Learning to Optimize Computational Resources: Frugal Training with Generalization Guarantees

Maria-Florina Balcan,[1] **Tuomas Sandholm,**[1,2,3,4] **Ellen Vitercik**[1]
[1]Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213
[2]Optimized Markets, Inc., Pittsburgh, PA 15213, USA
[3]Strategic Machine, Inc., Pittsburgh, PA 15213, USA
[4]Strategy Robot, Inc., Pittsburgh, PA 15213, USA
{ninamf, sandholm, vitercik}@cs.cmu.edu

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), computational biology (DeBlasio and Kececioglu 2018), and many other fields. This approach has also been used in industry for tens of billions of dollars of combinatorial auctions (Sandholm 2013).

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, Leyton-Brown, and Lucier 2017; Kleinberg et al. 2019; Weisz, György, and Szepesvári 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 distribu-

tion 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, Leyton-Brown, and Lucier 2017; Kleinberg et al. 2019; Weisz, György, and Szepesvári 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. In order to provide guarantees, 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, Zimmer, and Kobbelt 2009). Animation studios with thousands of meshes require carefully tuned solvers which, thus far, domain experts have handcrafted (Bommes, Zimmer, and Kobbelt 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, Raghavan, and Schütze 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 $\frac{1}{\epsilon}$.*

We prove that our sample complexity can be exponentially better than the best-known uniform convergence bound. Moreover, it can find strong configurations in scenarios where uniformly sampling configurations will fail.

2 Problem definition

The algorithm configuration model we adopt is a generalization of the model from prior research (Kleinberg, Leyton-Brown, and Lucier 2017; Kleinberg et al. 2019; Weisz, György, and Szepesvári 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.

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

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

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, György, and Szepesvári (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

[1] As we describe in Section 2, we compete with a slightly more nuanced benchmark than OPT, in line with prior research.

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, György, and Szepesvári (2018; 2019), building off of work by Kleinberg, Leyton-Brown, and Lucier (2017), 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, $t_\delta(\boldsymbol{\rho}) = \text{argmax}_{\tau \in \mathbb{Z}} \{\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}, j) \geq \tau] \geq \delta\}$. 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, György, and Szepesvári (2018).

Definition 2.1 ($(\epsilon, \delta, \mathcal{P})$-optimality)**.** A parameter vector $\hat{\boldsymbol{\rho}}$ is $(\epsilon, \delta, \mathcal{P})$-*optimal* if $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\hat{\boldsymbol{\rho}}, j\right), t_\delta\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \leq (1 + \epsilon) \inf_{\boldsymbol{\rho}\in\mathcal{P}}\left\{\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}, j\right), t_{\delta/2}(\boldsymbol{\rho})\right\}\right]\right\}$.

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 $OPT_{c\delta} := \inf_{\boldsymbol{\rho}\in\mathcal{P}}\{\mathbb{E}_{j\sim\Gamma}[\min\{\ell(\boldsymbol{\rho}, j), t_{c\delta}(\boldsymbol{\rho})\}]\}$. If an algorithm returns an $\left(\epsilon, \delta, \bar{\mathcal{P}}\right)$-optimal parameter from within a finite set $\bar{\mathcal{P}}$, we call it a *configuration algorithm for finite parameter spaces*. Weisz, György, and Szepesvári (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, Hoos, and Leyton-Brown 2009; Dickerson and Sandholm 2013; He, Daume III, and Eisner 2014; Balafrej, Bessiere, and Paparrizou 2015; Khalil et al. 2016; 2017; Kruber, Lübbecke, and Parmentier 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, Balcan, and Voevodski 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, Raghavan, and Schütze 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-

[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, György, and Szepesvári 2018).

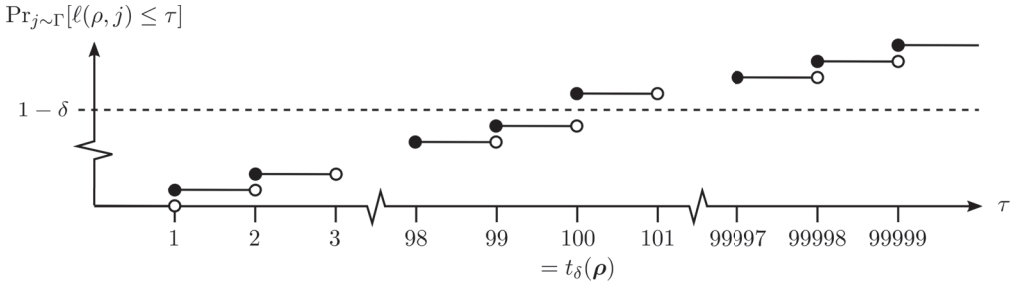

Figure 1: Fix a parameter vector $\boldsymbol{\rho}$. The figure is a hypothetical illustration of the cumulative density function of $\ell(\boldsymbol{\rho}, j)$ when j is sampled from Γ. For each value τ along the x-axis, the solid line equals $\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}, j) \le \tau]$. The dotted line equals the constant function $1-\delta$. Since 100 is the largest integer such that $\Pr_{j\sim\Gamma}[\ell(\boldsymbol{\rho}, j) \ge 100] \ge \delta$, we have that $t_\delta(\boldsymbol{\rho}) = 100$.

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, György, and Szepesvári 2019), 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 $\hat{\boldsymbol{\rho}} \in \bar{\mathcal{P}}$ such that $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\hat{\boldsymbol{\rho}}, j\right), t_{\delta/2}\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \le \sqrt{1+\epsilon} \cdot OPT_{\delta/4}$.

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 $\hat{\boldsymbol{\rho}} \in \bar{\mathcal{P}}$ is $\left(\epsilon', \delta, \bar{\mathcal{P}}\right)$-optimal. Then $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\hat{\boldsymbol{\rho}}, j\right), t_\delta\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \le (1+\epsilon) \cdot OPT_{\delta/4}$.*

Proof. Since the parameter $\hat{\boldsymbol{\rho}}$ is $\left(\epsilon', \delta, \bar{\mathcal{P}}\right)$-optimal, we know that $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\hat{\boldsymbol{\rho}}, j\right), t_\delta\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \le \sqrt{1+\epsilon} \cdot \min_{\boldsymbol{\rho}\in\bar{\mathcal{P}}}\left\{\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}, j\right), t_{\delta/2}(\boldsymbol{\rho})\right\}\right]\right\}$. (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 $\boldsymbol{\rho}' \in \bar{\mathcal{P}}$ such that $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}', j\right), t_{\delta/2}\left(\boldsymbol{\rho}'\right)\right\}\right] \le \sqrt{1+\epsilon} \cdot OPT_{\delta/4}$. Therefore,

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

so the theorem statement holds. □

4 Our main result: 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). More formally, this structure holds if for any problem instance $j \in \Pi$ and cap $\tau \in \mathbb{Z}_{\ge 0}$, there is a finite partition of the parameter space $\mathcal{P}$ such that in any one region R of this partition, for all pairs of parameter vectors $\boldsymbol{\rho}, \boldsymbol{\rho}' \in R$, $\min\{\ell(\boldsymbol{\rho}, j), \tau\} = \min\{\ell(\boldsymbol{\rho}', j), \tau\}$.

To exploit this piecewise-constant structure, we require access to a function PARTITION that takes as input a set $\mathcal{S}$ of problem instances and an integer τ and returns this partition of the parameters. Namely, it returns a set of tuples $(\mathcal{P}_1, z_1, \boldsymbol{\tau}_1), \ldots, (\mathcal{P}_k, z_k, \boldsymbol{\tau}_k) \in 2^{\mathcal{P}} \times [0,1] \times \mathbb{Z}^{|\mathcal{S}|}$ 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 $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathcal{P}_i$, $\frac{1}{|\mathcal{S}|}\sum_{j\in\mathcal{S}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, j)\le\tau\}} = \frac{1}{|\mathcal{S}|}\sum_{j\in\mathcal{S}} \mathbf{1}_{\{\ell(\boldsymbol{\rho}', j)\le\tau\}} = z_i$.
3. For all subsets $\mathcal{P}_i$, all $\boldsymbol{\rho}, \boldsymbol{\rho}' \in \mathcal{P}_i$, and all $j \in \mathcal{S}$, $\min\{\ell(\boldsymbol{\rho}, j), \tau\} = \min\{\ell(\boldsymbol{\rho}', j), \tau\} = \tau_i[j]$.

We assume the number of tuples PARTITION returns is monotone: if $\tau \le \tau'$, then $|\text{PARTITION}(\mathcal{S}, \tau)| \le |\text{PARTITION}(\mathcal{S}, \tau')|$ and if $\mathcal{S} \subseteq \mathcal{S}'$, then $|\text{PARTITION}(\mathcal{S}, \tau)| \le |\text{PARTITION}(\mathcal{S}', \tau)|$.

As we describe in the full version (Balcan, Sandholm, and Vitercik 2019), results from prior research imply guidance for implementing PARTITION 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, $|\text{PARTITION}(\mathcal{S}, \tau)| = O\left(|\mathcal{S}|n^8\right)$, though empirically, $|\text{PARTITION}(\mathcal{S}, \tau)|$ is often several orders of magnitude smaller (Balcan, Dick, and Lang 2019). Balcan et al. (2017) and Balcan, Dick, and Lang (2019) provide guidance for implementing PARTITION.

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** $\eta\delta$ is smaller than

$$\sqrt{\frac{2d \ln \left|\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)\right|}{|\mathcal{S}_t|}} + \sqrt{\frac{8}{|\mathcal{S}_t|} \ln \frac{8\left(2^t |\mathcal{S}_t| t\right)^2}{\zeta}}$$

do Draw $j \sim \Gamma$ and add j to $\mathcal{S}_t$.
5: Compute tuples $(\mathcal{P}_1, z_1, \boldsymbol{\tau}_1), \ldots, (\mathcal{P}_k, z_k, \boldsymbol{\tau}_k) \leftarrow \textsc{Partition}\left(\mathcal{S}_t, 2^t\right)$.
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 $\{\boldsymbol{\rho}_{\mathcal{P}'} \mid \mathcal{P}' \in \mathcal{G}\}$.

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).

Next, the algorithm evaluates the function $\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)$ to obtain the tuples $(\mathcal{P}_1, z_1, \boldsymbol{\tau}_1), \ldots, (\mathcal{P}_k, z_k, \boldsymbol{\tau}_k) \in 2^{\mathcal{P}} \times [0,1] \times \mathbb{Z}^{|\mathcal{S}_t|}$. 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. *With probability $1-\zeta$, the following conditions hold, with $\eta = \min\left\{\left(\sqrt[4]{1+\epsilon}-1\right)/8, 1/9\right\}$ and $c = 16\sqrt[4]{1+\epsilon}/\delta$:*

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 $\sum_{t=1}^{\bar{t}} \left|\textsc{Partition}\left(\mathcal{S}_t, c \cdot OPT_{\delta/4}\right)\right|$.*
3. *The sample complexity on round $t \in [\bar{t}]$, $|\mathcal{S}_t|$, is*

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

Proof. We split the proof into separate lemmas. Lemma 4.3 proves Part 1. Lemma 4.5 as well as Lemma B.11 in the full version (Balcan, Sandholm, and Vitercik 2019) prove Part 2. Finally, Part 3 follows from classic results in learning theory on Rademacher complexity. In particular, it follows from an inversion of the inequality in Step 4 and the fact that $2^t \leq 2^{\bar{t}} \leq c \cdot OPT_{\delta/4}$, as we prove in Lemma 4.3. □

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 4.1 (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:

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

$$\left|\frac{1}{b}\sum_{i=1}^{b} \min\left\{\ell\left(\boldsymbol{\rho}, j_i^{(t)}\right), \tau\right\} - \mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell(\boldsymbol{\rho}, j), \tau\right\}\right]\right|$$

$\leq \tau \cdot \gamma(t, b, \tau)$, where $\gamma(t, b, \tau)$ equals

$$\sqrt{\frac{2d \ln\left|\textsc{Partition}\left(\left\{j_1^{(t)}, \ldots, j_b^{(t)}\right\}, \tau\right)\right|}{b}} + 2\sqrt{\frac{2}{b}\ln\frac{8(\tau b t)^2}{\zeta}}.$$

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 which we include in the full version (Balcan, Sandholm, and Vitercik 2019). Intuitively, there are only $|\textsc{Partition}(\mathcal{S},\tau)|$ algorithms with varying τ-capped losses over any set of samples $\mathcal{S}$. We can therefore invoke Massart's finite lemma (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 termination. We begin with a proof sketch of the first part of Theorem 4.1. The full proof is in the full version (Balcan, Sandholm, and Vitercik 2019).

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: $\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,j\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right] \leq OPT_{\delta/4}+\gamma$. We prove this vector's $\delta/4$-capped expected loss bounds $\bar{t}$:

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

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 $\frac{8}{\delta}\cdot\mathbb{E}_{j\sim\Gamma}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,j\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]$, which implies Inequality (2) holds. We leave the other case to the full version (Balcan, Sandholm, and Vitercik 2019). □

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 the full version (Balcan, Sandholm, and Vitercik 2019).

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

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 the full version (Balcan, Sandholm, and Vitercik 2019). 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

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

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

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 the full version (Balcan, Sandholm, and Vitercik 2019)). Together, Lemmas 4.3 and 4.5, as well as Lemma B.11 in the full version (Balcan, Sandholm, and Vitercik 2019), 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, Leyton-Brown, and Lucier 2017; Kleinberg et al. 2019; Weisz, György, and Szepesvári 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 are infinitely-many distributions Γ over IPs with n variables and a B&B parameter*[3] *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, Leyton-Brown, and Lucier 2017; Kleinberg et al. 2019; Weisz, György, and Szepesvári 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 the full version (Balcan, Sandholm, and Vitercik 2019).

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. *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 (where $\eta = \min\left\{\frac{1}{8}\left(\sqrt[4]{1+\epsilon} - 1\right), \frac{1}{9}\right\}$), and returns a set containing an optimal parameter in (a, b).*

[3] As we describe in the full version (Balcan, Sandholm, and Vitercik 2019), ρ controls the variable selection policy. The theorem holds for any node selection policy.

Similarly, Balcan et al. (2018b) 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 $\left(2^n/\epsilon'\right)^2$ 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.

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.

Acknowledgments

This material is based on work supported by the National Science Foundation under grants IIS-1718457, IIS-1617590, IIS-1618714, IIS-1901403, CCF-1535967, CCF-1910321, SES-1919453, and CCF-1733556, the ARO under award W911NF-17-1-0082, an Amazon Research Award, an AWS Machine Learning Research Award, and a Bloomberg Data Science research grant.

References

Awasthi, P.; Balcan, M.-F.; and Voevodski, K. 2017. Local algorithms for interactive clustering. *The Journal of Machine Learning Research* 18(1):75–109.

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

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

Balcan, M.-F.; Dick, T.; Sandholm, T.; and Vitercik, E. 2018a. Learning to branch. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Balcan, M.-F.; Nagarajan, V.; Vitercik, E.; and White, C. 2018b. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. *arXiv preprint arXiv:1611.04535*.

Balcan, M.-F.; Dick, T.; and Lang, M. 2019. Learning to link. *arXiv preprint arXiv:1907.00533*.

Balcan, M.-F.; Sandholm, T.; and Vitercik, E. 2019. Learning to optimize computational resources: Frugal training with generalization guarantees. *arXiv preprint arXiv:1905.10819*.

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

Bommes, D.; Zimmer, H.; and Kobbelt, L. 2009. Mixed-integer quadrangulation. *ACM Transactions On Graphics (TOG)* 28(3):77.

Bommes, D.; Zimmer, H.; and Kobbelt, L. 2010. Practical mixed-integer optimization for geometry processing. In *International Conference on Curves and Surfaces*, 193–206.

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

Dickerson, J. P., and Sandholm, T. 2013. 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)*. Also in AAAI-13 Late-Breaking paper track.

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

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

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

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

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

Kleinberg, R.; Leyton-Brown, K.; Lucier, B.; and Graham, D. 2019. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *arXiv preprint arXiv:1902.05454*.

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

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

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

Manning, C.; Raghavan, P.; and Schütze, H. 2010. Introduction to information retrieval. *Natural Language Engineering* 16(1):100–103.

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

Saeed, M.; Maqbool, O.; Babri, H. A.; Hassan, S. Z.; and Sarwar, S. M. 2003. Software clustering techniques and the use of combined algorithm. In *Proceedings of the European Conference on Software Maintenance and Reengineering*, 301–306. IEEE.

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

Weisz, G.; György, A.; and Szepesvári, C. 2018. LEAPSANDBOUNDS: A method for approximately optimal algorithm configuration. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Weisz, G.; György, A.; and Szepesvári, C. 2019. CAPSANDRUNS: An improved method for approximately optimal algorithm configuration. *Proceedings of the International Conference on Machine Learning (ICML)*.

White, J. R.; Navlakha, S.; Nagarajan, N.; Ghodsi, M.-R.; Kingsford, C.; and Pop, M. 2010. Alignment and clustering of phylogenetic markers-implications for microbial diversity studies. *BMC bioinformatics* 11(1):152.

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

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32:565–606.