufficiently good" parameter, a configuration algorithm for finite parameter spaces, such as CapsAndRuns [Weisz et al., 2019][3], returns a parameter that's nearly optimal over the infinite set $\mathcal{P}$. Therefore, our goal is to provide an algorithm that takes as input an infinite parameter space and returns a finite subset that contains at least one good parameter. A bit more formally, a parameter is "sufficiently good" if its $\delta/2$-capped expected loss is within a $\sqrt{1 + \epsilon}$-factor of $OPT_{\delta/4}$. We say a finite parameter set $\bar{\mathcal{P}}$ is an (ϵ, δ)-optimal subset if it contains a good parameter.

Definition 3.1 ((ϵ, δ)-optimal subset)**.** A finite set $\bar{\mathcal{P}} \subset \mathcal{P}$ is an (ϵ, δ)-optimal subset if there is a vector $\hat{\rho} \in \bar{\mathcal{P}}$ such that $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\hat{\rho}, j), t_{\delta/2}(\hat{\rho}) \right\} \right] \leq \sqrt{1 + \epsilon} \cdot OPT_{\delta/4}$.

We now prove that given an (ϵ, δ)-optimal subset $\bar{\mathcal{P}} \subset \mathcal{P}$, a configuration algorithm for finite parameter spaces returns a nearly optimal parameter from the infinite space $\mathcal{P}$.

Theorem 3.1. Let $\bar{\mathcal{P}} \subset \mathcal{P}$ be an (ϵ, δ)-optimal subset and let $\epsilon' = \sqrt{1 + \epsilon} - 1$. Suppose $\hat{\rho} \in \bar{\mathcal{P}}$ is $\left(\epsilon', \delta, \bar{\mathcal{P}} \right)$-optimal. Then $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\hat{\rho}, j), t_{\delta}(\hat{\rho}) \right\} \right] \leq (1 + \epsilon) \cdot OPT_{\delta/4}$.

Proof. Since the parameter $\hat{\rho}$ is $\left(\epsilon', \delta, \bar{\mathcal{P}} \right)$-optimal, we know that $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\hat{\rho}, j), t_{\delta}(\hat{\rho}) \right\} \right] \leq \sqrt{1 + \epsilon} \cdot \min_{\rho \in \bar{\mathcal{P}}} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\rho, j), t_{\delta/2}(\rho) \right\} \right] \right\}$. (We use a minimum instead of an infimum because $\bar{\mathcal{P}}$ is a finite set by Definition 3.1.) The set $\bar{\mathcal{P}}$ is an (ϵ, δ)-optimal subset of the parameter space $\mathcal{P}$, so there exists a parameter vector $\rho' \in \bar{\mathcal{P}}$ such that $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\rho', j), t_{\delta/2}(\rho') \right\} \right] \leq \sqrt{1 + \epsilon} \cdot OPT_{\delta/4}$. Therefore,

$$\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\hat{\rho}, j), t_{\delta}(\hat{\rho}) \right\} \right] \leq \sqrt{1 + \epsilon} \cdot \min_{\rho \in \bar{\mathcal{P}}} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\rho, j), t_{\delta/2}(\rho) \right\} \right] \right\}$$
$$\leq \sqrt{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\rho', j), t_{\delta/2}(\rho') \right\} \right]$$
$$\leq (1 + \epsilon) \cdot OPT_{\delta/4},$$

so the theorem statement holds. $\qquad \square$

4 Our main result: An algorithm for learning (ϵ, δ)-optimal subsets

We present an algorithm for learning (ϵ, δ)-optimal subsets for configuration problems that satisfy a simple, yet ubiquitous structure: for any problem instance j, the loss function $\ell(\cdot, j)$ is piecewise constant. This structure has been observed throughout a diverse array of configuration problems ranging from clustering to integer programming [Balcan et al., 2017, 2018a]. A bit more formally, this structure holds if for any problem instance $j \in \Pi$ and cap $\tau \in \mathbb{Z}_{\geq 0}$, there is a finite partition

---

[3]We believe our algorithm can easily be extended to precede the algorithms by Kleinberg et al. [2017, 2019], who employ a slightly different notion of approximate optimality. We leave this extension to future work.

Algorithm 1 Algorithm for learning (ϵ, δ)-optimal subsets

Input: Parameters $\delta, \zeta \in (0,1), \epsilon > 0$.

1: Set $\eta \leftarrow \min\left\{\frac{1}{8}\left(\sqrt[4]{1+\epsilon} - 1\right), \frac{1}{9}\right\}$, $t \leftarrow 1$, $T \leftarrow \infty$, and $\mathcal{G} \leftarrow \emptyset$.
2: **while** $2^{t-3}\delta < T$ **do**
3:     Set $\mathcal{S}_t \leftarrow \{j\}$, where $j \sim \Gamma$.
4:     **while** $\sqrt{\frac{2d\ln f(\mathcal{S}_t, 2^t)}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8(2^t|\mathcal{S}_t|t)^2}{\zeta}} > \eta\delta$ **do** Draw $j \sim \Gamma$ and add j to $\mathcal{S}_t$.
5:     Compute tuples $(\mathcal{P}_1, z_1, \boldsymbol{\tau}_1), \ldots, (\mathcal{P}_k, z_k, \boldsymbol{\tau}_k) \leftarrow \textsc{GetPartition}\left(\mathcal{S}_t, 2^t\right)$.
6:     **for** $i \in \{1, \ldots, k\}$ with $z_i \geq 1 - 3\delta/8$ **do**
7:         Set $\mathcal{G} \leftarrow \mathcal{G} \cup \{\mathcal{P}_i\}$.
8:         Sort the elements of $\boldsymbol{\tau}_i$: $\tau_1 \leq \cdots \leq \tau_{|\mathcal{S}_t|}$.
9:         Set $T' \leftarrow \frac{1}{|\mathcal{S}_t|}\sum_{m=1}^{|\mathcal{S}_t|}\min\left\{\tau_m, \tau_{\lfloor|\mathcal{S}_t|(1-3\delta/8)\rfloor}\right\}$.
10:         **if** $T' < T$ **then** Set $T \leftarrow T'$.
11:     $t \leftarrow t + 1$.
12: For each set $\mathcal{P}' \in \mathcal{G}$, select a vector $\boldsymbol{\rho}_{\mathcal{P}'} \in \mathcal{P}'$.

Output: The (ϵ, δ)-optimal set $\{\boldsymbol{\rho}_{\mathcal{P}'} \mid \mathcal{P}' \in \mathcal{G}\}$.

---

of the parameter space $\mathcal{P}$ such that in any one region R of this partition, for all pairs of parameter vectors $\boldsymbol{\rho}, \boldsymbol{\rho}' \in R$, $\min\{\ell(\boldsymbol{\rho}, j), \tau\} = \min\{\ell(\boldsymbol{\rho}', j), \tau\}$. The number of samples our algorithm requires depends on the size of this partition, which we quantify as follows.

Definition 4.1 (f-piecewise configuration problem). Let $f : 2^\Pi \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be a function mapping subsets $\mathcal{S} \subseteq \Pi$ of problem instances and caps $\tau \in \mathbb{Z}_{\geq 0}$ to integers $f(\mathcal{S}, \tau) \in \mathbb{Z}_{\geq 0}$. We require that f is monotone in both of its parameters: for all $\tau \leq \tau'$, $f(\mathcal{S}, \tau) \leq f(\mathcal{S}, \tau')$ and for all subsets $\mathcal{S}' \subseteq \mathcal{S} \subseteq \Pi$, $f(\mathcal{S}', \tau) \leq f(\mathcal{S}, \tau)$. We say that a configuration problem $(\Pi, \mathcal{P}, \ell)$ is f-*piecewise* if for all subsets $\mathcal{S} \subseteq \Pi$ and all caps $\tau \in \mathbb{Z}_{\geq 0}$, there exists a partition $\mathcal{P}_1, \ldots, \mathcal{P}_k$ of the parameter space $\mathcal{P}$ of size $k = f(\mathcal{S}, \tau)$ such that for all subsets $\mathcal{P}_i$, all problem instances $j \in \mathcal{S}$, and all parameter pairs $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathcal{P}_i$, $\min\{\ell(\boldsymbol{\rho}, j), \tau\} = \min\{\ell(\boldsymbol{\rho}', j), \tau\}$.

We require access to a function $\textsc{GetPartition}$ that takes as input a set $\mathcal{S}$ of problem instances and an integer τ and returns the partition referred to in Definition 4.1. Namely, it returns a set of tuples $(\mathcal{P}_1, z_1, \boldsymbol{\tau}_1), \ldots, (\mathcal{P}_k, z_k, \boldsymbol{\tau}_k) \in 2^{\mathcal{P}} \times [0,1] \times \mathbb{Z}^{|\mathcal{S}|}$ with $k = f(\mathcal{S}, \tau)$ such that:

1. The sets $\mathcal{P}_1, \ldots, \mathcal{P}_k$ make up a partition of $\mathcal{P}$.

2. For all subsets $\mathcal{P}_i$ and vectors $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathcal{P}_i$, $\frac{1}{|\mathcal{S}|}\sum_{j \in \mathcal{S}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq \tau\}} = \frac{1}{|\mathcal{S}|}\sum_{j \in \mathcal{S}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}', j) \leq \tau\}} = z_i$.

3. For all subsets $\mathcal{P}_i$, all $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathcal{P}_i$, and all $j \in \mathcal{S}$, $\min\{\ell(\boldsymbol{\rho}, j), \tau\} = \min\{\ell(\boldsymbol{\rho}', j), \tau\} = \tau_i[j]$.

As we describe in Appendix C, results from prior research imply bounds on the function f and guidance for implementing $\textsc{GetPartition}$ in the contexts of clustering and integer programming. For example, in the clustering application we describe in Section 2.1, the distribution Γ is over clustering instances. Suppose n is an upper bound on the number of points in each instance. Balcan et al. [2017] prove that for any set $\mathcal{S}$ of samples and any cap τ, in the worst case, $f(\mathcal{S}, \tau) = O\left(|\mathcal{S}|n^8\right)$, though empirically, $f(\mathcal{S}, \tau)$ is often several orders of magnitude smaller [Balcan et al., 2019a]. Balcan et al. [2017] and Balcan et al. [2019a] provide guidance for implementing $\textsc{GetPartition}$.

High-level description of algorithm. We now describe our algorithm for learning (ϵ, δ)-optimal subsets. See Algorithm 1 for the pseudocode. The algorithm maintains a variable T, initially set

to ∞, which roughly represents an upper confidence bound on $OPT_{\delta/4}$. It also maintains a set $\mathcal{G}$ of parameters which the algorithm believes might be nearly optimal. The algorithm begins by aggressively capping the maximum loss ℓ it computes by 1. At the beginning of each round, the algorithm doubles this cap until the cap grows sufficiently large compared to the upper confidence bound T. At that point, the algorithm terminates. On each round t, the algorithm draws a set $\mathcal{S}_t$ of samples (Step 4) that is just large enough to estimate the expected 2^t-capped loss $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), 2^t\right\}\right]$ for every parameter $\boldsymbol{\rho} \in \mathcal{P}$. The number of samples it draws is a data-dependent quantity that depends on *empirical Rademacher complexity* [Koltchinskii, 2001, Bartlett and Mendelson, 2002] (see Appendix A for a refresher on this standard learning theoretic notion).

Next, the algorithm uses the function GETPARTITION to obtain the tuples

$$(\mathcal{P}_1, z_1, \boldsymbol{\tau}_1), \ldots, (\mathcal{P}_k, z_k, \boldsymbol{\tau}_k) \in 2^{\mathcal{P}} \times [0,1] \times \mathbb{Z}^{|\mathcal{S}_t|}$$

with $k = f\left(\mathcal{S}_t, 2^t\right)$. By definition of this function, for all subsets $\mathcal{P}_i$ and parameter vector pairs $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathcal{P}_i$, the fraction of instances $j \in \mathcal{S}_t$ with $\ell(\boldsymbol{\rho}, j) \leq 2^t$ is equal to the fraction of instances $j \in \mathcal{S}_t$ with $\ell(\boldsymbol{\rho}', j) \leq 2^t$. In other words, $\frac{1}{|\mathcal{S}_t|}\sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq 2^t\}} = \frac{1}{|\mathcal{S}_t|}\sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho}', j) \leq 2^t\}} = z_i$. If this fraction is sufficiently high (at least $1 - 3\delta/8$), the algorithm adds $\mathcal{P}_i$ to the set of good parameters $\mathcal{G}$ (Step 7). The algorithm estimates the $\delta/4$-capped expected loss of the parameters contained $\mathcal{P}_i$, and if this estimate is smaller than the current upper confidence bound T on $OPT_{\delta/4}$, it updates T accordingly (Steps 8 through 10). Once the cap 2^t has grown sufficiently large compared to the upper confidence bound T, the algorithm returns an arbitrary parmeter from each set in $\mathcal{G}$.

Algorithm analysis. We now provide guarantees on Algorithm 1's performance. We denote the values of t and T at termination by $\bar{t}$ and $\bar{T}$, and we denote the state of the set $\mathcal{G}$ at termination by $\bar{\mathcal{G}}$. For each set $\mathcal{P}' \in \bar{\mathcal{G}}$, we use the notation $\tau_{\mathcal{P}'}$ to denote the value $\tau_{\lfloor|\mathcal{S}_t|(1-3\delta/8)\rfloor}$ in Step 9 during the iteration t that $\mathcal{P}'$ is added to $\mathcal{G}$.

Theorem 4.1. *Let* $\eta = \min\left\{\left(\sqrt[4]{1+\epsilon} - 1\right)/8, 1/9\right\}$ *and* $c = 16\sqrt[4]{1+\epsilon}/\delta$. *With probability* $1 - \zeta$, *the following conditions hold:*

1. *Algorithm 1 terminates after* $\bar{t} = O\left(\log\left(c \cdot OPT_{\delta/4}\right)\right)$ *iterations.*

2. *Algorithm 1 returns an* (ϵ, δ)-*optimal set of parameters of size at most* $\sum_{t=1}^{\bar{t}} f\left(\mathcal{S}_t, c \cdot OPT_{\delta/4}\right)$.

3. *The sample complexity on round* $t \in [\bar{t}]$ *is*

$$|\mathcal{S}_t| = \tilde{O}\left(\frac{d \ln f\left(\mathcal{S}_t, c \cdot OPT_{\delta/4}\right) + c \cdot OPT_{\delta/4}}{\eta^2 \delta^2}\right).$$

Proof. We split this proof into separate lemmas. Lemma 4.3 proves Part 1. Lemmas 4.5 and B.11 prove Part 2. Finally, Part 3 follows from classic results in learning theory on empirical Rademacher complexity. In particular, it follows from an inversion of the inequality in Step 4, the monotonicity of f (Definition 4.1), and the fact that $2^t \leq 2^{\bar{t}} \leq c \cdot OPT_{\delta/4}$, as we prove in Lemma 4.3. $\square$

Theorem 4.1 hinges on the assumption that the samples $\mathcal{S}_1, \ldots, \mathcal{S}_{\bar{t}}$ Algorithm 1 draws in Step 4 are sufficiently representative of the distribution Γ, formalized as follows:

Definition 4.2 (ζ-representative run)**.** For each round $t \in [\bar{t}]$, denote the samples in $\mathcal{S}_t$ as $\mathcal{S}_t = \left\{j_i^{(t)} : i \in [|\mathcal{S}_t|]\right\}$. We say that Algorithm 1 has a *ζ-representative run* if for all rounds $t \in [\bar{t}]$, all integers $b \in [|\mathcal{S}_t|]$, all caps $\tau \in \mathbb{Z}_{\geq 0}$, and all parameters $\boldsymbol{\rho} \in \mathcal{P}$, the following conditions hold:

8

1. The average number of instances $j_i^{(t)} \in \left\{ j_1^{(t)}, \ldots, j_b^{(t)} \right\}$ with loss smaller than τ nearly matches the probability that $\ell(\boldsymbol{\rho}, j) \leq \tau$: $\left| \frac{1}{b} \sum_{i=1}^{b} \mathbf{1}_{\left\{ \ell\left(\boldsymbol{\rho}, j_i^{(t)}\right) \leq \tau \right\}} - \Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \leq \tau] \right| \leq \gamma(t, b, \tau)$, and

2. The average τ-capped loss of the instances $j_1^{(t)}, \ldots, j_b^{(t)}$ nearly matches the expected τ-capped loss: $\left| \frac{1}{b} \sum_{i=1}^{b} \min \left\{ \ell\left(\boldsymbol{\rho}, j_i^{(t)}\right), \tau \right\} - \mathbb{E}_{j \sim \Gamma} [\min \{\ell(\boldsymbol{\rho}, j), \tau\}] \right| \leq \tau \cdot \gamma(t, b, \tau)$,

where $\gamma(t, b, \tau) = \sqrt{\frac{2d \ln f\left(\left\{ j_1^{(t)}, \ldots, j_b^{(t)} \right\}, \tau \right)}{b}} + 2\sqrt{\frac{2}{b} \ln \frac{8(\tau b t)^2}{\zeta}}$.

In Step 4, we ensure $\mathcal{S}_t$ is large enough that Algorithm 1 has a ζ-representative run with probability $1 - \zeta$.

Lemma 4.2. *With probability $1 - \zeta$, Algorithm 1 has a ζ-representative run.*

Lemma 4.2 is a corollary of a Rademacher complexity analysis in Appendix B (Lemma B.1). Intuitively, since the configuration problem is f-piecewise, there are only $f(\mathcal{S}, \tau)$ algorithms with varying τ-capped losses over any set of samples $\mathcal{S}$. We can therefore invoke Massart's finite lemma [Massart, 2000], which guarantees that each set $\mathcal{S}_t$ is sufficiently large to ensure that Algorithm 1 indeed has a ζ-representative run. The remainder of our analysis will assume that Algorithm 1 has a ζ-representative run.

Number of iterations until Algorithm 1 terminates. We begin with a proof sketch of the first part of Theorem 4.1. The full proof is in Appendix B.

Lemma 4.3. *Suppose Algorithm 1 has a ζ-representative run. Then $2^{\bar{t}} \leq \frac{16}{\delta} \sqrt[4]{1 + \epsilon} \cdot OPT_{\delta/4}$.*

Proof sketch. By definition of $OPT_{\delta/4}$, for every $\gamma > 0$, there exists a vector $\boldsymbol{\rho}^* \in \mathcal{P}$ whose $\delta/4$-capped expected loss is within a γ-factor of optimal: $\mathbb{E}_{j \sim \Gamma} [\min \{\ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*)\}] \leq OPT_{\delta/4} + \gamma$. We prove this vector's $\delta/4$-capped expected loss bounds $\bar{t}$:

$$2^{\bar{t}} \leq \frac{16 \sqrt[4]{1 + \epsilon}}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right]. \tag{2}$$

This implies the lemma statement holds. We split the proof of Inequality (2) into two cases: one where the vector $\boldsymbol{\rho}^*$ is contained within a set $\mathcal{P}' \in \bar{\mathcal{G}}$, and the other where it is not. In the latter case, Lemma 4.4 bounds $2^{\bar{t}}$ by $\frac{8}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right]$, which implies Inequality (2) holds. We leave the other case to the appendix. $\square$

In the next lemma, we prove the upper bound on $2^{\bar{t}}$ that we use in Lemma 4.3. The full proof is in Appendix B.

Lemma 4.4. *Suppose Algorithm 1 has a ζ-representative run. For any parameter vector $\boldsymbol{\rho} \notin \bigcup_{\mathcal{P}' \in \bar{\mathcal{G}}} \mathcal{P}'$, $2^{\bar{t}} \leq \frac{8}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right]$.*

Proof sketch. The last round that Algorithm 1 adds any subset to the set $\mathcal{G}$ is round $\bar{t} - 1$. For ease of notation, let $\bar{\mathcal{S}} = \mathcal{S}_{\bar{t}-1}$. Since $\boldsymbol{\rho}$ is not an element of any set in $\mathcal{G}$, the cap $2^{\bar{t}-1}$ must be too small compared to the average loss of the parameter $\boldsymbol{\rho}$. Specifically, it must be that $\frac{1}{|\bar{\mathcal{S}}|} \sum_{j \in \bar{\mathcal{S}}} \mathbf{1}_{\left\{ \ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1} \right\}} < 1 - \frac{3\delta}{8}$. Since Algorithm 1 had a ζ-representative run, the probability the loss of $\boldsymbol{\rho}$ is smaller than $2^{\bar{t}-1}$ converges to the fraction of samples with loss smaller than $2^{\bar{t}-1}$: $\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1} \right] \leq \frac{1}{|\bar{\mathcal{S}}|} \sum_{j \in \bar{\mathcal{S}}} \mathbf{1}_{\left\{ \ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1} \right\}} + \eta \delta$, so $\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1} \right] < 1 - (3/8 - \eta)\delta$. Since $\eta \leq 1/9$, it must be that $\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \geq 2^{\bar{t}-1} \right] \geq \delta/4$, so by definition of $t_{\delta/4}(\boldsymbol{\rho})$, we have that $2^{\bar{t}-1} \leq t_{\delta/4}(\boldsymbol{\rho})$. Therefore, $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right] \geq \frac{\delta}{4} \cdot t_{\delta/4}(\boldsymbol{\rho}) \geq 2^{\bar{t}-3} \delta$. $\square$

Optimality of Algorithm 1's output. Next, we provide a proof sketch of the second part of Theorem 4.1, which guarantees that Algorithm 1 returns an (ϵ, δ)-optimal subset. The full proof is in Appendix B. For each set $\mathcal{P}' \in \bar{\mathcal{G}}$, $\tau_{\mathcal{P}'}$ denotes the value of $\tau_{\lfloor |\mathcal{S}_t|(1-3\delta/8)\rfloor}$ in Step 9 of Algorithm 1 during the iteration t that $\mathcal{P}'$ is added to $\mathcal{G}$.

Lemma 4.5. *If Algorithm 1 has a ζ-representative run, it returns an (ϵ, δ)-optimal subset.*

Proof sketch. By definition of $OPT_{\delta/4}$, for every $\gamma > 0$, there is a vector $\boldsymbol{\rho}^* \in \mathcal{P}$ such that $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*)\right\}\right] \leq OPT_{\delta/4} + \gamma$. Let $\mathcal{P}^*$ be the output of Algorithm 1. We claim there exists a parameter $\boldsymbol{\rho}' \in \mathcal{P}^*$ such that

$$\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}', j\right), t_{\delta/2}\left(\boldsymbol{\rho}'\right)\right\}\right] \leq \sqrt{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*)\right\}\right], \tag{3}$$

which implies the lemma statement. There are two cases: either the vector $\boldsymbol{\rho}^*$ is contained within a set $\mathcal{P}' \in \bar{\mathcal{G}}$, or it is not. In this sketch, we analyze the latter case.

By Lemma 4.4, we know that

$$\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, j\right), t_{\delta/4}(\boldsymbol{\rho}^*)\right\}\right] \geq 2^{\bar{t}-3}\delta.$$

When Algorithm 1 terminates, $2^{\bar{t}-3}\delta$ is greater than the upper confidence bound $\bar{T}$, which means that $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*)\right\}\right] > \bar{T}$. We next derive a lower bound on $\bar{T}$: we prove that there exists a set $\mathcal{P}' \in \bar{\mathcal{G}}$ and parameter vector $\boldsymbol{\rho} \in \mathcal{P}'$ such that $\sqrt[4]{1 + \epsilon} \cdot \bar{T} \geq \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}, j\right), \tau_{\mathcal{P}'}\right\}\right]$ (Lemma B.7). Our upper bound on $\bar{T}$ implies that

$$\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}, j\right), \tau_{\mathcal{P}'}\right\}\right] \leq \sqrt[4]{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, j\right), t_{\delta/4}(\boldsymbol{\rho})\right\}\right].$$

Finally, we prove that there is a parameter $\boldsymbol{\rho}' \in \mathcal{P}' \cap \mathcal{P}^*$ whose $\delta/2$-capped expected loss is within a $\sqrt[4]{1 + \epsilon}$-factor of the expected loss $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}, j\right), \tau_{\mathcal{P}'}\right\}\right]$ (Lemma B.8). This follows from a proof that $\tau_{\mathcal{P}'}$ approximates $t_{\delta/4}(\boldsymbol{\rho}')$ and the fact that $\boldsymbol{\rho}$ and $\boldsymbol{\rho}'$ are both elements of $\mathcal{P}'$. Stringing these inequalities together, we prove that Equation (3) holds. $\square$

The second part of Theorem 4.1 also guarantees that the size of the set Algorithm 1 returns is bounded (see Lemma B.11 in Appendix B). Together, Lemmas 4.3, 4.5, and B.11 bound the number of iterations Algorithm 1 makes until it returns an (ϵ, δ)-optimal subset.

5 Comparison to prior research

We now provide comparisons to prior research on algorithm configuration with provable guarantees. Both comparisons revolve around branch-and-bound (B&B) configuration for integer programming (IP), overviewed in Section 2.1.

Uniformly sampling configurations. Prior research provides algorithms for finding nearly-optimal configurations from a finite set [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019]. If the parameter space is infinite and their algorithms optimize over a uniformly-sampled set of $\tilde{\Omega}(1/\gamma)$ configurations, then the output configuration will be within the top γ-quantile, with high probability. If the set of good parameters is small, however, the uniform sample might not include any of them. Algorithm configuration problems where the high-performing parameters lie within a small region do exist, as we illustrate in the following theorem.

Theorem 5.1 (Balcan et al. [2018a]). *For any $\frac{1}{3} < a < b < \frac{1}{2}$ and $n \geq 6$, there is a distribution Γ over IPs with n variables and a B&B parameter[4] with range $[0, 1]$ such that:*

1. *If $\rho \leq a$, then $\ell(\rho, j) = 2^{(n-5)/4}$ with probability $\frac{1}{2}$ and $\ell(\rho, j) = 8$ with probability $\frac{1}{2}$.*

2. *If $\rho \in (a, b)$, then $\ell(\rho, j) = 8$ with probability 1.*

3. *If $\rho \geq b$, then $\ell(\rho, j) = 2^{(n-4)/2}$ with probability $\frac{1}{2}$ and $\ell(\rho, j) = 8$ with probability $\frac{1}{2}$.*

Here, $\ell(\rho, j)$ measures the size of the tree B&B builds using the parameter ρ on the input integer program j.

In the above configuration problem, any parameter in the range (a, b) has a loss of 8 with probability 1, which is the minimum possible loss. Any parameter outside of this range has an abysmal expected loss of at least $2^{(n-6)/2}$. In fact, for any $\delta \leq 1/2$, the δ-capped expected loss of any parameter in the range $[0, a] \cup [b, 1]$ is at least $2^{(n-6)/2}$. Therefore, if we uniformly sample a finite set of parameters and optimize over this set using an algorithm for finite parameter spaces [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019], we must ensure that we sample at least one parameter within (a, b). As a and b converge, however, the required number of samples shoots to infinity, as we formalize below. This section's omitted proofs are in Appendix C.

Theorem 5.2. *For the B&B configuration problem in Theorem 5.1, with constant probability over the draw of $m = \lfloor 1/(b-a) \rfloor$ parameters $\rho_1, \ldots, \rho_m \sim Uniform[0, 1]$, $\{\rho_1, \ldots, \rho_m\} \cap (a, b) = \emptyset$.*

Meanwhile, Algorithm 1 quickly terminates, having found an optimal parameter, as we describe below.

Theorem 5.3. *Let $\eta = \min\left\{ \frac{1}{8}\left(\sqrt[4]{1+\epsilon} - 1\right), \frac{1}{9} \right\}$. For the B&B configuration problem in Theorem 5.1, Algorithm 1 terminates after $\tilde{O}(\log 1/\delta)$ iterations, having drawn $\tilde{O}((\delta\eta)^{-2})$ sample problem instances, and returns a set containing an optimal parameter in (a, b).*

Similarly, Balcan et al. [2018c] exemplify clustering configuration problems—which we overview in Section 2.1—where the optimal parameters lie within an arbitrarily small region, and any other parameter leads to significantly worse performance. As in Theorem 5.2, this means a uniform sampling of the parameters will fail to find optimal parameters.

Uniform convergence. Prior research has provided uniform convergence sample complexity bounds for algorithm configuration. These guarantees bound the number of samples sufficient to ensure that for any configuration, its average loss over the samples nearly matches its expected loss.

We prove that in the case of B&B configuration, Algorithm 1 may use far fewer samples to find a nearly optimal configuration than the best-known uniform convergence sample complexity bound. Balcan et al. [2018a] prove uniform convergence sample complexity guarantees for B&B configuration. They bound the number of samples sufficient to ensure that for any configuration in their infinite parameter space, the size of the search tree B&B builds on average over the samples generalizes to the expected size of the tree it builds. For integer programs over n variables, the best-known sample complexity bound guarantees that $(2^n/\epsilon')^2$ samples are sufficient to ensure that the average tree size B&B builds over the samples is within an additive ϵ' factor of the expected tree size [Balcan et al., 2018a]. Meanwhile, as we describe in Theorem 5.3, there are B&B configuration problems where our algorithm's sample complexity bound is significantly better: our algorithm finds an optimal parameter using only $\tilde{O}((\delta\eta)^{-2})$ samples.

---

[4]As we describe in Appendix C, ρ controls the variable selection policy. The theorem holds for any node selection policy.

6 Conclusion

We presented an algorithm that learns a finite set of promising parameters from an infinite parameter space. It can be used to determine the input to a configuration algorithm for finite parameter spaces, or as a tool for compiling an algorithm portfolio. We proved bounds on the number of iterations before our algorithm terminates, its sample complexity, and the size of its output. A strength of our approach is its modularity: it can determine the input to a configuration algorithm for finite parameter spaces without depending on specifics of that algorithm's implementation. There is an inevitable tradeoff, however, between modularity and computational efficiency. In future research, our approach can likely be folded into existing configuration algorithms for finite parameter spaces.

References

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

Daniel Alabi, Adam Tauman Kalai, Katrina Ligett, Cameron Musco, Christos Tzamos, and Ellen Vitercik. Learning to prune: Speeding up repeated computations. In *Proceedings of the Conference on Learning Theory (COLT)*, 2019.

Pranjal Awasthi, Maria-Florina Balcan, and Konstantin Voevodski. Local algorithms for interactive clustering. *The Journal of Machine Learning Research*, 18(1):75–109, 2017.

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

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

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

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

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. *arXiv preprint arXiv:1611.04535*, 2018c.

Maria-Florina Balcan, Travis Dick, and Manuel Lang. Learning to link. *arXiv preprint arXiv:1907.00533*, 2019a.

Maria-Florina Balcan, Travis Dick, and Wesley Pegden. Semi-bandit optimization in the dispersed setting. *CoRR*, abs/1904.09014, 2019b.

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

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2546–2554, 2011.

Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. No-regret bayesian optimization with unknown hyperparameters. *Journal of Machine Learning Research*, 20:1–24, 2019.

David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Transactions On Graphics (TOG)*, 28(3):77, 2009.

David Bommes, Henrik Zimmer, and Leif Kobbelt. Practical mixed-integer optimization for geometry processing. In *International Conference on Curves and Surfaces*, pages 193–206, 2010.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

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

Vincent Cohen-Addad and Varun Kanade. Online Optimization of Smoothed Piecewise Constant Functions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

Dan DeBlasio and John D Kececioglu. *Parameter Advising for Multiple Sequence Alignment*. Springer, 2018.

John P. Dickerson and Tuomas Sandholm. Throwing darts: Random sampling helps tree search when the number of short certificates is moderate. In *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 2013. Also in AAAI-13 Late-Breaking paper track.

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

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

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

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

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Bayesian optimization with censored response data. In *NIPS workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits (BayesOpt11)*, 2011a.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization (LION)*, pages 507–523, 2011b.

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

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

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

Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *arXiv preprint arXiv:1902.05454*, 2019.

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

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

Jeff T Linderoth and Martin WP Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

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

Mehreen Saeed, Onaiza Maqbool, Haroon Atique Babri, Syed Zahoor Hassan, and S Mansoor Sarwar. Software clustering techniques and the use of combined algorithm. In *Proceedings of the European Conference on Software Maintenance and Reengineering*, pages 301–306. IEEE, 2003.

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.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2951–2959, 2012.

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

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

James R White, Saket Navlakha, Niranjan Nagarajan, Mohammad-Reza Ghodsi, Carl Kingsford, and Mihai Pop. Alignment and clustering of phylogenetic markers-implications for microbial diversity studies. *BMC bioinformatics*, 11(1):152, 2010.

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

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

A   Review of Rademacher complexity

At a high level, Rademacher complexity measures the extent to which a class of functions fit random noise. Intuitively, this measures the richness of a function class because more complex classes should be able to fit random noise better than simple classes. Empirical Rademacher complexity can be measured on the set of samples and implies generalization guarantees that improve based on structure exhibited by the set of samples. Formally, let $\mathcal{F} \subseteq [0, H]^{\mathcal{X}}$ be an abstract function class mapping elements of a domain $\mathcal{X}$ to the interval $[0, H]$. Given a set $\mathcal{S} = \{x_1, \ldots, x_N\} \subseteq \mathcal{X}$, the *empirical Rademacher complexity* of $\mathcal{F}$ with respect to $\mathcal{S}$ is defined as

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) = \mathbb{E}_{\boldsymbol{\sigma}} \left[\sup_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \sigma_i \cdot f(x_i) \right],$$

where $\sigma_i \sim \text{Uniform}(\{-1, 1\})$. Let $\mathcal{D}$ be a distribution over $\mathcal{X}$. Classic results from learning theory [Koltchinskii, 2001, Bartlett and Mendelson, 2002] guarantee that with probability $1 - \delta$ over the draw $\mathcal{S} \sim \mathcal{D}^N$, for every function $f \in \mathcal{F}$,

$$\left| \frac{1}{N} \sum_{x \in \mathcal{S}} f(x) - \mathbb{E}_{x \sim \mathcal{D}}[f(x)] \right| \leq \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) + 2H \sqrt{\frac{2 \ln(4/\delta)}{N}}.$$

Massart [2000] proved the following Rademacher complexity bound for the case where the set $\{(f(x_1), \ldots, f(x_N)) : f \in \mathcal{F}\}$ is finite.

Lemma A.1. *[Massart [2000]] Let $\mathcal{F}|_{\mathcal{S}} = \{(f(x_1), \ldots, f(x_N)) : f \in \mathcal{F}\}$. If $|\mathcal{F}|_{\mathcal{S}}|$ is finite, then $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}) \leq \frac{r\sqrt{2 \log |\mathcal{F}|_{\mathcal{S}}|}}{N}$, where $r = \max_{\boldsymbol{a} \in \mathcal{F}|_{\mathcal{S}}} ||\boldsymbol{a}||_2$.*

B   Additional proofs and lemmas about Algorithm 1

The following Rademacher complexity bound guarantees that with probability $1 - \zeta$, Algorithm 1 has a ζ-representative run.

Lemma B.1. *For any $\tau \in \mathbb{Z}_{\geq 0}$, define the function classes $\mathcal{F}_{\tau} = \{j \mapsto \min\{\ell(\boldsymbol{\rho}, j), \tau\} \mid \boldsymbol{\rho} \in \mathcal{P}\}$ and $\mathcal{H}_{\tau} = \{j \mapsto \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq \tau\}} \mid \boldsymbol{\rho} \in \mathcal{P}\}$. Let $\mathcal{S} \subseteq \Pi$ be a set of problem instances. Then $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{F}_{\tau}) \leq \tau \sqrt{\frac{2d \ln f(\mathcal{S}, \tau)}{|\mathcal{S}|}}$ and $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{H}_{\tau}) \leq \sqrt{\frac{2d \ln f(\mathcal{S}, \tau)}{|\mathcal{S}|}}$.*

Proof. This lemma follows from Lemma A.1. □

Lemma B.2. *Suppose Algorithm 1 has a ζ-representative run. For any set $\mathcal{P}' \in \bar{\mathcal{G}}$ and all parameters $\boldsymbol{\rho} \in \mathcal{P}'$, $t_{\delta/2}(\boldsymbol{\rho}) \leq \tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$.*

Proof. We begin by proving that $t_{\delta/2}(\boldsymbol{\rho}) \leq \tau_{\mathcal{P}'}$.

Claim B.3. *Suppose Algorithm 1 has a ζ-representative run. For any set $\mathcal{P}' \in \mathcal{G}$ and all parameters $\boldsymbol{\rho} \in \mathcal{P}'$, $t_{\delta/2}(\boldsymbol{\rho}) \leq \tau_{\mathcal{P}'}$.*

Proof of Claim B.3. Let t be the round that $\mathcal{P}'$ is added to $\mathcal{G}$. We know that for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}'$,

$$\frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) \leq \tau_{\mathcal{P}'}\}} \geq \frac{\lfloor |\mathcal{S}_t|\,(1 - 3\delta/8) \rfloor}{|\mathcal{S}_t|} \geq 1 - \frac{3\delta}{8} - \frac{1}{|\mathcal{S}_t|}. \tag{4}$$

Since Algorithm 1 had a ζ-representative run, we know that

$$\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} \right] \geq \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) \leq \tau_{\mathcal{P}'}\}} - \sqrt{\frac{2d \ln f\,(\mathcal{S}_t, \tau_{\mathcal{P}'})}{|\mathcal{S}_t|}} - 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8\,(\tau_{\mathcal{P}'}\,|\mathcal{S}_t|\,t)^2}{\zeta}}$$

$$\geq \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) \leq \tau_{\mathcal{P}'}\}} - \sqrt{\frac{2d \ln f\,(\mathcal{S}_t, 2^t)}{|\mathcal{S}_t|}} - 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8\,(2^t\,|\mathcal{S}_t|\,t)^2}{\zeta}},$$

where the second inequality follows from the fact that $\tau_{\mathcal{P}'} \leq 2^t$ and f is monotone (Definition 4.1). Based on Step 4 of Algorithm 1, we know that

$$\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} \right] \geq \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) \leq \tau_{\mathcal{P}'}\}} - \eta\delta.$$

Moreover, by Equation (4), $\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} \right] \geq 1 - \frac{3\delta}{8} - \frac{1}{|\mathcal{S}_t|} - \eta\delta$. Based on Step 4 of Algorithm 1, we also know that $\frac{1}{\sqrt{|\mathcal{S}_t|}} \leq \eta\delta$. Therefore, $\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} \right] \geq 1 - \left(\frac{3}{8} + \eta^2 + \eta \right) \delta$. Finally, since $\eta \leq \frac{1}{9}$, we have that $\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} \right] > 1 - \delta/2$, which means that

$$\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) > \tau_{\mathcal{P}'} \right] = \Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \geq \tau_{\mathcal{P}'} + 1 \right] < \frac{\delta}{2}. \tag{5}$$

We claim that Equation (5) implies that $t_{\delta/2}(\boldsymbol{\rho}) \leq \tau_{\mathcal{P}'}$. For a contradiction, suppose $t_{\delta/2}(\boldsymbol{\rho}) > \tau_{\mathcal{P}'}$, or in other words, $t_{\delta/2}(\boldsymbol{\rho}) \geq \tau_{\mathcal{P}'} + 1$. Since $t_{\delta/2}(\boldsymbol{\rho}) = \operatorname{argmax}_{\tau \in \mathbb{Z}} \{ \Pr_{j \sim \Gamma}[\ell(\boldsymbol{\rho}, j) \geq \tau] \geq \delta/2 \}$, this would mean that $\delta/2 \leq \Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \geq t_{\delta/2}(\boldsymbol{\rho}) \right] \leq \Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \geq \tau_{\mathcal{P}'} + 1 \right] < \delta/2$, which is a contradiction. Therefore, the claim holds. $\qquad \square$

Next, we prove that $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$.

Claim B.4. *Suppose Algorithm 1 has a ζ-representative run. For any set $\mathcal{P}' \in \mathcal{G}$ and all parameters $\boldsymbol{\rho} \in \mathcal{P}'$, $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$.*

Proof of Claim B.4. Let t be the round that $\mathcal{P}'$ is added to $\mathcal{G}$. We know that for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}'$,

$$1 - \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) \leq \tau_{\mathcal{P}'} - 1\}} = 1 - \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) < \tau_{\mathcal{P}'}\}}$$

$$= \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \left(1 - \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) < \tau_{\mathcal{P}'}\}} \right)$$

$$= \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho},j) \geq \tau_{\mathcal{P}'}\}}$$

16

$$\geq \frac{|\mathcal{S}_t| - \lfloor |\mathcal{S}_t| (1 - 3\delta/8) \rfloor}{|\mathcal{S}_t|}$$

$$\geq \frac{3\delta}{8}.$$

Therefore, $\frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} - 1\}} \leq 1 - \frac{3\delta}{8}$. Since Algorithm 1 had a ζ-representative run, we know that

$$\Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} - 1]$$

$$\leq \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} - 1\}} + \sqrt{\frac{2d \ln f(\mathcal{S}_t, \tau_{\mathcal{P}'} - 1)}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8((\tau_{\mathcal{P}'} - 1)|\mathcal{S}_t| t)^2}{\zeta}}$$

$$\leq 1 - \frac{3\delta}{8} + \sqrt{\frac{2d \ln f(\mathcal{S}_t, \tau_{\mathcal{P}'} - 1)}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8((\tau_{\mathcal{P}'} - 1)|\mathcal{S}_t| t)^2}{\zeta}}$$

$$< 1 - \frac{3\delta}{8} + \sqrt{\frac{2d \ln f(\mathcal{S}_t, 2^t)}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8(2^t |\mathcal{S}_t| t)^2}{\zeta}}$$

because $\tau_{\mathcal{P}'} - 1 < \tau_{\mathcal{P}'} \leq 2^t$ and f is monotone (Definition 4.1). Based on Step 4 of Algorithm 1, $\Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} - 1] < 1 - 3\delta/8 + \eta\delta$. Since $\eta \leq 1/9$, we have that $\Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \leq \tau_{\mathcal{P}'} - 1] < 1 - \delta/4$. Therefore, $\Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \geq \tau_{\mathcal{P}'}] = \Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) > \tau_{\mathcal{P}'} - 1] > \delta/4$. Since

$$t_{\delta/4}(\boldsymbol{\rho}) = \operatorname{argmax}_{\tau \in \mathbb{Z}} \left\{ \Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \geq \tau] \geq \delta/4 \right\},$$

we have that $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$. $\qquad \square$

The lemma statement follows from Claims B.3 and B.4. $\qquad \square$

Corollary B.5. *Suppose Algorithm 1 has a ζ-representative run. For every set $\mathcal{P}' \in \bar{\mathcal{G}}$ and any parameter vector $\boldsymbol{\rho} \in \mathcal{P}'$, $\tau_{\mathcal{P}'} \delta/4 \leq \mathbb{E}_{j \sim \Gamma} [\min \{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\}]$.*

Proof. By Lemma B.2, we know that $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$, so $\Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \geq \tau_{\mathcal{P}'}] \geq \frac{\delta}{4}$. Therefore, $\mathbb{E}_{j \sim \Gamma} [\min \{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\}] \geq \tau_{\mathcal{P}'} \Pr_{j \sim \Gamma} [\ell(\boldsymbol{\rho}, j) \geq \tau_{\mathcal{P}'}] \geq \frac{\tau_{\mathcal{P}'} \delta}{4}$. $\qquad \square$

Lemma 4.4. *Suppose Algorithm 1 has a ζ-representative run. For any parameter vector $\boldsymbol{\rho} \notin \bigcup_{\mathcal{P}' \in \bar{\mathcal{G}}} \mathcal{P}'$, $2^{\bar{t}} \leq \frac{8}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} [\min \{\ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho})\}]$.*

Proof. The last round that Algorithm 1 adds any subset to the set $\mathcal{G}$ is round $\bar{t} - 1$. For ease of notation, let $\bar{\mathcal{S}} = \mathcal{S}_{\bar{t}-1}$. Since $\boldsymbol{\rho}$ is not an element of any set in $\mathcal{G}$, the cap $2^{\bar{t}-1}$ must be too small compared to the average loss of the parameter $\boldsymbol{\rho}$. Specifically, it must be that $\frac{1}{|\bar{\mathcal{S}}|} \sum_{j \in \bar{\mathcal{S}}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1}\}} < 1 - \frac{3\delta}{8}$. Otherwise, the algorithm would have added a parameter set containing $\boldsymbol{\rho}$ to the set $\mathcal{G}$ on round $\bar{t} - 1$ (Step 6). Since Algorithm 1 had a ζ-representative run, we know that the probability the loss of $\boldsymbol{\rho}$ is smaller than $2^{\bar{t}-1}$ converges to the fraction of samples with loss smaller than $2^{\bar{t}-1}$. Specifically,

$$\Pr_{j \sim \Gamma} \left[\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1} \right]$$

$$\leq \frac{1}{|\bar{\mathcal{S}}|} \sum_{j \in \bar{\mathcal{S}}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1}\}} + \sqrt{\frac{2d \ln f(\bar{\mathcal{S}}, 2^{\bar{t}-1})}{|\bar{\mathcal{S}}|}} + 2\sqrt{\frac{2}{|\bar{\mathcal{S}}|} \ln \frac{8(2^{\bar{t}-1} |\bar{\mathcal{S}}| (\bar{t}-1))^2}{\zeta}}$$

17

$$\leq \frac{1}{|\bar{\mathcal{S}}|} \sum_{j \in \bar{\mathcal{S}}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1}\}} + \eta\delta,$$

where the second inequality follows from Step 4 of Algorithm 1. Using our bound of $1 - 3\delta/8$ on the fraction of samples with loss smaller than $2^{\bar{t}-1}$, we have that $\Pr_{j \sim \Gamma}\left[\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1}\right] < 1 - (3/8 - \eta)\delta$. Since $\eta \leq 1/9$, it must be that $\Pr_{j \sim \Gamma}\left[\ell(\boldsymbol{\rho}, j) \leq 2^{\bar{t}-1}\right] < 1 - \delta/4$, or conversely, $\Pr_{j \sim \Gamma}\left[\ell(\boldsymbol{\rho}, j) \geq 2^{\bar{t}-1}\right] \geq \Pr_{j \sim \Gamma}\left[\ell(\boldsymbol{\rho}, j) > 2^{\bar{t}-1}\right] > \delta/4$. Since

$$t_{\delta/4}(\boldsymbol{\rho}) = \operatorname{argmax}_{\tau \in \mathbb{Z}} \left\{ \Pr_{j \sim \Gamma}[\ell(\boldsymbol{\rho}, j) \geq \tau] \geq \delta/4 \right\},$$

we have that $2^{\bar{t}-1} \leq t_{\delta/4}(\boldsymbol{\rho})$. Therefore,

$$\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq t_{\delta/4}(\boldsymbol{\rho}) \Pr_{j \sim \Gamma}\left[\ell(\boldsymbol{\rho}, j) \geq t_{\delta/4}(\boldsymbol{\rho})\right]$$

$$\geq \frac{\delta}{4} \cdot t_{\delta/4}(\boldsymbol{\rho})$$

$$\geq 2^{\bar{t}-3}\delta.$$

$\square$

Lemma B.6. *Suppose Algorithm 1 has a ζ-representative run. For any parameter vector $\boldsymbol{\rho} \notin \bigcup_{\mathcal{P}' \in \bar{\mathcal{G}}} \mathcal{P}'$, $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq \bar{T}$.*

Proof. From Lemma 4.4, we know that $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq 2^{\bar{t}-3}\delta$. Moreover, from Step 2 of Algorithm 1, $\bar{T} \leq 2^{\bar{t}-3}\delta$, so $\mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq \bar{T}$. $\square$

Lemma B.7. *Suppose Algorithm 1 has a ζ-representative run. There exists a set $\mathcal{P}' \in \bar{\mathcal{G}}$ and a parameter vector $\boldsymbol{\rho} \in \mathcal{P}'$ such that $\bar{T} \geq \frac{1}{\sqrt[4]{1+\epsilon}} \cdot \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}\right]$.*

Proof. By definition of the upper confidence bound $\bar{T}$ (Steps 8 through 10 of Algorithm 1), there is some round t, some set $\mathcal{P}' \in \bar{\mathcal{G}}$, and some parameter vector $\boldsymbol{\rho} \in \mathcal{P}'$ such that $\bar{T} = \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}$. Since Algorithm 1 had a ζ-representative run,

$$\bar{T} = \frac{1}{|\mathcal{S}_t|} \sum_{j \in \mathcal{S}_t} \min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}$$

$$\geq \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}\right] - \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d \ln f(\mathcal{S}_t, \tau_{\mathcal{P}'})}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8(\tau_{\mathcal{P}'}|\mathcal{S}_t|t)^2}{\zeta}}\right).$$

By Step 6, we know that at least a $(1 - 3\delta/8)$-fraction of the problem instances $j \in \mathcal{S}_t$ have a loss $\ell(\boldsymbol{\rho}, j)$ that is at most 2^t. Therefore, by definition of $\tau_{\mathcal{P}'} = \tau_{\lfloor|\mathcal{S}_t|(1-3\delta/8)\rfloor}$, it must be that $\tau_{\mathcal{P}'} \leq 2^t$. By the monotonicity of the function f (Definition 4.1), this means that $\bar{T} \geq \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}\right] - \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d \ln f(\mathcal{S}_t, 2^t)}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|} \ln \frac{8(2^t|\mathcal{S}_t|t)^2}{\zeta}}\right)$. Based on Step 4 of Algorithm 1,

$$\bar{T} \geq \mathbb{E}_{j \sim \Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}\right] - \tau_{\mathcal{P}'}\eta\delta.$$

From Corollary B.5, $\tau_{\mathcal{P}'}\delta/4 \leq \mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}\right]$, which means that

$$\bar{T} \geq (1 - 4\eta)\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau_{\mathcal{P}'}\right\}\right].$$

Finally, the lemma statement follows from the fact that

$$\eta = \min\left\{\frac{1}{8}\left(\sqrt[4]{1 + \epsilon} - 1\right), \frac{1}{9}\right\} \leq \frac{1}{4}\left(1 - \frac{1}{\sqrt[4]{1 + \epsilon}}\right).$$

$\square$

Lemma B.8. *Suppose Algorithm 1 has a ζ-representative run. For every set $\mathcal{P}' \in \bar{\mathcal{G}}$ and every pair of parameter vectors $\boldsymbol{\rho}_1, \boldsymbol{\rho}_2 \in \mathcal{P}'$, $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right] \leq \sqrt[4]{1 + \epsilon} \cdot \mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\}\right].$*

Proof. Let t be the round that the interval $\mathcal{P}'$ was added to $\mathcal{G}$. Since Algorithm 1 had a ζ-representative run,

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right]$$
$$\leq \frac{1}{|\mathcal{S}_t|}\sum_{j\in\mathcal{S}_t}\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\} + \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln f(\mathcal{S}_t, \tau_{\mathcal{P}'})}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8(\tau_{\mathcal{P}'}|\mathcal{S}_t|t)^2}{\zeta}}\right).$$

By definition of the set $\mathcal{P}'$, for all problem instances $j \in \mathcal{S}_t$, $\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\} = \min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\}.$ Therefore,

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right]$$
$$\leq \frac{1}{|\mathcal{S}_t|}\sum_{j\in\mathcal{S}_t}\min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\} + \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln f(\mathcal{S}_t, \tau_{\mathcal{P}'})}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8(\tau_{\mathcal{P}'}|\mathcal{S}_t|t)^2}{\zeta}}\right).$$

Again, since Algorithm 1 had a ζ-representative run,

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right]$$
$$\leq \mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\}\right] + 2\tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln f(\mathcal{S}_t, \tau_{\mathcal{P}'})}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8(\tau_{\mathcal{P}'}|\mathcal{S}_t|t)^2}{\zeta}}\right).$$

Since $\tau_{\mathcal{P}'} \leq 2^t$ and f is monotone (Definition 4.1),

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right]$$
$$\leq \mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\}\right] + 2\tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln f(\mathcal{S}_t, 2^t)}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8(2^t|\mathcal{S}_t|t)^2}{\zeta}}\right).$$

Based on Step 4 of Algorithm 1, this means that

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right] \leq \mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\}\right] + 2\tau_{\mathcal{P}'}\eta\delta.$$

By Corollary B.5, $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_1, j), \tau_{\mathcal{P}'}\right\}\right] \leq (1 + 8\eta)\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}_2, j), \tau_{\mathcal{P}'}\right\}\right].$ The lemma statement follows from the fact that $\eta \leq \left(\sqrt[4]{1 + \epsilon} - 1\right)/8.$ $\square$

Lemma 4.5. *If Algorithm 1 has a ζ-representative run, it returns an (ϵ, δ)-optimal subset.*

Proof. Since $OPT_{\delta/4} := \inf_{\boldsymbol{\rho} \in \mathcal{P}} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}, j\right), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right] \right\}$, we know that for any $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho} \in \mathcal{P}$ such that $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right] \leq OPT_{\delta/4} + \gamma$. We claim there exists a parameter $\boldsymbol{\rho}' \in \mathcal{P}^*$ such that

$$\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}', j\right), t_{\delta/2}\left(\boldsymbol{\rho}'\right) \right\} \right] \leq \sqrt{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right], \tag{6}$$

and thus the lemma statement holds (see Lemma B.9).

First, suppose $\boldsymbol{\rho}$ is contained in a set $\mathcal{P}' \in \bar{\mathcal{G}}$. By Lemmas B.2 and B.8, there exists a parameter $\boldsymbol{\rho}' \in \mathcal{P}' \cap \mathcal{P}^*$ such that

$$\begin{aligned}
\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}', j\right), t_{\delta/2}\left(\boldsymbol{\rho}'\right) \right\} \right] &\leq \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}', j\right), \tau_{\mathcal{P}'} \right\} \right] \\
&\leq \sqrt[4]{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}, j\right), \tau_{\mathcal{P}'} \right\} \right] \\
&\leq \sqrt[4]{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}, j\right), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right].
\end{aligned}$$

Since $\sqrt[4]{1 + \epsilon} \leq \sqrt{1 + \epsilon}$, Equation (6) holds in this case.

Otherwise, suppose $\boldsymbol{\rho} \notin \bigcup_{\mathcal{P}' \in \bar{\mathcal{G}}} \mathcal{P}'$. By Lemma B.6, we know that $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right] \geq \bar{T}$. Moreover, by Lemma B.7, there exists a set $\mathcal{P}' \in \bar{\mathcal{G}}$ and parameter vector $\boldsymbol{\rho}^* \in \mathcal{P}'$ such that $\sqrt[4]{1 + \epsilon} \cdot \bar{T} \geq \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}^*, j\right), \tau_{\mathcal{P}'} \right\} \right]$. Finally, by Lemma B.8, there exists a parameter vector $\boldsymbol{\rho}' \in \mathcal{P}' \cap \mathcal{P}^*$ such that

$$\begin{aligned}
\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}', j\right), t_{\delta/2}\left(\boldsymbol{\rho}'\right) \right\} \right] &\leq \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}', j\right), \tau_{\mathcal{P}'} \right\} \right] \\
&\leq \sqrt[4]{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell\left(\boldsymbol{\rho}^*, j\right), \tau_{\mathcal{P}'} \right\} \right] \\
&\leq \sqrt{1 + \epsilon} \cdot \bar{T} \\
&< \sqrt{1 + \epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right].
\end{aligned}$$

Therefore, Equation (6) holds in this case as well. $\qquad\square$

Lemma B.9. *Let $\mathcal{P}^*$ be the set of parameters output by Algorithm 1. Suppose that for every $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho}' \in \mathcal{P}^*$ such that $\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}', j), t_{\delta/2}(\boldsymbol{\rho}') \right\} \right] \leq \sqrt{1 + \epsilon} \left(OPT_{\delta/4} + \gamma \right)$. Then $\min_{\boldsymbol{\rho} \in \mathcal{P}^*} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/2}(\boldsymbol{\rho}) \right\} \right] \right\} \leq \sqrt{1 + \epsilon} \cdot OPT_{\delta/4}$.*

Proof. For a contradiction, suppose that $\min_{\boldsymbol{\rho} \in \mathcal{P}^*} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/2}(\boldsymbol{\rho}) \right\} \right] \right\} > \sqrt{1 + \epsilon} \cdot OPT_{\delta/4}$ and let $\gamma' = \min_{\boldsymbol{\rho} \in \mathcal{P}^*} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/2}(\boldsymbol{\rho}) \right\} \right] \right\} - \sqrt{1 + \epsilon} \cdot OPT_{\delta/4}$. Setting $\gamma = \frac{\gamma'}{2\sqrt{1+\epsilon}}$, we know there exists a parameter vector $\boldsymbol{\rho}' \in \mathcal{P}^*$ such that

$$\begin{aligned}
\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}', j), t_{\delta/2}(\boldsymbol{\rho}') \right\} \right] &\leq \sqrt{1 + \epsilon} \left(OPT_{\delta/4} + \frac{\gamma'}{2\sqrt{1+\epsilon}} \right) \\
&= \sqrt{1 + \epsilon} \cdot OPT_{\delta/4} + \frac{\gamma'}{2} \\
&= \min_{\boldsymbol{\rho} \in \mathcal{P}^*} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/2}(\boldsymbol{\rho}) \right\} \right] \right\} - \frac{\gamma'}{2} \\
&< \min_{\boldsymbol{\rho} \in \mathcal{P}^*} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/2}(\boldsymbol{\rho}) \right\} \right] \right\},
\end{aligned}$$

which is a contradiction. $\qquad\square$

Lemma 4.3. *Suppose Algorithm 1 has a ζ-representative run. Then $2^{\bar{t}} \leq \frac{16}{\delta} \sqrt[4]{1 + \epsilon} \cdot OPT_{\delta/4}$.*

20

Proof. For each set $\mathcal{P}' \in \bar{\mathcal{G}}$, let $t_{\mathcal{P}'}$ be the round where $\mathcal{P}'$ is added to the set $\mathcal{G}$, let $\mathcal{S}_{\mathcal{P}'} = \mathcal{S}_{t_{\mathcal{P}'}}$, and let $\boldsymbol{\rho}_{\mathcal{P}'}$ be an arbitrary parameter vector in $\mathcal{P}'$. Since no set is added to $\mathcal{G}$ when $t = \bar{t}$, it must be that for all sets $\mathcal{P}' \in \bar{\mathcal{G}}$, $t_{\mathcal{P}'} \leq \bar{t} - 1$. Moreover, since $OPT_{\delta/4} := \inf_{\boldsymbol{\rho} \in \mathcal{P}} \left\{ \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}, j), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right] \right\}$, we know that for every $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho}^*$ such that

$$\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right] \leq OPT_{\delta/4} + \gamma.$$

Below, we prove that $2^{\bar{t}} \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right]$ and thus the lemma statement holds (see Lemma B.10).

Case 1: $\boldsymbol{\rho}^* \notin \bigcup_{\mathcal{P}' \in \bar{\mathcal{G}}} \mathcal{P}'$. By Lemma 4.4, we know that $2^{\bar{t}-3}\delta \leq \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right]$. Therefore, $2^{\bar{t}} \leq \frac{8}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right] \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right]$.

Case 2: $\boldsymbol{\rho}^*$ is an element of a set $\mathcal{P}' \in \bar{\mathcal{G}}$ and $t_{\mathcal{P}'} \leq \bar{t} - 2$. Let T' be the value of T at the beginning of round $\bar{t} - 1$. Since the algorithm does not terminate on round $\bar{t} - 1$, it must be that $2^{\bar{t}-4}\delta < T'$. By definition of T',

$$2^{\bar{t}-4}\delta < T' = \min_{\bar{\mathcal{P}}: t_{\bar{\mathcal{P}}} \leq \bar{t}-2} \frac{1}{|\mathcal{S}_{\bar{\mathcal{P}}}|} \sum_{j \in \mathcal{S}_{\bar{\mathcal{P}}}} \min \left\{ \ell(\boldsymbol{\rho}_{\bar{\mathcal{P}}}, j), \tau_{\bar{\mathcal{P}}} \right\} \leq \frac{1}{|\mathcal{S}_{\mathcal{P}'}|} \sum_{j \in \mathcal{S}_{\mathcal{P}'}} \min \left\{ \ell(\boldsymbol{\rho}^*, j), \tau_{\mathcal{P}'} \right\}.$$

Since Algorithm 1 had a ζ-representative run, $2^{\bar{t}-4}\delta$ is upper-bounded by

$$\mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), \tau_{\mathcal{P}'} \right\} \right] + \tau_{\mathcal{P}'} \left(\sqrt{\frac{2d \ln f(\mathcal{S}_{\mathcal{P}'}, \tau_{\mathcal{P}'})}{|\mathcal{S}_{\mathcal{P}'}|}} + 2\sqrt{\frac{2}{|\mathcal{S}_{\mathcal{P}'}|} \ln \frac{8(\tau_{\mathcal{P}'}|\mathcal{S}_{\mathcal{P}'}|t_{\mathcal{P}'})^2}{\zeta}} \right).$$

Since $\tau_{\mathcal{P}'} \leq 2^{t_{\mathcal{P}'}}$ and f is monotone, $2^{\bar{t}-4}\delta$ is at most

$$E_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), \tau_{\mathcal{P}'} \right\} \right] + \tau_{\mathcal{P}'} \left(\sqrt{\frac{2d \ln f(\mathcal{S}_{\mathcal{P}'}, 2^{t_{\mathcal{P}'}})}{|\mathcal{S}_{\mathcal{P}'}|}} + 2\sqrt{\frac{2}{|\mathcal{S}_{\mathcal{P}'}|} \ln \frac{8(2^{t_{\mathcal{P}'}}|\mathcal{S}_{\mathcal{P}'}|t_{\mathcal{P}'})^2}{\zeta}} \right).$$

By Step 4 of Algorithm 1, $2^{\bar{t}-4}\delta \leq \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), \tau_{\mathcal{P}'} \right\} \right] + \tau_{\mathcal{P}'} \eta \delta$. Finally, by Corollary B.5, $2^{\bar{t}-4}\delta \leq (1 + 4\eta) \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), \tau_{\mathcal{P}'} \right\} \right]$. Since $\eta < \left(\sqrt[4]{1+\epsilon} - 1 \right)/4$,

$$2^{\bar{t}-4}\delta < \sqrt[4]{1+\epsilon} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), \tau_{\mathcal{P}'} \right\} \right].$$

Recalling that $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho}^*)$ by Lemma B.2, we conclude that

$$2^{\bar{t}} \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta} \cdot \mathbb{E}_{j \sim \Gamma} \left[\min \left\{ \ell(\boldsymbol{\rho}^*, j), t_{\delta/4}(\boldsymbol{\rho}^*) \right\} \right].$$

Case 3: $\boldsymbol{\rho}^*$ is not an element of any set $\bar{\mathcal{P}} \in \bar{\mathcal{G}}$ with $t_{\bar{\mathcal{P}}} \leq \bar{t} - 2$, but $\boldsymbol{\rho}^*$ is an element of a set $\mathcal{P}' \in \bar{\mathcal{G}}$ with $t_{\mathcal{P}'} = \bar{t} - 1$. Let $\mathcal{S}' = \mathcal{S}_{\bar{t}-2}$ and let $\bar{\mathcal{P}}$ be the set containing $\boldsymbol{\rho}^*$ in Step 5 on round $\bar{t}-2$. Since $\bar{\mathcal{P}}$ was not added to $\mathcal{G}$ on round $\bar{t}-2$, we know that fewer than a $\left(1 - \frac{3\delta}{8}\right)$-fraction of the instances in $\mathcal{S}'$ have a loss of at most $2^{\bar{t}-2}$ when run with any parameter vector $\boldsymbol{\rho} \in \bar{\mathcal{P}}$ (including $\boldsymbol{\rho}^*$). In other words,

$$\frac{1}{|\mathcal{S}'|} \sum_{j \in \mathcal{S}'} \mathbf{1}_{\left\{ \ell(\boldsymbol{\rho}^*, j) \leq 2^{\bar{t}-2} \right\}} < 1 - \frac{3\delta}{8}.$$

Since Algorithm 1 had a ζ-representative run,

$$\Pr_{j\sim\Gamma}\left[\ell\left(\boldsymbol{\rho}^*,j\right)\leq 2^{\bar{t}-2}\right]$$

$$\leq \frac{1}{|\mathcal{S}'|}\sum_{j\in\mathcal{S}'}\mathbf{1}_{\left\{\ell(\boldsymbol{\rho}^*,j)\leq 2^{\bar{t}-2}\right\}}+\sqrt{\frac{2d\ln f\left(\mathcal{S}',2^{\bar{t}-2}\right)}{|\mathcal{S}'|}}+2\sqrt{\frac{2}{|\mathcal{S}'|}\ln\frac{8\left(2^{\bar{t}-2}|\mathcal{S}'|\left(\bar{t}-2\right)\right)^2}{\zeta}}$$

$$< 1-\frac{3\delta}{8}+\sqrt{\frac{2d\ln f\left(\mathcal{S}',2^{\bar{t}-2}\right)}{|\mathcal{S}'|}}+2\sqrt{\frac{2}{|\mathcal{S}'|}\ln\frac{8\left(2^{\bar{t}-2}|\mathcal{S}'|\left(\bar{t}-2\right)\right)^2}{\zeta}}.$$

Based on Step 4 of Algorithm 1, $\Pr_{j\sim\Gamma}\left[\ell\left(\boldsymbol{\rho}^*,j\right)\leq 2^{\bar{t}-2}\right] < 1-(3/8-\eta)\delta$. Since $\eta\leq 1/9$, $\Pr_{j\sim\Gamma}\left[\ell\left(\boldsymbol{\rho}^*,j\right)\leq 2^{\bar{t}-2}\right] < 1-\delta/4$. Therefore, $\Pr_{j\sim\Gamma}\left[\ell\left(\boldsymbol{\rho}^*,j\right)\geq 2^{\bar{t}-2}\right]\geq \Pr_{j\sim\Gamma}\left[\ell\left(\boldsymbol{\rho}^*,j\right) > 2^{\bar{t}-2}\right]\geq \delta/4$. Since $t_{\delta/4}(\boldsymbol{\rho}^*)=\operatorname{argmax}_{\tau\in\mathbb{Z}}\left\{\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}^*,j)\geq\tau]\geq\delta/4\right\}$, we have that $2^{\bar{t}-2}\leq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)$. Therefore,

$$\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,j\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]\geq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\Pr\left[\ell\left(\boldsymbol{\rho}^*,j\right)\geq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right]$$

$$\geq 2^{\bar{t}-2}\Pr\left[\ell\left(\boldsymbol{\rho}^*,j\right)\geq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right]$$

$$\geq \delta 2^{\bar{t}-4},$$

which means that $2^{\bar{t}}\leq\frac{16}{\delta}\cdot\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,j\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]$. $\qquad\square$

Lemma B.10. *Suppose that for all $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho}^*\in\mathcal{P}$ such that $2^{\bar{t}}\leq\frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,j\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]\leq\frac{16\sqrt[4]{1+\epsilon}}{\delta}\left(OPT_{\delta/4}+\gamma\right)$. Then $2^{\bar{t}}\leq\frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot OPT_{\delta/4}$.*

Proof. For a contradiction, suppose $2^{\bar{t}} > \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot OPT_{\delta/4}$ and let $\gamma'=2^{\bar{t}}-\frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot OPT_{\delta/4}$. Letting $\gamma=\frac{\gamma'\delta}{32\sqrt[4]{1+\epsilon}}$, we know there exists a parameter vector $\boldsymbol{\rho}^*\in\mathcal{P}$ such that

$$2^{\bar{t}}\leq\frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,j\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]$$

$$\leq\frac{16\sqrt[4]{1+\epsilon}}{\delta}\left(OPT_{\delta/4}+\frac{\gamma'\delta}{32\sqrt[4]{1+\epsilon}}\right)$$

$$=\frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot OPT_{\delta/4}+\frac{\gamma'}{2}$$

$$=2^{\bar{t}}-\frac{\gamma'}{2}$$

$$<2^{\bar{t}},$$

which is a contradiction. Therefore, the lemma statement holds. $\qquad\square$

Lemma B.11. *Suppose Algorithm 1 has a ζ-representative run. The size of the set $\mathcal{P}^*\subset\mathcal{P}$ that Algorithm 1 returns is bounded by $\sum_{t=1}^{\bar{t}}f\left(\mathcal{S}_t,\frac{16}{\delta}\sqrt[4]{1+\epsilon}\cdot OPT_{\delta/4}\right)$.*

Proof. Based on Step 12 of Algorithm 1, the size of $\mathcal{P}^*$ equals the size of the set $\bar{\mathcal{G}}$. Algorithm 1 only adds sets to $\mathcal{G}$ on Step 7, and on each round t, the number of sets it adds is bounded by $f\left(\mathcal{S}_t,2^t\right)$. Based on the monotonicity of the function f (Definition 4.1), we know that $f\left(\mathcal{S}_t,2^t\right)\leq f\left(\mathcal{S}_t,2^{\bar{t}}\right)\leq f\left(\mathcal{S}_t,\frac{16}{\delta}\sqrt[4]{1+\epsilon}\cdot OPT_{\delta/4}\right)$. Therefore, $|\bar{\mathcal{G}}|=|\mathcal{P}^*|\leq\sum_{t=1}^{\bar{t}}f\left(\mathcal{S}_t,\frac{16}{\delta}\sqrt[4]{1+\epsilon}\cdot OPT_{\delta/4}\right)$. $\qquad\square$

C   Additional information about related research

In this section, we begin by surveying additional related research. We then describe the bounds on the function $f(\mathcal{S}, \tau)$ that prior research has provided in the contexts of integer programming and clustering. We also describe an integer programming algorithm configuration problem where the best-known uniform convergence bound is exponential in the number of variables [Balcan et al., 2018a], whereas our algorithm only requires $\tilde{O}\left((\delta\eta)^{-2}\right)$ samples.

C.1   Survey of additional related research

A related line of research [Gupta and Roughgarden, 2017, Cohen-Addad and Kanade, 2017, Balcan et al., 2018b, 2019b, Alabi et al., 2019] studies algorithm configuration in online learning settings, where the learner encounters a sequence—perhaps adversarially selected—of problem instances over a series of timesteps. The learner's goal is to select an algorithm on each timestep so that the learner has strong cumulative performance across all timesteps (as quantified by *regret*, typically). In contrast, this paper is focused on the batch learning setting, where the problem instances are not adversarially generated but come from a fixed distribution.

A number of papers have explored Bayesian optimization as a tool for parameter optimization. These algorithms have strong performance in practice [Bergstra et al., 2011, Hutter et al., 2011a,b, Snoek et al., 2012], but often do not come with theoretical guarantees, which is the focus of our paper. Those papers that do include provable guarantees typically require that the loss function is smooth, as quantified by its Lipschitz constant [Brochu et al., 2010] or its RKHS norm [Berkenkamp et al., 2019], which is not the case in our setting.

C.2   Integer programming

In this section, to match prior research, we use the notation Q to denote an integer program (rather than the notation j, as in the main body).

Balcan et al. [2018a] study *mixed integer linear programs* (MILPs) where the goal is to maximize an objective function $\boldsymbol{c}^\top \boldsymbol{x}$ subject to the constraints that $A\boldsymbol{x} \leq \boldsymbol{b}$ and that some of the components of $\boldsymbol{x}$ are contained in $\{0, 1\}$. Given a MILP Q, we use the notation $\breve{\boldsymbol{x}}_Q = (\breve{x}_Q[1], \ldots \breve{x}_Q[n])$ to denote an optimal solution to the MILP's LP relaxation. We denote the optimal objective value to the MILP's LP relaxation as $\breve{c}_Q$, which means that $\breve{c}_Q = \boldsymbol{c}^\top \breve{\boldsymbol{x}}_Q$.

The most popular algorithm for solving MILPs is called *branch-and-bound (B&B)*, which we now describe at a high level. Let Q' be a MILP we want to solve. B&B builds a search tree $\mathcal{T}$ with Q' at the root. At each round, the algorithm uses a *node selection policy* (such as *depth- or best-first search*) to choose a leaf of $\mathcal{T}$. This leaf node corresponds to a MILP we denote as Q. Using a *variable selection policy*, the algorithm then chooses one of that MILP's variables. Specifically, let Q_i^+ (resp., Q_i^-) equal the MILP Q after adding the constraint $x_i = 1$ (resp., $x_i = 0$). The algorithm defines the right (resp., left) child of the leaf Q to equal Q_i^+ (resp., Q_i^-). B&B then tries to "fathom" these leafs. At a high level, B&B fathoms a leaf if it can guarantee that it will not find any better solution by branching on that leaf than the best solution found so far. See, for example, the research by Balcan et al. [2018a] for the formal protocol. Once B&B has fathomed every leaf, it terminates. It returns the best feasible solution to Q' that it found in the search tree, which is provably optimal.

Balcan et al. [2018a] focus on variable selection policies, and in particular, *score-based variable selection policies*, defined below.

Definition C.1 (Score-based variable selection policy [Balcan et al., 2018a])**.** Let `score` be a deterministic function that takes as input a partial search tree $\mathcal{T}$, a leaf Q of that tree, and an index i, and returns a real value $\texttt{score}(\mathcal{T}, Q, i) \in \mathbb{R}$. For a leaf Q of a tree $\mathcal{T}$, let $N_{\mathcal{T},Q}$ be the set of variables that have not yet been branched on along the path from the root of $\mathcal{T}$ to Q. A score-based variable selection policy selects the variable $\text{argmax}_{x_i \in N_{\mathcal{T},Q}}\{\texttt{score}(\mathcal{T}, Q, i)\}$ to branch on at the node Q.

Score-based variable selection policies are extremely popular in B&B implementations [Linderoth and Savelsbergh, 1999, Achterberg, 2009, Gilpin and Sandholm, 2011]. See the research by Balcan et al. [2018a] for examples.

C.2.1   Bounding the function f

Given d arbitrary scoring rules $\texttt{score}_1, \ldots, \texttt{score}_d$, Balcan et al. [2018a] provide guidance for learning a linear combination $\rho_1 \texttt{score}_1 + \cdots + \rho_d \texttt{score}_d$ that leads to small expected tree sizes. They assume that all aspects of the tree search algorithm except the variable selection policy, such as the node selection policy, are fixed. In their analysis, they prove the following lemma.

Lemma C.1. *[Balcan et al. [2018a]] Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules and let Q be an arbitrary MILP over n binary variables. Suppose we limit B&B to producing search trees of size τ. There is a set $\mathcal{H}$ of at most $n^{2(\tau+1)}$ hyperplanes such that for any connected component R of $[0,1]^d \setminus \mathcal{H}$, the search tree B&B builds using the scoring rule $\rho_1 \texttt{score}_1 + \cdots + \rho_d \texttt{score}_d$ is invariant across all $(\rho_1, \ldots, \rho_d) \in R$.*

Balcan et al. [2018a] observe that in practice, the number of hyperplanes is significantly smaller. Given a set $\mathcal{S}$ of MILP instances, there are $|\mathcal{S}|n^{2(\tau+1)}$ relevant hyperplanes $\mathcal{H}^*$. The number of connected components of the set $[0,1]^d \setminus \mathcal{H}^*$ is at most $\left(|\mathcal{S}|n^{2(\tau+1)} + 1\right)^d$ [Buck, 1943]. Therefore, in our context, $f(\mathcal{S}, \tau) \leq \left(|\mathcal{S}|n^{2(\tau+1)} + 1\right)^d$. When $d = 2$, Balcan et al. [2018a] provide guidance for finding the partition of $[0,1]$ into intervals I where the search tree B&B builds using the scoring rule $\rho \cdot \texttt{score}_1 + (1 - \rho) \cdot \texttt{score}_2$ is invariant across all $\rho \in I$. These intervals correspond to the output of the function GETPARTITION. An important direction for future research is extending the implementation to multi-dimensional parameter spaces.

Uniformly sampling configurations.

Theorem 5.2. *For the B&B configuration problem in Theorem 5.1, with constant probability over the draw of $m = \lfloor 1/(b-a) \rfloor$ parameters $\rho_1, \ldots, \rho_m \sim \text{Uniform}[0,1]$, $\{\rho_1, \ldots, \rho_m\} \cap (a, b) = \emptyset$.*

Proof. We know that the probability $\{\rho_1, \ldots, \rho_m\} \cap (a, b) = \emptyset$ is $(1 - (b-a))^m \geq (1 - (b-a))^{1/(b-a)} \geq \frac{1}{3}$ since $b - a \leq \frac{1}{6}$. $\qquad\square$

Uniform convergence versus Algorithm 1.   We now describe an integer programming algorithm configuration problem where the best-known uniform convergence bound is exponential in the number of variables [Balcan et al., 2018a], whereas our algorithm only requires $\tilde{O}\left((\delta\eta)^{-2}\right)$ samples. We use a family of MILP distributions introduced by Balcan et al. [2018a]:

Theorem C.2 (Balcan et al. [2018a])**.** *For any MILP Q, let*

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

and

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

Define $\ell(\rho, Q)$ to be the size of the tree B&B produces using the scoring rule $\rho\cdot\texttt{score}_1+(1-\rho)\cdot\texttt{score}_2$ given Q as input. For every a,b such that $\frac{1}{3} < a < b < \frac{1}{2}$ and for all even $n \geq 6$, there exists an infinite family of distributions $\mathcal{D}$ over MILP instances with n variables such that the following conditions hold:

1. *If $\rho \leq a$, then $\ell(\rho, Q) = 2^{(n-5)/4}$ with probability $\frac{1}{2}$ and $\ell(\rho, Q) = 8$ with probability $\frac{1}{2}$.*

2. *If $\rho \in (a, b)$, then $\ell(\rho, Q) = 8$ with probability 1.*

3. *If $\rho \geq b$, then $\ell(\rho, Q) = 2^{(n-4)/2}$ with probability $\frac{1}{2}$ and $\ell(\rho, Q) = 8$ with probability $\frac{1}{2}$.*

This holds no matter which node selection policy B&B uses.

As we describe in Section 5, the sample complexity bound Balcan et al. [2018a] provide implies that at least $\left(\frac{2^n}{\epsilon'}\right)^2$ samples are required to ensure that the average tree size branch-and-bound builds over the samples is within an additive ϵ' factor of the expected tree size. Even for $\epsilon' = 2^{n/2}$, this sample complexity bound is exponential in n.

Theorem 5.3. *Let $\eta = \min\left\{\frac{1}{8}\left(\sqrt[4]{1+\epsilon} - 1\right), \frac{1}{9}\right\}$. For the B&B configuration problem in Theorem 5.1, Algorithm 1 terminates after $\tilde{O}(\log 1/\delta)$ iterations, having drawn $\tilde{O}((\delta\eta)^{-2})$ sample problem instances, and returns a set containing an optimal parameter in (a, b).*

Proof. In the proof of Theorem C.2, Balcan et al. [2018a] show that for any distribution $\mathcal{D}$ in this family and any subset $\mathcal{S}$ from the support of $\mathcal{D}$, $f(\mathcal{S}, \tau) = 3$, and the partition GETPARTITION returns is $[0, a]$, (a, b), and $[b, 1]$. Therefore, for the first three iterations, our algorithm will use $\tilde{O}\left((\delta\eta)^{-2}\right)$ samples. At that point, $t = 3$, and the tree-size cap is 8. Algorithm 1 will discover that for all of the samples $Q \in \mathcal{S}_3$, when $\rho \in (a, b)$, $\ell(\rho, Q) = 8$. Therefore, it add (a, b) to $\mathcal{G}$ and it will set $T = 8$. It will continue drawing $\tilde{O}\left((\delta\eta)^{-2}\right)$ samples at each round until $2^{t-3}\delta \geq T = 8$, or in other words, until $t = O(\log(1/\delta))$. Therefore, the total number of samples it draws is $\tilde{O}\left((\delta\eta)^{-2}\right)$. The set it returns will contain a point $\rho \in (a, b)$, which is optimal. $\qquad\square$

C.3 Clustering

We begin with an overview of agglomerative clustering algorithms. A clustering instance (V, d) consists of a set V of n points and a distance metric $d : V \times V \to \mathbb{R}_{\geq 0}$ specifying all pairwise distances between these points. The goal is to partition the points into groups such that distances within each group are minimized and distances between each group are maximized. Typically, the quality of a clustering is measured by an objective function, such as the classic k-means, k-median, or k-center objectives. Unfortunately, it is NP-hard to determine the clustering that minimizes any of these objectives.

An agglomerative clustering algorithm is characterized by a merge function $\xi(A, B) \to \mathbb{R}_{\geq 0}$, which defines the distance between any two sets of points $A, B \subseteq V$. The algorithm builds a *cluster tree* $\mathcal{T}$, starting with n singleton leaf nodes, each of which contains one point from V. The algorithm iteratively merges the two sets with minimum distance until there is a single node remaining, consisting of the set V. The children of any node N in this tree correspond to the two sets of points that were merged to form N. Common choices for the merge function ξ include

$\min_{a \in A, b \in B} d(a,b)$ (single-linkage), $\frac{1}{|A| \cdot |B|} \sum_{a \in A, b \in B} d(a,b)$ (average-linkage) and $\max_{a \in A, b \in B} d(a,b)$ (complete-linkage). The linkage procedure is followed by a dynamic programming step, which returns the pruning of the tree that minimizes a fixed objective function, such as the k-means, k-median, or k-center objectives. If the linkage procedure is terminated early, we will be left with a forest, rather than a single tree. The dynamic programming procedure can easily be adjusted to find the best pruning of the forest (for example, by completing the hierarchy arbitrarily and enforcing that the dynamic programming algorithm only return clusters contained in the original forest).

Balcan et al. [2017] define three infinite families of merge functions. The families $\mathcal{A}_1$ and $\mathcal{A}_2$ consist of merge functions $\xi(A, B)$ that depend on the minimum and maximum of all pairwise distances between A and B. The second family, denoted by $\mathcal{A}_3$, depends on all pairwise distances between A and B. All classes are parameterized by a single value ρ.

$$\mathcal{A}_1 = \left\{ \xi_{1,\rho} : (A,B) \mapsto \left(\min_{u \in A, v \in B} (d(u,v))^\rho + \max_{u \in A, v \in B} (d(u,v))^\rho \right)^{1/\rho} \middle| \rho \in \mathbb{R} \cup \{\infty, -\infty\} \right\},$$

$$\mathcal{A}_2 = \left\{ \xi_{2,\rho} : (A,B) \mapsto \rho \min_{u \in A, v \in B} d(u,v) + (1 - \rho) \max_{u \in A, v \in B} d(u,v) \middle| \rho \in [0,1] \right\},$$

$$\mathcal{A}_3 = \left\{ \xi_{3,\rho} : (A,B) \mapsto \left(\frac{1}{|A||B|} \sum_{u \in A, v \in B} (d(u,v))^\rho \right)^{1/\rho} \middle| \rho \in \mathbb{R} \cup \{\infty, -\infty\} \right\}.$$

The classes $\mathcal{A}_1$ and $\mathcal{A}_2$ define spectra of merge functions ranging from single-linkage ($\xi_{1,-\infty}$ and $\xi_{2,1}$) to complete-linkage ($\xi_{1,\infty}$ and $\xi_{2,0}$). The class $\mathcal{A}_3$ includes average-, complete-, and single-linkage.

Lemma C.3. *Let (V, d) be an arbitrary clustering instance over n points. There is a partition of $\mathbb{R}$ into $k = O(n^8)$ intervals $I_1, \dots, I_k$ such that for any interval I_i and any two parameters $\rho, \rho' \in I_i$, the sequences of merges the agglomerative clustering algorithm makes using the merge functions $\xi_{1,\rho}$ and $\xi_{1,\rho'}$ are identical. The same holds for the set of merge functions $\mathcal{A}_2$.*

Given a set $\mathcal{S}$ of clustering instances, there are $O\left(|\mathcal{S}|n^8\right)$ relevant intervals. Therefore, in our context, $f(\mathcal{S}, \tau) = O(|\mathcal{S}|n^8)$. Balcan et al. [2017] provide guidance for finding these intervals, which correspond to the output of the function GETPARTITION. They also prove the following guarantee for the class $\mathcal{A}_3$.

Lemma C.4. *Let (V, d) be an arbitrary clustering instance over n points. There is a partition of $\mathbb{R}$ into $k = O\left(n^2 3^{2n}\right)$ intervals $I_1, \dots, I_k$ such that for any interval I_i and any two parameters $\rho, \rho' \in I_i$, the sequences of merges the agglomerative clustering algorithm makes using the merge functions $\xi_{3,\rho}$ and $\xi_{3,\rho'}$ are identical.*

Similarly, given a set $\mathcal{S}$ of clustering instances, there are $O\left(|\mathcal{S}|n^2 3^{2n}\right)$ relevant intervals. Therefore, in our context, $f(\mathcal{S}, \tau) = O\left(|\mathcal{S}|n^2 3^{2n}\right)$. Again, Balcan et al. [2017] provide guidance for finding these intervals.