Automated algorithm and mechanism configuration

Ellen Vitercik

July 1, 2020

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
vitercik@cs.cmu.edu

Thesis Committee:
Maria-Florina Balcan (co-chair)
Tuomas Sandholm (co-chair)
Eric Horvitz
Kevin Leyton-Brown
Ameet Talwalkar

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Abstract

Algorithms from diverse domains often have tunable parameters that control performance metrics such as solution quality, runtime, and memory usage. Typically, the optimal setting depends intimately on the application domain at hand. Hand-tuning parameters is often tedious, time-consuming, and error-prone, so a burgeoning line of research has studied automated algorithm configuration via machine learning, where a training set of typical problem instances from the application at hand is used to find high-performing parameter settings.

In this thesis, we help develop the theory and practice of automated algorithm configuration. We investigate both statistical and algorithmic questions. For example, how large should the training set be to ensure that an algorithm's average performance over the training set is indicative of its future performance on unseen instances? How can we algorithmically find provably high-performing configurations? As we answer these questions, we analyze parameterized algorithms from a variety of domains, including:

1. **Integer programming.** We study branch-and-bound algorithms for integer linear programming, the most widely-used tool for solving combinatorial and nonconvex problems. Beyond answering the algorithmic and statistical questions above, we provide experiments demonstrating that no one parameter setting is optimal across all application domains, and that tuning parameters using our approach can have a significant impact on algorithmic performance. We also analyze integer quadratic programming approximation algorithms, which can be used to find nearly-optimal solutions to a variety of combinatorial problems.
2. **Mechanism design.** A mechanism is a special type of algorithm that plays a crucial role in economics and political science. A mechanism's purpose is to help a set of rational agents come to a collective decision. In economics, for example, a mechanism might dictate how a set of items should be split among the agents, given their values for those items. Mechanisms often have tunable parameters that impact, for example, their revenue. No one setting is optimal across all mechanism design scenarios. In this thesis, we analyze sales mechanisms, where the goal is to maximize revenue, and voting mechanisms, where the goal is to maximize the agents' total value for the outcome the mechanism selects.
3. **Computational biology.** Algorithms from computational biology are often highly parameterized, and understanding which parameter settings are optimal in which scenarios is an active area of research. We analyze parameterized algorithms for several fundamental problems from computational biology, including sequence alignment, RNA folding, and predicting topologically associating domains in DNA sequences.

The key challenge from both an algorithmic and statistical perspective is that across these three diverse domains, an algorithm's performance is an erratic function of its parameters. This is because a small tweak to the parameters can cause a cascade of changes in the algorithm's performance. We develop tools for analyzing and optimizing these volatile performance functions, which we use to provide parameter tuning procedures and strong statistical guarantees.

Contents

List of notation

d	Number of tunable algorithm parameters
$\boldsymbol{\rho}$	Parameter vector in $\mathbb{R}^d$
$\mathcal{P}$	Set of parameter vectors in $\mathbb{R}^d$
z	Problem instance
$\mathcal{Z}$	Set of problem instances
$\mathcal{D}$	Distribution over problem instances
$u_{\boldsymbol{\rho}} : \mathcal{Z} \to \mathbb{R}$	Utility of the algorithm parameterized by $\boldsymbol{\rho}$ as a function of the input problem instance
$\mathcal{U}$	Set of all utility functions $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\right\}$
$\mathcal{B}^{\mathcal{A}}$	Set of all functions mapping a set $\mathcal{A}$ to a set $\mathcal{B}$

Chapter 1

Introduction

Algorithms regularly come with tunable parameters that have a significant impact on the algorithm's solution quality and requisite computational resources. Optimal parameter settings typically depend closely on the specific application domain at hand. Domain experts often fine-tune algorithm parameters by hand, but during this time-consuming and tedious search, they may overlook many high-performing parameter settings.

In this proposal, we study automated algorithm configuration via machine learning, with broad applications to problems in integer programming, computational biology, and economics. This automated approach relieves domain experts of this error-prone yet essential task. The majority of this proposal applies to settings where the domain expert has a set of typical problem instances from her application domain, also known as a *training set*. A natural approach is to select a parameter setting with strong *algorithmic performance* (for example, solution quality or runtime) on average over the training set, and then apply that parameter setting to solve future, unseen problem instances not present in the training set.

The domain expert must be careful, however, when employing this technique: if her set of training instances is too small, parameters with strong performance on average over the training set may have poor future performance. This phenomenon, known as "overfitting," raises the question:

> *How many samples are sufficient to ensure that any parameter setting's average performance over the training set generalizes to its future performance?*

Formally, we assume that both the future problem instances and those in the training set are independently drawn from the same unknown, application-specific distribution. In Part I of this proposal, we thoroughly investigate this question of sample complexity, with the chapters organized as follows.

Chapter 3: Integer linear programming algorithms. We begin by analyzing branch-and-bound algorithms for solving integer linear programs. Branch-and-bound algorithms are used under the hood by popular commercial solvers such as CPLEX and Gurobi. These algorithms have many tunable parameters that can have an enormous effect on the size of the search tree the algorithm builds before it finds an optimal solution. Along with sample complexity guarantees, we present experiments demonstrating that no one parameter setting is optimal across all application domains, so our machine learning approach can significantly improve tree size.

Chapter 4: Integer quadratic programming algorithms. We next analyze approximation algorithms for finding nearly-optimal solutions to integer quadratic programs (IQPs). These algorithms can be used to find approximate solutions to, for example, the canonical max-cut and max 2SAT problems.

Chapter 5: Computation biology algorithms. We study algorithm configuration for a variety of computational biology algorithms. We begin by analyzing sequence alignment algorithms, which are used, for example, to uncover similarities between multiple DNA, RNA, or protein sequences. We also analyze RNA folding algorithms, which predict how an input RNA strand would naturally fold, offering insight into the RNA molecule's function. Finally, we study algorithms for predicting topologically associating domains in DNA sequences, which shed light on how the input DNA sequence wraps into three-dimensional structures that influence genome function.

Chapter 6: Mechanism design. In this chapter, we study a specific type of algorithm—called a *mechanism*—used to help rational agents come to collective decisions. For example, mechanisms can be used to help a seller decide how to split a set of items for sale among a set of buyers, and what those buyers should pay. We analyze mechanisms for selling items as well as mechanisms for voting. Mechanisms typically come with a variety of tunable parameters which affect, for example, the mechanism's revenue. We provide sample complexity bounds for tuning these mechanism parameters.

Chapter 7: A general theory of sample complexity for algorithm configuration. When analyzing the diverse algorithm families in Chapters 3-6, we notice a key structure that links all of them and is at the root of our sample complexity analyses: for any fixed problem instance, algorithmic performance is a piecewise-structured function of the algorithm's parameters (for example, it is piecewise-constant or -linear). In this chapter, we formalize this structure, and provide a general theorem that recovers our sample complexity guarantees from Chapters 3-6.

Chapter 8: Data-dependent guarantees. In this chapter, we provide data-dependent sample guarantees (as opposed to the results in the previous chapters, which hold for worst-case distributions over problem instances). We instantiate our guarantees in the context of integer linear programming algorithm configuration, the focus of Chapter 3.

Chapter 9: Estimating approximate incentive compatibility. We conclude Part I by providing additional tools that can be used in the context of economic mechanism design via machine learning (the focus of Chapter 6). Namely, we show how to estimate to what extent an agent is incentivized to behave strategically under a variety of mechanisms, and provide sample complexity guarantees.

In Part II, we move on to providing machine learning procedures for algorithm configuration. We investigate the fundamental question of *how* to find an algorithm configuration with provably strong performance.

Chapter 10: Batch learning algorithms. A recent line of research [109, 110, 185, 186] provides algorithms that return nearly-optimal parameter settings from within a finite set. These algorithms can be used when the parameter space is infinite by providing as input a random sample of parameter settings. This data-independent discretization, however, might miss pockets of nearly-optimal parameter settings. We provide an algorithm that learns a finite set of promising parameter settings 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.

Chapter 11: Beyond batch learning. In the final chapter of this proposal, we analyze an online algorithm design model. In this model, the algorithm encounters a sequence of computational problems that are similar but not necessarily drawn from a fixed distribution (unlike the previous chapters). Running a standard algorithm with worst-case runtime guarantees on each instance would fail to take advantage of valuable structure shared across the problem instances. For example, when a commuter drives from work to home, there are typically only a handful of routes that will ever be the shortest path. A naïve algorithm that does not exploit this common structure may spend most of its time checking roads that will never be in the shortest path. More generally, we can often ignore large swaths of the search space that will likely never contain an optimal solution. We present an algorithm that learns to prune the search space on repeated computations, thereby reducing runtime while provably outputting the correct solution each period with high probability.

1.1 Related research

Machine learning as a tool for algorithm configuration has been studied extensively in prior research, primarily from an applied perspective [98, 101, 104, 171]. In contrast, we are particularly focused on providing practical algorithm configuration tools with provable guarantees. Gupta and Roughgarden [89] introduced the batch learning approach of algorithm configuration to the theoretical computer science community, and we adopt their model throughout the majority of this proposal.

In a related theoretical direction, Sayag et al. [176] studied a model where there are multiple algorithms capable of computing a correct solution to a given computational problem, but with different costs. The user can run multiple algorithms until one terminates with the correct solution. Given a training set of problem instances, the authors show how to efficiently learn a schedule with nearly optimal expected cost. In other words, they provide theoretical guarantees for configuring the schedule using the same model that we adopt.

In several of the specific domains we study, there is existing research that analyzes automated parameter tuning in that particular area. This is the case in integer linear programming (Chapter 3), computational biology (Chapter 5), and mechanism design (Chapter 6). In those corresponding chapters, we highlight this related research.

The results in this proposal are related in spirit to the field of meta-learning and hyperparameter optimization in machine learning [e.g., 79, 125, 181]. That direction is focused on automating parameter tuning for highly-parameterized machine learning algorithms, such as deep learning models. Unlike the results in this proposal, the vast majority of this literature is empirical. Moreover, across the domains we study, algorithmic performance as a function of the parameters has sharp discontinuities; a small shift of the parameters can cause a cascade of changes in the algorithm's behavior. This behavior is unlike well-understood functions in machine learning theory, so we require new tools to provide strong guarantees.

Finally, a related line of research has designed algorithms that are aided by "machine learned advice" [99, 115, 135, 141, 160]. In essence, these algorithms use machine learning to make predictions about structural aspects of the input. If the prediction is accurate, the algorithm's performance (for example, its error or runtime) is superior to the best-known worst-case algorithm, and if the prediction is incorrect, the algorithm performs as well as that worst-case algorithm.

Part I

Sample complexity guarantees

Chapter 2

Notation and learning-theoretic tools

In Part I of this proposal, we provide sample complexity guarantees for algorithm configuration. Throughout the proposal, we will analyze algorithms parameterized by some set $\mathcal{P} \subseteq \mathbb{R}^d$ of vectors. We use the notation $\mathcal{Z}$ to denote the set of problem instances the algorithm may take as input. For example, $\mathcal{Z}$ might consists of integer programs, if we are configuring an integer programming solver. For every parameter vector $\boldsymbol{\rho} \in \mathcal{P}$, there is a *utility function* $u_{\boldsymbol{\rho}} : \mathcal{Z} \to \mathbb{R}$ that measures, abstractly, the performance of the algorithm parameterized by $\boldsymbol{\rho}$ given an input $z \in \mathcal{Z}$. For example, $u_{\boldsymbol{\rho}}$ might measure runtime, the quality of the algorithm's output, and so on. We use the notation $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\right\}$ to denote the set of all utility functions.

We assume there is an unknown distribution $\mathcal{D}$ over problem instances in $\mathcal{Z}$. For example, $\mathcal{D}$ might represent the integer programs an airline must solve on a day-to-day basis. The configuration procedure does not know $\mathcal{D}$, but it can sample from $\mathcal{D}$.

Throughout Part I, we bound the number of samples sufficient to learn a parameter vector with high expected utility over $\mathcal{D}$. A sample complexity guarantee for the function class $\mathcal{U}$ bounds the number of samples sufficient to ensure that for any function in $\mathcal{U}$, its expected value over an arbitrary distribution is close to its average value over a set of points sampled from that distribution. More formally, a sample complexity bound is a function $N_{\mathcal{U}} : \mathbb{R}_{>0} \times (0,1) \to \mathbb{N}$ together with a guarantee that for any $\delta \in (0,1)$ and $\epsilon > 0$, with probability $1-\delta$ over the draw of $N \geq N_{\mathcal{U}}(\epsilon, \delta)$ samples $z_1, \ldots, z_{N_0} \sim \mathcal{D}$, for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}$,

$$\left| \frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}\left(z_i\right) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}}\left[u_{\boldsymbol{\rho}}(z)\right] \right| \leq \epsilon. \tag{2.1}$$

This is also known as a *uniform convergence* bound because Equation (2.1) holds uniformly across all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}$.

If uniform convergence holds, it is well-known that the empirically optimal parameter vector is nearly optimal in expectation as well. Specifically, given a set of samples $\mathcal{S} \sim \mathcal{D}^N$, let $\hat{\rho} = \operatorname{argmax}_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{z \in \mathcal{S}} u_{\boldsymbol{\rho}}(z)$. With probability at least $1-\delta$ over the draw of $N \geq N_{\mathcal{U}}(\epsilon, \delta)$ samples, $\max_{\boldsymbol{\rho} \in \mathcal{P}} \mathbb{E}_{z \sim \mathcal{D}}\left[u_{\boldsymbol{\rho}}(z)\right] - \mathbb{E}_{z \sim \mathcal{D}}\left[u_{\hat{\rho}}(z)\right] \leq 2\epsilon$.

At a high level, the specific form of $N_{\mathcal{U}}(\epsilon, \delta)$ depends on the *intrinsic complexity* of the class $\mathcal{U}$. There are a variety of well-studied tools for quantifying a class's intrinsic complexity, including pseudo-dimension and Rademacher complexity, which we define below, adapting the notation from our setting. Generally speaking, these tools measure how well functions in $\mathcal{U}$ are able to fit complex patterns over many domain elements in $\mathcal{Z}$. Classic results from learning theory provide sample complexity bounds in terms of pseudo-dimension and Rademacher complexity.

2.1 Pseudo-dimension

Pseudo-dimension [159] is a well-studied learning-theoretic tool used to measure the complexity of a function class. To formally define pseudo-dimension, we first introduce the notion of

shattering, which is a fundamental concept in machine learning theory.

Definition 2.1.1 (Shattering)**.** Let $\mathcal{U}$ be a set of functions mapping $\mathcal{Z}$ to $\mathbb{R}$. Let $\mathcal{S} = \{z_1, \ldots, z_N\}$ be a subset of $\mathcal{Z}$ and let $t_1, \ldots, t_N \in \mathbb{R}$ be a set of *targets*. We say that $t_1, \ldots, t_N$ *witness* the shattering of $\mathcal{S}$ by $\mathcal{U}$ if for all subsets $T \subseteq \mathcal{S}$, there exists some parameter vector $\boldsymbol{\rho} \in \mathcal{P}$ such that for all elements $z_i \in T$, $u_{\boldsymbol{\rho}}(z_i) \leq t_i$ and for all $z_i \notin T$, $u_{\boldsymbol{\rho}}(z_i) > t_i$.

Definition 2.1.2 (Pseudo-dimension [159])**.** Let $\mathcal{U}$ be a set of functions mapping $\mathcal{Z}$ to $\mathbb{R}$. Let $\mathcal{S} \subseteq \mathcal{Z}$ be a largest set that can be shattered by $\mathcal{U}$. Then $\operatorname{Pdim}(\mathcal{U}) = |\mathcal{S}|$.

When $\mathcal{U}$ is a set of binary-valued functions mapping $\mathcal{Z}$ to $\{0, 1\}$, the pseudo-dimension of $\mathcal{U}$ is more commonly referred to as the *VC-dimension* of $\mathcal{U}$, which we denote as $\operatorname{VCdim}(\mathcal{U})$ [182].

Theorem 2.1.3 provides generalization bounds in terms of pseudo-dimension.

Theorem 2.1.3 ([159])**.** *Let $\mathcal{U}$ be a set of functions mapping $\mathcal{Z}$ to an interval $[-H, H]$ and let $d_{\mathcal{U}}$ be the pseudo-dimension of $\mathcal{U}$. For any $\delta \in (0, 1)$ and any distribution $\mathcal{D}$ over $\mathcal{Z}$, with probability at least $1 - \delta$ over the draw of N samples $z_1, \ldots, z_N \sim \mathcal{D}$, for any parameter vector $\boldsymbol{\rho} \in \mathcal{P}$, the difference between the average value of $u_{\boldsymbol{\rho}}$ over the samples and the expected value of $u_{\boldsymbol{\rho}}$ is bounded as follows:*

$$\left| \frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(z_i) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(z) \right] \right| = O\left(H \sqrt{\frac{1}{N} \left(d_{\mathcal{U}} + \ln \frac{1}{\delta} \right)} \right).$$

Said another way, for any $\epsilon > 0$, let $N_{\mathcal{U}}(\epsilon, \delta) := \Theta\left(\frac{H^2}{\epsilon^2} \left(d_{\mathcal{U}} + \ln \frac{1}{\delta} \right) \right)$. With probability $1 - \delta$ over the draw of $N \geq N_{\mathcal{U}}(\epsilon, \delta)$ samples $z_1, \ldots, z_N \sim \mathcal{D}$, for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}$, the difference between the average value of $u_{\boldsymbol{\rho}}$ over the samples and the expected value of $u_{\boldsymbol{\rho}}$ is at most ϵ: $\left| \frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(z_i) - \mathbb{E}_{z \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(z) \right] \right| \leq \epsilon$.

Pseudo-dimension is a tool for providing worst-case sample complexity bounds: the definition of pseudo-dimension does not depend on the underlying distribution, and Theorem 2.1.3 holds for any worst-case distribution over the domain.

2.2 Rademacher complexity

Rademacher complexity [31, 113] is another learning-theoretic tool used to measure the complexity of a function class. Unlike pseudo-dimension, Rademacher complexity is a data-dependent quantity: it is a measurement that depends on both the function class $\mathcal{U}$ and the set $\mathcal{S} \subseteq \mathcal{Z}$ of samples. Rademacher complexity intuitively measures the extent to which functions in a class $\mathcal{U}$ match random noise vectors $\boldsymbol{\sigma} \in \{-1, 1\}^N$.

Definition 2.2.1 (Rademacher complexity)**.** The *empirical Rademacher complexity* of the function class $\mathcal{U}$ given a set $\mathcal{S} = \{z_1, \ldots, z_N\} \subseteq \mathcal{Z}$ is $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{N} \mathbb{E}_{\boldsymbol{\sigma} \sim \{-1,1\}^N} \left[\sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma_i u_{\boldsymbol{\rho}}(z_i) \right]$, where each σ_i equals -1 or 1 with equal probability.

Classic learning-theoretic results provide guarantees based on Rademacher complexity, such as the following.

Theorem 2.2.2 (e.g., Mohri et al. [143])**.** *Let $\mathcal{U}$ be a set of functions mapping $\mathcal{Z}$ to $[-H, H]$. For any $\delta \in (0, 1)$, with probability $1 - \delta$ over the draw of N samples $\mathcal{S} = \{z_1, \ldots, z_N\} \sim \mathcal{D}^N$, for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}$, $\left| \frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(z_i) - \mathbb{E}_{a \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(z) \right] \right| = O\left(\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) + H \sqrt{\frac{1}{N} \ln \frac{1}{\delta}} \right)$.*

Pseudo-dimension can be used to provide a worst-case upper bound on a class's Rademacher complexity.

Theorem 2.2.3 ([159]). *Let $\mathcal{U}$ be a set of functions mapping $\mathcal{Z}$ to $[-H, H]$. For any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{\frac{\text{Pdim}(\mathcal{U})}{N}}\right)$.*

Chapter 3

Integer linear programming algorithms

In this chapter, we study the configuration of tree search algorithms. These algorithms are the most widely used tools for solving combinatorial and nonconvex problems throughout artificial intelligence, operations research, and beyond (e.g., [164, 188]). For example, branch-and-bound (B&B) algorithms [119] solve mixed integer linear programs (MILPs), and thus have diverse applications.

A tree search algorithm systematically partitions the search space to find an optimal solution. The algorithm organizes this partition via a tree: the original problem is at the root and the children of a given node represent the subproblems formed by partitioning the feasible set of the parent node. A branch is pruned if it is infeasible or it cannot produce a better solution than the best one found so far by the algorithm. Typically the search space is partitioned by adding an additional constraint on some variable. For example, suppose the feasible set is defined by the constraint $A\boldsymbol{x} \leq \boldsymbol{b}$, with $\boldsymbol{x} \in \{0,1\}^n$. A tree search algorithm might partition this feasible set into two sets, one where $A\boldsymbol{x} \leq \boldsymbol{b}$, $x[1] = 0$, and $x[2], \ldots, x[n] \in \{0,1\}$, and another where $A\boldsymbol{x} \leq \boldsymbol{b}$, $x[1] = 1$, and $x[2], \ldots, x[n] \in \{0,1\}$, in which case the algorithm has *branched on* $x[1]$. A crucial question in tree search algorithm design is determining which variable to branch on at each step. An effective variable selection policy can have a tremendous effect on the size of the tree. Currently, there is no known optimal strategy and the vast majority of existing techniques are backed only by empirical comparisons. In the worst-case, finding an approximately optimal branching variable, even at the root of the tree alone, is NP-hard. This is true even in the case of satisfiability, which is a special case of constraint satisfaction and of MILP [126].

In this chapter, rather than attempt to characterize a branching strategy that is universally optimal, we show empirically and theoretically that it is possible to learn high-performing branching strategies for a given application domain. We model an application domain as a distribution over problem instances, such as a distribution over scheduling problems that an airline solves on a day-to-day basis. The algorithm designer does not know the underlying distribution over problem instances, but has sample access to the distribution. We show how to use samples from the distribution to learn a variable selection policy that will result in as small a search tree as possible in expectation over the underlying distribution.

Our learning algorithm adaptively partitions the parameter space of the variable selection policy into regions where for any parameter in a given region, the resulting tree sizes across the training set are invariant. The learning algorithm returns the empirically optimal parameter over the training set, and thus performs empirical risk minimization (ERM). We prove that the adaptive nature of our algorithm is necessary: performing ERM over a data-independent discretization of the parameter space can yield terrible results. In particular, for any discretization of the parameter space, we provide an infinite family of distributions over MILP instances such that every point in the discretization results in a B&B tree with exponential size in expectation, but there exist infinitely-many parameters outside of the discretized points that result in a tree with constant size with probability 1. A small change in parameters can thus cause a drastic change in the algorithm's behavior. This fact contradicts conventional wisdom. For example, SCIP, the

best open-source MILP solver, sets one of the parameters we investigate to 5/6, regardless of the input MILP's structure. Achterberg [2] wrote that 5/6 was empirically optimal when compared against four other data-independent values. In contrast, our analysis shows that a data-driven approach to parameter tuning can have an enormous benefit.

We present the first sample complexity guarantees for automated configuration of tree search algorithms. We provide worst-case bounds proving that a surprisingly small number of samples are sufficient for strong learnability guarantees: the sample complexity bound grows quadratically in the size of the problem instance, despite the complexity of the algorithms we study.

In our experiments section, we show that on many datasets based on real-world NP-hard problems, different parameters can result in B&B trees of vastly different sizes. Using an optimal parameter setting for one distribution on problems from a different distribution can lead to a dramatic tree size blowup.

The results in this section are joint work with Nina Balcan, Travis Dick, and Tuomas Sandholm. Our existing results appeared in ICML 2018 [21].

3.1 Related research

Several works have studied the use of machine learning techniques in the context of B&B; for an overview, see the summary by Lodi and Zarpellon [132].

As in this work, Khalil et al. [107] study variable selection policies. Their goal is to find a variable selection strategy that mimics the behavior of the classic branching strategy known as *strong branching* while running faster than strong branching. Alvarez et al. [8] study a similar problem, although in their work, the feature vectors in the training set describe nodes from multiple MILP instances. Neither of these works come with any theoretical guarantees, unlike our work.

Several other works study data-driven variable selection from a purely experimental perspective. Di Liberto et al. [66] devise an algorithm that learns how to dynamically switch between different branching heuristics along the branching tree. Karzan et al. [106] propose techniques for choosing problem-specific branching rules based on a partial B&B tree. Ideally, these branching rules will choose variables that will lead to fast fathoming. They do not rely on any techniques from machine learning. In the context of CSP tree search, Xia and Yap [190] apply existing multi-armed bandit algorithms to learning variable selection policies during tree search and Balafrej et al. [17] use a bandit approach to select different levels of propagation during search.

Other works have explored the use of machine learning techniques in the context of other aspects of B&B beyond variable selection. For example, He et al. [95] use machine learning to speed up branch-and-bound, focusing on speeding up the node selection policy. Their work does not provide any learning-theoretic guarantees. Other works that have studied machine learning techniques for branch-and-bound problems other than variable selection include Sabharwal et al. [165], who also study how to devise node selection policies, Hutter et al. [101], who study how to set CPLEX parameters, Kruber et al. [117], who study how to detect decomposable model structure, and Khalil et al. [108], who study how to determine when to run heuristics.

From a theoretical perspective, Le Bodic and Nemhauser [123] present a theoretical model for the selection of branching variables. It is based upon an abstraction of MIPs to a simpler setting in which it is possible to analytically evaluate the dual bound improvement of choosing a given variable. Based on this model, they present a new variable selection policy which has strong performance on many MIPLIB instances. Unlike our work, this paper is unrelated to machine learning.

3.2 Tree search

Tree search is a broad family of algorithms with diverse applications. To exemplify the specifics of tree search, we present a vast family of NP-hard problems — (mixed) integer linear programs — and describe how tree search finds optimal solutions to problems from this family. Later on in Section 3.5, we provide another example of tree search for constraint satisfaction problems.

3.2.1 Mixed integer linear programs

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

Example 3.2.1 (Winner determination)**.** Suppose there is a set $\{1, \ldots, m\}$ of items for sale and a set $\{1, \ldots, n\}$ of buyers. In a combinatorial auction, each buyer i submits bids $v_i(b)$ for any number of bundles $b \subseteq \{1, \ldots, m\}$. The goal of the winner determination problem is to allocate the goods among the bidders so as to maximize *social welfare*, which is the sum of the buyers' values for the bundles they are allocated. We can model this problem as a MILP by assigning a binary variable $x_{i,b}$ for every buyer i and every bundle b they submit a bid $v_i(b)$ on. The variable $x_{i,b}$ is equal to 1 if and only if buyer i receives the bundle b. Let B_i be the set of all bundles b that buyer i submits a bid on. An allocation is feasible if it allocates no item more than once ($\sum_{i=1}^{n} \sum_{b \in B_i, j \ni b} x_{i,b} \leq 1$ for all $j \in \{1, \ldots, m\}$) and if each bidder receives at most one bundle ($\sum_{b \in B_i} x_{i,b} \leq 1$ for all $i \in \{1, \ldots, n\}$). Therefore, the MILP is:

$$\begin{array}{lll} \text{maximize} & \sum_{i=1}^{n} \sum_{b \in B_i} v_i(b) x_{i,b} & \\ \text{s.t.} & \sum_{i=1}^{n} \sum_{b \in B_i, j \ni b} x_{i,b} \leq 1 & \forall j \in [m] \\ & \sum_{b \in B_i} x_{i,b} \leq 1 & \forall i \in [n] \\ & x_{i,b} \in \{0,1\} & \forall i \in [n], b \in B_i. \end{array}$$

MILP tree search

MILPs are typically solved using a tree search algorithm called branch-and-bound (B&B). Given a MILP problem instance, B&B relies on two subroutines that efficiently compute upper and lower bounds on the optimal value within a given region of the search space. The lower bound can be found by choosing any feasible point in the region. An upper bound can be found via a linear programming relaxation. The basic idea of B&B is to partition the search space into convex sets and find upper and lower bounds on the optimal solution within each. The algorithm uses these bounds to form global upper and lower bounds, and if these are equal, the algorithm terminates, since the feasible solution corresponding to the global lower bound must be optimal. If the global upper and lower bounds are not equal, the algorithm refines the partition and repeats.

In more detail, suppose we want to use B&B to solve a MILP z'. B&B iteratively builds a search tree $\mathcal{T}$ with the original MILP z' at the root. In the first iteration, $\mathcal{T}$ consists of a single node containing the MILP z'. At each iteration, B&B uses a *node selection policy* (which we expand on later) to select a leaf node of the tree $\mathcal{T}$, which corresponds to a MILP z. B&B then uses a *variable selection policy* (which we expand on in Section 3.2.1) to choose a variable $x[i]$ of the MILP z to branch on. Specifically, let z_i^+ (resp., z_i^-) be the MILP z except with the additional constraint

Algorithm 1 Branch and bound

Input: A MILP instance z'.

Let $\mathcal{T}$ be a tree that consists of a single node containing the MILP z'.
Let $c^* = -\infty$ be the objective value of the best-known feasible solution.
while there remains an unfathomed leaf in $\mathcal{T}$ **do**
    Use a *node selection policy* to select a leaf of the tree $\mathcal{T}$, which corresponds to a MILP z.
    Use a *variable selection policy* to choose a variable $x[i]$ of the MILP z to branch on.
    Let z_i^+ (resp., z_i^-) be the MILP z except with the constraint that $x[i] = 1$ (resp., $x[i] = 0$).
    Set the right (resp., left) child of z in $\mathcal{T}$ to be a node containing the MILP z_i^+ (resp., z_i^-).
    for $\tilde{z} \in \{z_i^+, z_i^-\}$ **do**
        if the LP relaxation of $\tilde{z}$ is feasible **then**
            Let $\check{\boldsymbol{x}}_{\tilde{z}}$ be an optimal solution to the LP and let $\check{c}_{\tilde{z}}$ be its objective value.
            if the vector $\check{\boldsymbol{x}}_{\tilde{z}}$ satisfies the constraints of the original MILP z' **then**
                Fathom the leaf containing $\tilde{z}$.
                if $c^* < \check{c}_{\tilde{z}}$ **then**
                    Set $c^* = \check{c}_{\tilde{z}}$.
            else if $\check{\boldsymbol{x}}_{\tilde{z}}$ is no better than the best known feasible solution, i.e., $c^* \geq \check{c}_{\tilde{z}}$ **then**
                Fathom the leaf containing $\tilde{z}$.
        else
            Fathom the leaf containing $\tilde{z}$.

that $x[i] = 1$ (resp., $x[i] = 0$). B&B sets the right (resp., left) child of z in $\mathcal{T}$ to be a node containing the MILP z_i^+ (resp., z_i^-). B&B then tries to "fathom" these leafs: the leaf containing z_i^+ (resp., z_i^-) is *fathomed* if:

1. The optimal solution to the LP relaxation of z_i^+ (resp., z_i^-) satisfies the constraints of the original MILP z'.

2. The relaxation of z_i^+ (resp., z_i^-) is infeasible, so z_i^+ (resp., z_i^-) must be infeasible as well.

3. The objective value of the LP relaxation of z_i^+ (resp., z_i^-) is smaller than the objective value of the best known feasible solution, so the optimal solution to z_i^+ (resp., z_i^-) is no better than the best known feasible solution.

B&B terminates when every leaf has been fathomed. It returns the best known feasible solution, which is optimal. See Algorithm 1 for the pseudocode.

The most common node selection policy is the *best bound policy*. Given a B&B tree, it selects the unfathomed leaf containing the MILP z with the maximum LP relaxation objective value. Another common policy is the *depth-first policy*, which selects the next unfathomed leaf in the tree in depth-first order.

Example 3.2.2. In Figure 3.1, we show the search tree built by B&B given as input the following MILP [112]:

$$\begin{array}{ll} \text{maximize} & 40x[1] + 60x[2] + 10x[3] + 10x[4] + 3x[5] + 20x[6] + 60x[7] \\ \text{subject to} & 40x[1] + 50x[2] + 30x[3] + 10x[4] + 10x[5] + 40x[6] + 30x[7] \leq 100 \\ & x[1], \ldots, x[7] \in \{0, 1\}. \end{array} \tag{3.1}$$

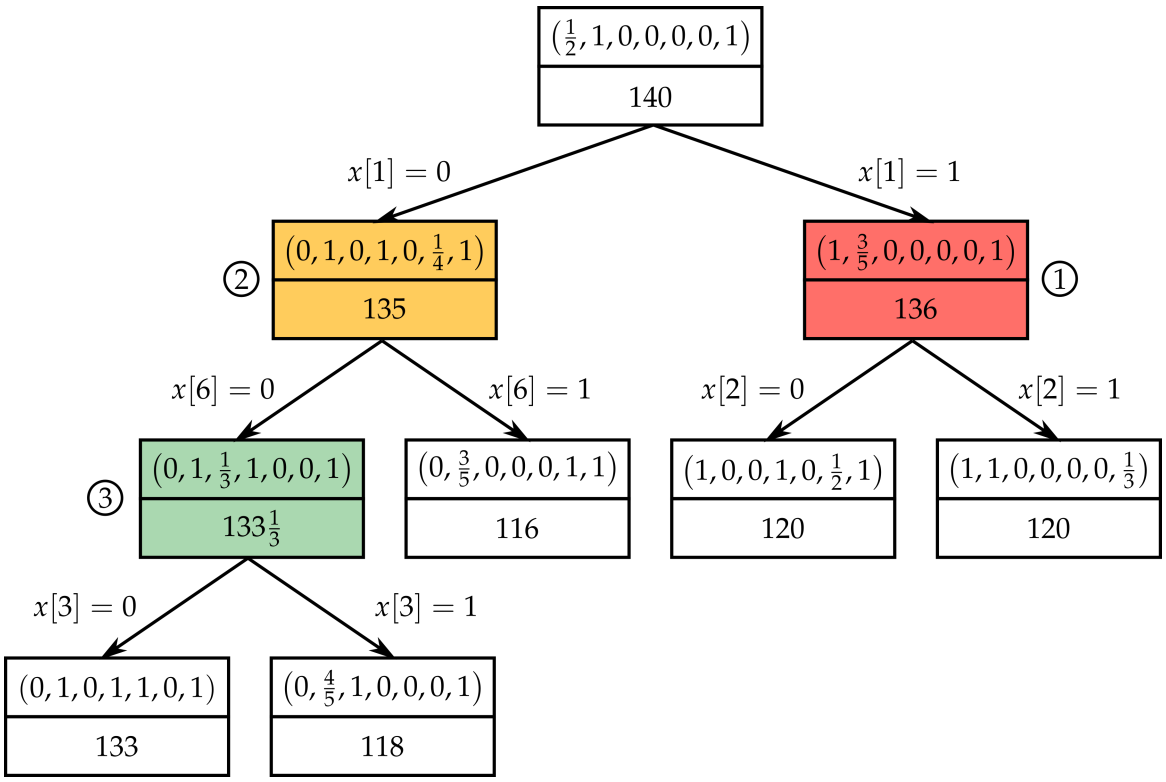


Figure 3.1: Illustration of Example 3.2.2.

Each rectangle denotes a node in the B&B tree. Given a node z, the top portion of its rectangle displays the optimal solution $\check{x}_z$ to the LP relaxation of z, which is the MILP (3.1) with the additional constraints labeling the edges from the root to z. The bottom portion of the rectangle corresponding to z displays the objective value $\check{c}_z$ of the optimal solution to this LP relaxation, i.e., $\check{c}_z = (40, 60, 10, 10, 3, 20, 60) \cdot \check{x}_z$. In this example, the node selection policy is the best bound policy and the variable selection policy selects the "most fractional" variable: the variable $x[i]$ such that $\check{x}_z[i]$ is closest to $\frac{1}{2}$, i.e., $i = \operatorname{argmax}\{\min\{1 - \check{x}_z[i], \check{x}_z[i]\}\}$.

In Figure 3.1, the algorithm first explores the root. At this point, it has the option of exploring either the left or the right child. Since the optimal objective value of the right child (136) is greater than the optimal objective value of the left child (135), B&B will next explore the pink node (marked ①). Next, B&B can either explore either of the pink node's children or the orange node (marked ②). Since the optimal objective value of the orange node (135) is greater than the optimal objective values of the pink node's children (120), B&B will next explore the orange node. After that B&B can explore either of the orange node's children or either of the pink node's children. The optimal objective value of the green node (marked ③) is higher than the optimal objective values of the orange node's right child (116) and the pink node's children (120), so B&B will next explore the green node. At this point, it finds an integral solution, which satisfies all of the constraints of the original MILP (3.1). This integral solution has an objective value of 133. Since all of the other leafs have smaller objective values, the algorithm cannot find a better solution by exploring those leafs. Therefore, the algorithm fathoms all of the leafs and terminates.

Variable selection in MILP tree search

Variable selection policies typically depend on a real-valued *score* per variable $x[i]$.

Definition 3.2.3 (Score-based variable selection policy)**.** Let score be a deterministic function that takes as input a partial search tree $\mathcal{T}$, a leaf z of that tree, and an index i and returns a real value ($\mathrm{score}(\mathcal{T}, z, i) \in \mathbb{R}$). For a leaf z of a tree $\mathcal{T}$, let $N_{\mathcal{T},z}$ be the set of variables that have not yet been branched on along the path from the root of $\mathcal{T}$ to z. A score-based variable selection policy selects the variable $\mathrm{argmax}_{x[j] \in N_{\mathcal{T},z}} \{\mathrm{score}(\mathcal{T}, z, j)\}$ to branch on at the node z.

We list several common definitions of the function score below. Recall that for a MILP z with objective function $\boldsymbol{c} \cdot \boldsymbol{x}$, we denote an optimal solution to the LP relaxation of z as $\check{\boldsymbol{x}}_z = (\check{x}_z[1], \ldots \check{x}_z[n])$. We also use the notation $\check{c}_z$ to denote the objective value of the optimal solution to the LP relaxation of z, i.e., $\check{c}_z = \boldsymbol{c}^\top \check{\boldsymbol{x}}_z$. Finally, we use the notation z_i^+ (resp., z_i^-) to denote the MILP z with the additional constraint that $x[i] = 1$ (resp., $x[i] = 0$). If z_i^+ (resp., z_i^-) is infeasible, then we set $\check{c}_z - \check{c}_{z_i^+}$ (resp., $\check{c}_z - \check{c}_{z_i^-}$) to be some large number greater than $||\boldsymbol{c}||_1$.

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

Linear scoring rule [130]. In this case, $\mathrm{score}(\mathcal{T}, z, i) = (1-\rho) \cdot \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\} + \rho \cdot \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$ where $\rho \in [0,1]$ is a user-specified parameter. This parameter balances an "optimistic" and a "pessimistic" approach to branching: An optimistic approach would choose the variable that maximizes $\max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$, which corresponds to $\rho = 0$, and a pessimistic approach would choose the variable that maximizes $\min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$, which corresponds to $\rho = 1$.

Product scoring rule [2]. In this case, $\mathrm{score}(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^-}, \gamma\right\} \cdot \max\left\{\check{c}_z - \check{c}_{z_i^+}, \gamma\right\}$ where $\gamma = 10^{-6}$. Comparing $\check{c}_z - \check{c}_{z_i^-}$ and $\check{c}_z - \check{c}_{z_i^+}$ to γ allows the algorithm to compare two variables even if $\check{c}_z - \check{c}_{z_i^-} = 0$ or $\check{c}_z - \check{c}_{z_i^+} = 0$. After all, suppose the scoring rule simply calculated the product $\left(\check{c}_z - \check{c}_{z_i^-}\right) \cdot \left(\check{c}_z - \check{c}_{z_i^+}\right)$ without comparing to γ. If $\check{c}_z - \check{c}_{z_i^-} = 0$, then the score equals 0, canceling out the value of $\check{c}_z - \check{c}_{z_i^+}$ and thus losing the information encoded by this difference.

Entropic lookahead scoring rule [82]. Let

$$e(x) = \begin{cases} -x\log_2(x) - (1-x)\log_2(1-x) & \text{if } x \in (0,1) \\ 0 & \text{if } x \in \{0,1\}. \end{cases}$$

Set $\mathrm{score}(\mathcal{T}, z, i) = -\sum_{j=1}^{n} (1 - \check{x}_z[i]) \cdot e\left(\check{x}_{z_i^-}[j]\right) + \check{x}_z[i] \cdot e\left(\check{x}_{z_i^+}[j]\right)$.

Alternative definitions of the linear and product scoring rules. In practice, it is often too slow to compute the differences $\check{c}_z - \check{c}_{z_i^-}$ and $\check{c}_z - \check{c}_{z_i^+}$ for every variable, since it requires solving as many as $2n$ LPs. A faster option is to partially solve the LP relaxations of z_i^- and z_i^+,

starting at $\check{\boldsymbol{x}}_z$ and running a small number of simplex iterations. Denoting the new objective values as $\tilde{c}_{z_i^-}$ and $\tilde{c}_{z_i^+}$, we can revise the linear scoring rule to be $\text{score}(\mathcal{T}, z, i) = (1-\rho) \cdot \max\left\{\check{c}_z - \tilde{c}_{z_i^+}, \check{c}_z - \tilde{c}_{z_i^-}\right\} + \rho \cdot \min\left\{\check{c}_z - \tilde{c}_{z_i^+}, \check{c}_z - \tilde{c}_{z_i^-}\right\}$ and we can revise the product scoring rule to be $\text{score}(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \tilde{c}_{z_i^-}, \gamma\right\} \cdot \max\left\{\check{c}_z - \tilde{c}_{z_i^+}, \gamma\right\}$. Other popular alternatives to computing $\check{c}_{z_i^-}$ and $\check{c}_{z_i^+}$ that fit within our framework are *pseudo-cost branching* [33, 81, 130] and *reliability branching* [3].

3.3 Guarantees for data-driven learning to branch

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

3.3.1 Problem statement

Let $\mathcal{Z}$ be the set of all binary MILPs with at most n variables, for some integer n. Let $\mathcal{D}$ be a distribution over $\mathcal{Z}$. For example, $\mathcal{D}$ could be a distribution over clustering problems a biology lab solves day to day, formulated as MILPs. Let $\text{score}_1, \ldots, \text{score}_d$ be a fixed set of variable selection scoring rules, such as those in Section 3.2.1. Our goal is to learn a convex combination $\rho_1\text{score}_1 + \cdots + \rho_d\text{score}_d$ of the scoring rules that is nearly optimal in expectation over $\mathcal{D}$. More formally, let $u_{\boldsymbol{\rho}}$ be an abstract utility function that takes as input a problem instance z and returns some measure of the quality of B&B using the scoring rule $\rho_1\text{score}_1 + \cdots + \rho_d\text{score}_d$ on input z. For example, $u_{\boldsymbol{\rho}}(z)$ might be zero minus the size of the tree B&B builds (in line with Chapter 2, since our goal is to maximize utility, this would mean we would minimize tree size). Our goal is to bound the pseudo-dimension of $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in [0,1]^d\right\}$.

Our results hold for utility functions that are *tree-constant*, which means that for any problem instance z, so long as the scoring rules $\rho_1\text{score}_1 + \cdots + \rho_d\text{score}_d$ and $\rho'_1\text{score}_1 + \cdots + \rho'_d\text{score}_d$ result in the same search tree, $u_{\boldsymbol{\rho}}(z) = u_{\boldsymbol{\rho}'}(z)$. For example, the size of the search tree is tree-constant.

3.3.2 Impossibility results for data-independent approaches

In this section, we focus on MILP tree search and prove that it is *impossible* to find a nearly optimal B&B configuration using a data-independent discretization of the parameters. Specifically, suppose $\text{score}_1(\mathcal{T}, z, i) = \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$ and $\text{score}_2(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$ and suppose the utility function $u_\rho(z)$ equals zero minus the size of the tree produced by B&B when using a fixed but arbitrary node selection policy. We would like to learn a nearly optimal convex combination $\rho\text{score}_1 + (1-\rho)\text{score}_2$ of these two rules with respect to this utility function. Gauthier and Ribière [81] proposed setting $\rho = 1/2$, Bénichou et al. [33] and Beale [32] suggested setting $\rho = 1$, and Linderoth and Savelsbergh [130] found that $\rho = 2/3$ performs well. Achterberg [2] found that experimentally, $\rho = 5/6$ performed best when comparing among $\rho \in \{0, 1/2, 2/3, 5/6, 1\}$.

We show that for *any* discretization of the parameter space $[0,1]$, there exists an infinite family of distributions over MILP problem instances such that for any parameter in the discretization,

the expected tree size is exponential in n. Yet, there exists an infinite number of parameters such that the tree size is just a constant (with probability 1).

Theorem 3.3.1. *For every a, b such that $\frac{1}{3} < a < b < \frac{1}{2}$ and for all even $n \geq 6$, there exists an infinite family of distributions $\mathcal{D}$ over MILP instances with n variables such that if $\rho \in [0,1] \setminus (a,b)$, then*

$$\mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[u_\rho(z) \right] = -\Omega\left(2^{(n-9)/4}\right)$$

and if $\rho \in (a,b)$, then with probability 1, $u_\rho(z) = -O(1)$. This holds no matter which node selection policy B&B uses.

3.3.3 Sample complexity guarantees

We now bound the pseudo-dimension of the set $\mathcal{U} = \left\{ u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in [0,1]^d \right\}$. First, we provide generalization guarantees for a family of scoring rules we call *path-wise*, which includes many well-known scoring rules as special cases. In this case, the number of samples is surprisingly small given the complexity of these problems: it grows only quadratically with the number of variables. Then, we provide guarantees that apply to any scoring rule, path-wise or otherwise.

Path-wise scoring rules

The guarantees in this section apply broadly to a class of scoring rules we call *path-wise* scoring rules. Given a node z in a search tree $\mathcal{T}$, we denote the path from the root of $\mathcal{T}$ to the node z as $\mathcal{T}_z$. The path $\mathcal{T}_z$ includes all nodes and edge labels from the root of $\mathcal{T}$ to z. For example, Figure 3.2b illustrates the path $\mathcal{T}_z$ from the root of the tree $\mathcal{T}$ in Figure 3.2a to the node labeled z. We now state the definition of path-wise scoring rules.

Definition 3.3.2 (Path-wise scoring rule)**.** The function score is a path-wise scoring rule if for all search trees $\mathcal{T}$, all nodes z in $\mathcal{T}$, and all variables x_i,

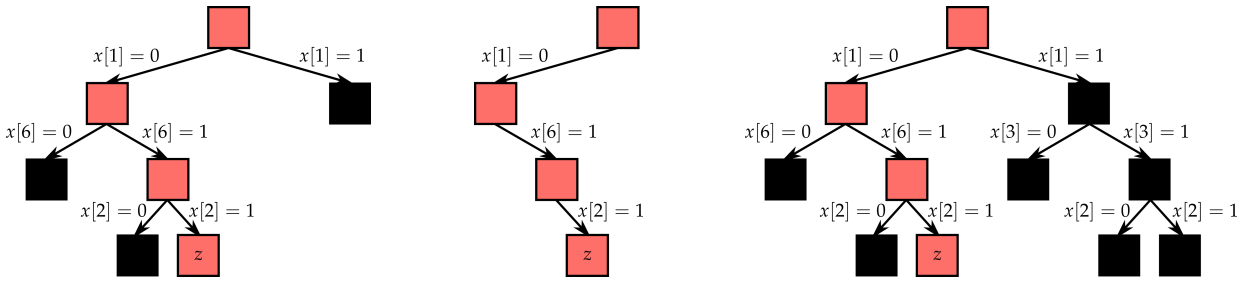
$$\mathrm{score}(\mathcal{T}, z, i) = \mathrm{score}(\mathcal{T}_z, z, i) \tag{3.2}$$

where $\mathcal{T}_z$ is the path from the root of $\mathcal{T}$ to z.[1] See Figure 3.2 for an illustration.

Definition 3.3.2 requires that if the node z appears at the end of the same path in two different B&B trees, then any path-wise scoring rule must assign every variable the same score with respect to z in both trees.

Path-wise scoring rules include many well-studied rules as special cases, such as the *most fractional*, *product*, and *linear* scoring rules, as defined in Section 3.2.1. The same is true when B&B only partially solves the LP relaxations of z_i^- and z_i^+ for every variable $x[i]$ by running a small number of simplex iterations, as we describe in Section 3.2.1 and as is our approach in our experiments. In fact, these scoring rules depend only on the node in question, rather than the path from the root to the node. We present our sample complexity bound for the more general class of path-wise scoring rules because this class captures the level of generality the proof holds for. On the other hand, *pseudo-cost branching* [33, 81, 130] and *reliability branching* [3], two widely-used branching strategies, are not path-wise, but our more general results later in this section do apply to those strategies.

[1] Under this definition, the scoring rule can simulate B&B for any number of steps starting at any point in the tree and use that information to calculate the score, so long as Equality (3.2) always holds.



(a) A B&B search tree $\mathcal{T}$.

(b) The path $\mathcal{T}_z$ from the root of the tree $\mathcal{T}$ in Figure 3.2a to the node labeled z.

(c) Another tree $\mathcal{T}'$ that has the path $\mathcal{T}_z$ as a rooted subtree.

Figure 3.2: Illustrations to accompany the definition of a path-wise scoring rule (Definition 3.3.2). If the scoring rule score is path-wise, then for any variable x_i, $\text{score}(\mathcal{T}, z, i) = \text{score}(\mathcal{T}', z, i) = \text{score}(\mathcal{T}_z, z, i)$.

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

Lemma 3.3.3. *Let* score_1 *and* score_2 *be two path-wise scoring rules and let* z *be an arbitrary problem instance over* n *binary variables. There are* $T \leq 2^{n(n-1)/2} n^n$ *intervals* $I_1, \ldots, I_T$ *partitioning* $[0,1]$ *where for any interval* I_j*, across all* $\rho \in I_j$*, the scoring rule* $\rho\text{score}_1 + (1-\rho)\text{score}_2$ *results in the same search tree.*

This lemma implies the following pseudo-dimension bound.

Theorem 3.3.4. *The pseudo-dimension of* $\mathcal{U}$ *is* $O(n^2)$.

From Theorem 2.1.3, we know that this pseudo-dimension bound implies the following sample complexity guarantee. Let $[-H, H]$ be the range of each utility function $u_\rho : \mathcal{Z} \to [-H, H]$. For any $\epsilon > 0$ and $\delta \in (0,1)$, let $N_{\mathcal{U}}(\epsilon, \delta) := \Theta\left(\frac{H^2}{\epsilon^2}\left(n^2 + \ln\frac{1}{\delta}\right)\right)$. With probability $1-\delta$ over the draw of $N \geq N_{\mathcal{U}}(\epsilon, \delta)$ samples $\mathcal{S} \sim \mathcal{D}^N$, for all parameters $\rho \in [0,1]$, the difference between the average value of u_ρ over the samples and the expected value of u_ρ is at most ϵ: $\left|\frac{1}{N}\sum_{z \in \mathcal{S}} u_\rho(z) - \mathbb{E}_{z \sim \mathcal{D}}\left[u_\rho(z)\right]\right| \leq \epsilon$.

General scoring rules

In this section, we provide generalization guarantees that apply to learning convex combinations of any set of scoring rules. Unlike Theorem 3.3.4, they depend on the size of the search trees B&B is allowed to build. The following lemma corresponds to Lemma 3.3.3 for this setting.

Lemma 3.3.5. *Let* $\text{score}_1, \ldots, \text{score}_d$ *be* d *arbitrary scoring rules and let* z *be an arbitrary MILP over* n *binary variables. Suppose we limit B&B to producing search trees of size* $\bar{\kappa}$*. There is a set* $\mathcal{H}$ *of at most* $n^{2(\bar{\kappa}+1)}$ *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* $\rho_1\text{score}_1 + \cdots + \rho_d\text{score}_d$ *is invariant across all* $(\rho_1, \ldots, \rho_d) \in R$.

This piecewise structure implies the following guarantee.

Theorem 3.3.6. *The pseudo-dimension of $\mathcal{U}$ is $O\left(d\left(\bar{\kappa}\log n + \log d\right)\right)$.*

In contrast to Theorem 3.3.4, this pseudo-dimension bound implies a sample complexity bound of $N_{\mathcal{U}}(\epsilon,\delta) := \Theta\left(\frac{H^2}{\epsilon^2}\left(d\left(\bar{\kappa}\log n + \log d\right) + \ln\frac{1}{\delta}\right)\right)$. In some ways, this bound is stronger than that implied by Theorem 3.3.4 since it holds for d-dimension parameters and any arbitrary set of scoring rules. At the same time, it is weaker in some ways because it depends on the tree size bound $\bar{\kappa}$, whereas the bound implied by Theorem 3.3.4 only depends on the number of variables n.

Learning algorithm

For the case where we wish to learn the optimal tradeoff between two scoring rules, we provide an algorithm that finds the empirically optimal parameter $\hat{\rho}$ given a sample of N problem instances. By our pseudo-dimension bounds, we know that so long as N is sufficiently large, $\hat{\rho}$ is nearly optimal in expectation. Our algorithm stems from the observation that for any tree-constant utility function and any problem instance z, the dual function u_z^* is simple: it is a piecewise-constant function of ρ with a finite number of pieces. This is the same observation that we prove in Lemma 3.3.5. Given a sample of N problem instances, our ERM algorithm constructs all N piecewise-constant functions, takes their average, and finds the minimizer of that function. In practice, we find that the number of pieces making up this piecewise-constant function is small, so our algorithm can learn over a training set of many problem instances.

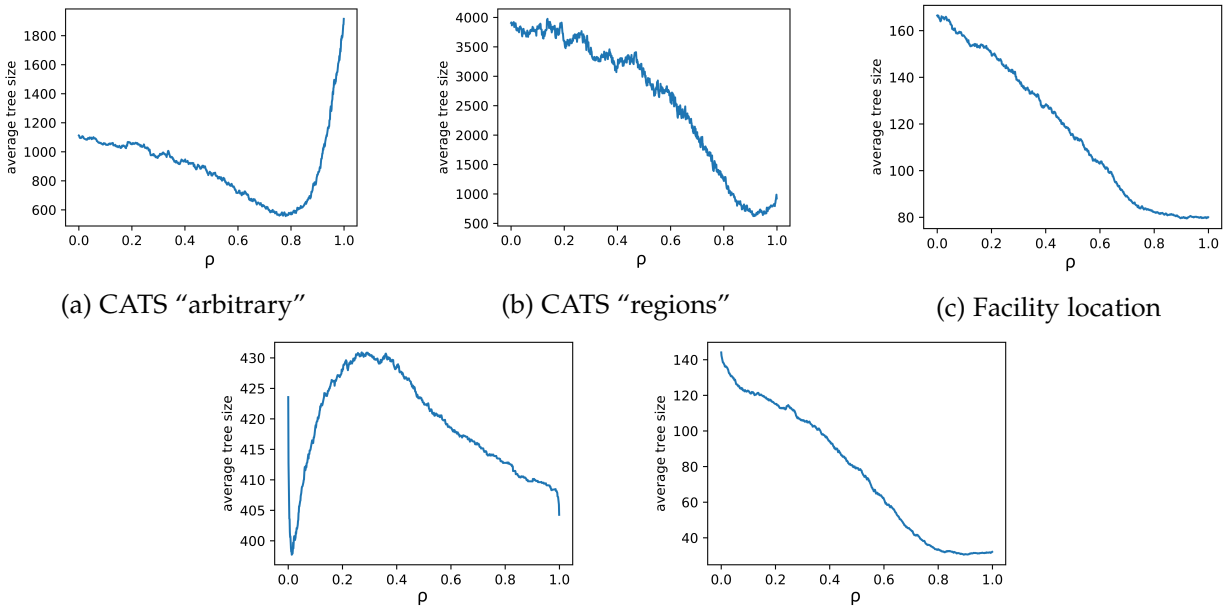
3.4 Experiments

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

Experimental setup. We use the C API of IBM ILOG CPLEX 12.8.0.0 to override the default variable selection rule using a branch callback. Additionally, our callback performs extra bookkeeping to determine a finite set of values for the parameter that give rise to *all* possible B&B trees for a given instance (for the given choice of branching rules that our algorithm is learning to weight). This ensures that there are no good or bad values for the parameter that get skipped; such skipping could be problematic according to our theory in Section 3.3.2. We run CPLEX exactly once for each possible B&B tree on each instance. Following prior research by Fischetti and Monaci [80], Khalil et al. [107], and Karzan et al. [106], we disable CPLEX's cuts and primal heuristics, and we also disable its root-node preprocessing. The CPLEX node selection policy is set to "best bound" (aka. A^* in AI), which is the most typical choice in MILP. All experiments are run on a cluster of 64 c3.large Amazon AWS instances.

For each of the following application domains, Figure 3.3 shows the average B&B tree size produced for each possible value of the ρ parameter for the linear scoring rule, averaged over N independent samples from the distribution.

Combinatorial auctions. We generate $N = 100$ instances of the combinatorial auction winner determination problem under the OR-bidding language [169], which makes this problem equivalent to weighted set packing. The problem is NP-complete. We encode each instance as



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

(d) Linear separators (e) Clustering

Figure 3.3: The average tree size produced by B&B when run with the linear scoring rule with parameter ρ.

a binary MILP (see Example 3.2.1). We use the Combinatorial Auction Test Suite (CATS) [124] to generate these instances. We use the "arbitrary" generator with 200 bids and 100 goods and "regions" generator with 400 bids and 200 goods.

Facility location. Suppose there is a set I of customers and a set J of facilities that have not yet been built. The facilities each produce the same good, and each consumer demands one unit of that good. Consumer i can obtain some fraction y_{ij} of the good from facility j, which costs them $d_{ij}y_{ij}$. Moreover, it costs f_j to construct facility j. The goal is to choose a subset of facilities to construct while minimizing total cost. We generate $N = 500$ instances with 70 facilities and 70 customers each. Each cost d_{ij} is uniformly sampled from $[0, 10^4]$ and each cost f_j is uniformly sampled from $[0, 3 \cdot 10^3]$. This distribution has regularly been used to generate benchmark sets for facility location [5, 82, 84, 97, 111].

Clustering. Given m points $P = \{p_1, \ldots, p_m\}$ and pairwise distances $d(p_i, p_j)$ between each pair of points p_i and p_j, the goal of k-means clustering is to find k *centers* $C = \{c_1, \ldots, c_k\} \subseteq P$ such that the following objective function is minimized: $\sum_{i=1}^{m} \min_{j \in [k]} d\left(p_i, c_j\right)^2$. We generate $N = 500$ instances with 35 points each and $k = 5$. We set $d(i, i) = 0$ for all i and choose $d(i, j)$ uniformly at random from $[0, 1]$ for $i \neq j$. These distances do not satisfy the triangle inequality and they are not symmetric (i.e., $d(i, j) \neq d(j, i)$), which tends to lead to harder MILP instances than using Euclidean distances between randomly chosen points.

Agnostically learning linear separators. Let $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m$ be points in $\mathbb{R}^d$ labeled by $z_1, \ldots, z_m \in \{-1, 1\}$. Suppose we wish to learn a linear separator $\boldsymbol{w} \in \mathbb{R}^d$ that minimizes 0-1 loss, i.e., $\sum_{i=1}^{m} \mathbf{1}_{\{z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle < 0\}}$. We generate $N = 500$ problem instances with 50 points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{50}$ from the

(a) (b)

Figure 3.4: Illustrations of Example 3.5.1.

2-dimensional standard normal distribution. We sample the true linear separator $\boldsymbol{w}^*$ from the 2-dimensional standard Gaussian distribution and label point $\boldsymbol{p}_i$ by $z_i = \text{sign}(\langle \boldsymbol{w}^*, \boldsymbol{p}_i \rangle)$. We then choose 10 random points and flip their labels so that there is no consistent linear separator.

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

3.5 Constraint satisfaction problems

In this section, we describe tree search for constraint satisfaction problems. The generalization guarantee from Section 3.3.3 also applies to tree search in this domain.

A constraint satisfaction problem (CSP) is a tuple (X, D, C), where $X = \{x_1, \ldots, x_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of domains where D_i is the set of values variable x_i can take on, and C is a set of constraints between variables. Each constraint in C is a pair $((x_{i_1}, \ldots, x_{i_r}), \psi)$ where ψ is a function mapping $D_{i_1} \times \cdots \times D_{i_r}$ to $\{0, 1\}$ for some $r \in [n]$ and some $i_1, \ldots, i_r \in [n]$. Given an assignment $(y_1, \ldots, y_n) \in D_1 \times \cdots \times D_n$ of the variables in X, a constraint $((x_{i_1}, \ldots, x_{i_r}), \psi)$ is satisfied if $\psi(y_{i_1}, \ldots, y_{i_r}) = 1$. The goal is to find an assignment that maximizes the number of satisfied constraints.

The *degree* of a variable x, denoted $\deg(x)$, is the number of constraints involving x. The *dynamic degree* of (an unassigned variable) x given a partial assignment $\boldsymbol{y}$, denoted $\text{ddeg}(x, \boldsymbol{y})$ is the number of constraints involving x and at least one other unassigned variable.

Example 3.5.1 (Graph k-coloring)**.** Given a graph, the goal of this problem is to color its vertices using at most k colors such that no two adjacent vertices share the same color. This problem can be formulated as a CSP, as illustrated by the following example. Suppose we want to 3-color the graph in Figure 3.4a using pink, green, and orange. The four vertices correspond to the four variables $X = \{x_1, \ldots, x_4\}$. The domain $D_1 = \cdots = D_4 = \{\text{pink, green, orange}\}$. The only constraints on this problem are that no two adjacent vertices share the same color. Therefore we define ψ to be the "not equal" relation mapping $\{\text{pink, green, orange}\} \times \{\text{pink, green, orange}\} \to \{0, 1\}$ such that $\psi(\omega_1, \omega_2) = \mathbf{1}_{\{\omega_1 \neq \omega_2\}}$. Finally, we define the set of constraints to be

$$C = \{((x_1, x_2), \psi), ((x_1, x_3), \psi), ((x_2, x_3), \psi), ((x_3, x_4), \psi)) \,.$$

See Figure 3.4b for a coloring that satisfies all constraints ($y_1 = y_4 =$ green, $y_2 =$ orange, and $y_3 =$ pink).

3.5.1 CSP tree search

CSP tree search begins by choosing a variable x_i with domain D_i and building $|D_i|$ branches, each one corresponding to one of the $|D_i|$ possible value assignments of x. Next, a node z of the tree is chosen, another variable x_j is chosen, and $|D_j|$ branches from z are built, each corresponding to the possible assignments of x_j. The search continues and a branch is pruned if any of the constraints are not feasible given the partial assignment of the variables from the root to the leaf of that branch.

3.5.2 Variable selection in CSP tree search

As in MILP tree search, there are many variable selection policies researchers have suggested for choosing which variable to branch on at a given node. Typically, algorithms associate a score for branching on a given variable x_i at node z in the tree $\mathcal{T}$, as in B&B. The algorithm then branches on the variable with the highest score. We provide several examples of common variable selection policies below.

deg/dom and ddeg/dom [34]: deg/dom corresponds to the scoring rule $\text{score}(\mathcal{T}, z, i) = \frac{\deg(x_i)}{|D_i|}$ and ddeg/dom corresponds to the scoring rule $\text{score}(\mathcal{T}, z, i) = \frac{\text{ddeg}(x_i, \boldsymbol{y})}{|D_i|}$, where $\boldsymbol{y}$ is the assignment of variables from the root of $\mathcal{T}$ to z.

Smallest domain [92]: In this case, $\text{score}(\mathcal{T}, z, i) = \frac{1}{|D_i|}$.

Our theory is for tree search and applies to both MILPs and CSPs. It applies both to lookahead approaches that require learning the weighting of the two children (the more promising and less promising child) and to approaches that require learning the weighting of several different scoring rules.

Chapter 4

Integer quadratic programming algorithms

In this chapter, we study algorithm configuration for a class of integer quadratic programming approximation algorithms. This class consists of SDP rounding algorithms and is a generalization of the seminal Goemans-Williamson (GW) max-cut algorithm [83]. We focus on integer quadratic programs of the form $\sum_{i,j\in[n]} a_{ij}x_ix_j$, where the goal is to find an assignment of the binary variables $X = \{x_1, \ldots, x_n\}$ maximizing this sum for a given matrix $A = (a_{ij})_{i,j\in[n]}$. Specifically, each variable in X is set to either -1 or 1. This problem is also known as MaxQP [45]. Most algorithms with the best approximation guarantees use an SDP relaxation. The SDP relaxation has the form

$$\text{maximize} \sum_{i,j\in[n]} a_{ij}\langle \boldsymbol{u}_i, \boldsymbol{u}_j\rangle \qquad \text{subject to } \boldsymbol{u}_i \in S^{n-1}. \tag{4.1}$$

Given the set of vectors $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$, we must decide how they represent an assignment of the binary variables in X. In the GW algorithm, the vectors are projected onto a random vector $\boldsymbol{Z}$ drawn from the n-dimensional Gaussian distribution $\mathcal{Z}$. If the directed distance of the resulting projection is greater than 0, then the corresponding binary variable is set to 1, and otherwise it is set to -1.

In some cases, the GW algorithm can be improved upon by probabilistically assigning each binary variable to 1 or -1. In the final rounding step, any rounding function $r : \mathbb{R} \to [-1, 1]$ can be used to specify that a variable x_i is set to 1 with probability $\frac{1}{2} + \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{u}_i\rangle\right)$ and -1 with probability $\frac{1}{2} - \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{u}_i\rangle\right)$. See Algorithm 2 for the pseudocode. Algorithm 2 is known as a *Random Projection, Randomized Rounding* (RPR^2) algorithm, so named by the seminal work of Feige and Langberg [74].

We focus on the class of s-linear rounding functions in this section. For the max-cut problem, Feige and Langberg [74] prove that when the maximium cut in the graph is not very large, a worst-case approximation ratio above the GW ratio is possible using an s-linear rounding function. An s-linear rounding function $\phi_s : \mathbb{R} \to [-1, 1]$ is parameterized by a real-value $s > 0$. The function ϕ_s is defined as follows:

$$\phi_s(y) = \begin{cases} -1 & \text{if } y < -s \\ y/s & \text{if } -s \le y \le s \\ 1 & \text{if } y > s. \end{cases}$$

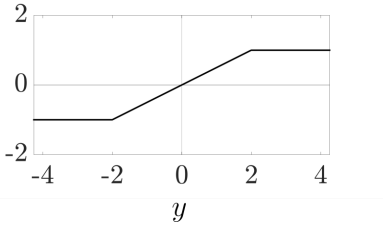


Figure 4.1: A graph of the 2-linear function ϕ_2.

Our goal is to design an algorithm L_{slin} that learns a nearly-optimal s-linear rounding function. In other words, we want to find a parameter s such that the expected objective value $\sum_{i,j\in[n]} a_{ij}x_ix_j$ is maximized, where the expectation is over three sources of randomness: the matrix A, the vector $\boldsymbol{Z}$, and the final assignment of the variables $x_1, \ldots, x_n$, which depends on A,

Algorithm 2 SDP rounding algorithm with rounding function r

Input: Matrix $A \in \mathbb{R}^{n \times n}$.

1: Draw a random vector $\boldsymbol{Z}$ from $\mathcal{Z}$, the n-dimensional Gaussian distribution.
2: Solve the SDP (4.1) for the optimal embedding $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$.
3: Compute set of fractional assignments $r(\langle \boldsymbol{Z}, \boldsymbol{u}_1 \rangle), \ldots, r(\langle \boldsymbol{Z}, \boldsymbol{u}_n \rangle)$.
4: For all $i \in [n]$, set x_i to 1 with probability $\frac{1}{2} + \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{u}_i \rangle\right)$ and -1 with probability $\frac{1}{2} - \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{u}_i \rangle\right)$.

Output: $x_1, \ldots, x_n$.

$\boldsymbol{Z}$, and the choice of a parameter s. This expected value is thus over distributions that are both external and internal to Algorithm 2: the unknown distribution over matrices is external and defines the algorithm's input, whereas the distribution over vectors and the distribution defining the final assignment of the variables $x_1, \ldots, x_n$ are internal to Algorithm 2. We call this expected value the *true utility of the parameter* s.

Since the distribution $\mathcal{D}$ over matrices is unknown, we cannot evaluate the true utility of any parameter, so we use samples to find a nearly optimal parameter. We draw samples from the first two sources of randomness: the distribution over matrices and the distribution over vectors. Thus, our set of samples has the form $\mathcal{S} = \left\{\left(A^{(1)}, \boldsymbol{Z}^{(1)}\right), \ldots, \left(A^{(N)}, \boldsymbol{Z}^{(N)}\right)\right\} \sim (\mathcal{D} \times \mathcal{Z})^N$. In this way, to ease our analysis, we sample the distribution over Gaussians — an internal source of randomness — rather than analyzing its expected value directly. Given these samples, we define the *empirical utility* of a parameter s to be the expected value of the solution returned by Algorithm 2 given A as input when it uses the hyperplane $\boldsymbol{Z}$ and the s-linear rounding function ϕ_s in Step 3, averaged over all $(A, \boldsymbol{Z}) \in \mathcal{S}$. At a high level, upon sampling from the first two sources of randomness, we have isolated the third source of randomness, whose expectation is simple to analyze. In the following analysis, we show that every parameter's empirical utility converges to its true utility as the sample size increases, and thus the parameter with the highest empirical utility has a nearly optimal true utility.

Since the distribution over vectors is known to be Gaussian, an alternative route would be to only sample the external source of randomness $\mathcal{D}$ over the matrices. We would then define the empirical utility of a parameter s to be the expected value of the solution returned by Algorithm 2 given A as input when it uses the s-linear rounding function ϕ_s in Step 3, averaged over all $A \in \mathcal{S}$. This would require us to incorporate the density function of a multi-dimensional Gaussian in our analysis. We abstract out this complication by sampling the Gaussian vectors and including them as a part of the learning algorithm's training set, thus simplifying the analysis significantly.

We now define the true and empirical utility of a parameter more formally. Let $p_{(i,\boldsymbol{Z},A,s)}$ be the distribution from which the value of x_i is drawn when Algorithm 2, given A as input, uses the hyperplane $\boldsymbol{Z}$ and the rounding function $r = \phi_s$ in Step 3. The true utility of the parameter s is $\mathbb{E}_{A,\boldsymbol{Z} \sim \mathcal{D} \times \mathcal{Z}}\left[\mathbb{E}_{x_i \sim p_{(i,\boldsymbol{Z},A,s)}}\left[\sum_{i,j} a_{ij} x_i x_j\right]\right]$.[1] Our goal is to find a parameter whose true utility is (nearly) optimal. Said another way, we want to find the value of s leading to the highest expected objective value over all sources of randomness.

[1] We use the abbreviated notation

$$\mathop{\mathbb{E}}_{A,\boldsymbol{Z} \sim \mathcal{D} \times \mathcal{Z}}\left[\mathop{\mathbb{E}}_{x_i \sim p_{(i,\boldsymbol{Z},A,s)}}\left[\sum_{i,j} a_{ij} x_i x_j\right]\right] = \mathop{\mathbb{E}}_{A,\boldsymbol{Z} \sim \mathcal{D} \times \mathcal{Z}}\left[\mathop{\mathbb{E}}_{x_1 \sim p_{(1,\boldsymbol{Z},A,s)}, \ldots, x_n \sim p_{(n,\boldsymbol{Z},A,s)}}\left[\sum_{i,j} a_{ij} x_i x_j\right]\right].$$

Algorithm 3 An algorithm for finding an empirical value maximizing s-linear rounding function

Input: Set of samples $(A^{(1)}, \boldsymbol{Z}^{(1)}), \ldots, (A^{(m)}, \boldsymbol{Z}^{(N)})$

1: For all $i \in [N]$, solve for the SDP embedding $U^{(i)}$ of $A^{(i)}$, where $U^{(i)} = \left\{\boldsymbol{u}_1^{(i)}, \ldots, \boldsymbol{u}_n^{(i)}\right\}$.

2: Let $T = \left\{s_1, \ldots, s_{|T|}\right\}$ be the set of all values $s > 0$ such that there exists a pair of indices $j \in [n], i \in [N]$ with $\left|\left\langle \boldsymbol{Z}^{(i)}, \boldsymbol{u}_j^{(i)} \right\rangle\right| = s$.

3: For $i \in [|T| - 1]$, let $\hat{s}_i$ be the value in $[s_i, s_{i+1}]$ which maximizes $\frac{1}{N}\sum_{i=1}^{N} u_{A^{(i)}, \boldsymbol{Z}^{(i)}}(s)$.

4: Let $\hat{s}$ be the value in $\{\hat{s}_1, \ldots, \hat{s}_{|T|-1}\}$ that maximizes $\frac{1}{N}\sum_{i=1}^{N} u_{A^{(i)}, \boldsymbol{Z}^{(i)}}(s)$.

Output: $\hat{s}$

We do not know the distribution $\mathcal{D}$ over matrices, so we also need to define the *empirical utility* of the parameter s given a set of samples. We will then show that this empirical utility approaches the true utility as the number of samples grows. Thus, a parameter which is nearly optimal on average over the samples will be nearly optimal in expectation as well. The definition of a parameter's empirical utility depends on a function u_s: let $u_s(A, \boldsymbol{Z})$ denote the expected value of the solution returned by Algorithm 2 given A as input when it uses the hyperplane $\boldsymbol{Z}$ and the rounding function $r = \phi_s$ in Step 3. The expectation is over the randomness in the assignment of each variable x_i to either 1 or -1. Explicitly, $u_s(A, \boldsymbol{Z}) = \mathbb{E}_{x_i \sim p_{(i, \boldsymbol{Z}, A, s)}}\left[\sum_{i,j} a_{ij} x_i x_j\right]$. By definition, the true utility of the parameter s equals $\mathbb{E}_{A, \boldsymbol{Z} \sim \mathcal{D} \times \mathcal{Z}}\left[u_s(A, \boldsymbol{Z})\right]$.

We now define the empirical utility of a parameter s as follows. Given a set of samples $\left(A^{(1)}, \boldsymbol{Z}^{(1)}\right), \ldots, \left(A^{(N)}, \boldsymbol{Z}^{(N)}\right) \sim \mathcal{D} \times \mathcal{Z}$, we define the empirical utility of the parameter s to be $\frac{1}{N}\sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$. Bounding the pseudo-dimension[2] of the class of functions $\mathcal{U} = \{u_s : s > 0\}$, we bound the number of samples sufficient to ensure that with high probability, for all parameters s, the true utility of s nearly matches its expected utility. In other words, $\frac{1}{N}\sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$ nearly matches $\mathbb{E}_{A, \boldsymbol{Z} \sim \mathcal{D} \times \mathcal{Z}}\left[u_s(A, \boldsymbol{Z})\right]$. Thus, if we find the parameter $\hat{s}$ that maximizes $\frac{1}{N}\sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$, then the true utility of $\hat{s}$ is nearly optimal. In Theorem 4.0.3, we provide a sample efficient and computationally efficient algorithm for finding $\hat{s}$.

We prove that the functions in $\mathcal{U}$ have a particularly simple form, which facilitates our pseudo-dimension analysis. Roughly speaking, for a fixed matrix A and vector $\boldsymbol{Z}$, each function in $\mathcal{U}$ is a piecewise inverse-quadratic function of the parameter s.

Lemma 4.0.1. *For any matrix A and vector $\boldsymbol{Z}$, let $u^*_{A,\boldsymbol{Z}} : \mathbb{R}_{>0} \to \mathbb{R}$ denote the function $u^*_{A,\boldsymbol{Z}}(s) = u_s(A, \boldsymbol{Z})$. Each function $u^*_{A,\boldsymbol{Z}}$ is made up of $n + 1$ piecewise components of the form $\frac{a}{s^2} + \frac{b}{s} + c$ for some $a, b, c \in \mathbb{R}$. Moreover, if the border between two components falls at some $s \in \mathbb{R}_{>0}$, then it must be that $s = |\langle \boldsymbol{u}_i, \boldsymbol{Z}\rangle|$ for some $\boldsymbol{u}_i$ in the optimal SDP embedding of A.*

This fact implies the following pseudo-dimension bound.

Theorem 4.0.2. *The pseudo-dimension of $\mathcal{U}$ is $O(\ln n)$.*

Lemma 4.0.1 also suggests a learning algorithm, Algorithm 3, that is computationally and sample efficient.

[2]Since pseudo-dimension bounds imply uniform convergence guarantees for worst-case distributions, the distribution $\mathcal{Z}$ over vectors need not be Gaussian, although this is the classic distribution of choice in the works by Goemans and Williamson [83] and Feige and Langberg [74]. Indeed, our results hold when $\mathcal{Z}$ is any arbitrary distribution over $\mathbb{R}^n$.

Theorem 4.0.3. *Let $H = \sup_{A \in \text{supp}(\mathcal{D})} ||A||_c$, where $||\cdot||_c$ is the cut norm and supp$(\mathcal{D})$ denotes the support of $\mathcal{D}$.*[3] *Given a set of $N = \Theta\left(\left(\frac{H}{\epsilon}\right)^2 \log \frac{n}{\delta}\right)$ samples drawn from $\mathcal{D} \times \mathcal{Z}$, let $\hat{s}$ be the output of Algorithm 3. With probability at least $1 - \delta$, the true utility of $\hat{s}$ is ϵ-close optimal:*

$$\max_{s>0} \mathop{\mathbb{E}}_{A\sim\mathcal{D},\mathbf{Z}\sim\mathcal{Z}} [u_s(A, \mathbf{Z})] - \mathop{\mathbb{E}}_{A\sim\mathcal{D},\mathbf{Z}\sim\mathcal{Z}} [u_{\hat{s}}(A, \mathbf{Z})] \leq \epsilon.$$

The results in this chapter are joint work with Nina Balcan, Vaishnavh Nagarajan, and Colin White [20]. Our existing results appeared in COLT 2017.

[3] H is an upper bound on the value of $u_s(A, \mathbf{Z})$ for any $s > 0$ and any $(A, \mathbf{Z})$ in the support of $\mathcal{D} \times \mathcal{Z}$.

Chapter 5

Computational biology algorithms

In this chapter, we study the sample complexity of three common problems from biology: pairwise sequence alignment, RNA folding, and predicting topologically associated domains. In all of these applications, there are two unifying similarities, which we describe below.

First, a solution's quality, which we also refer to as its utility, is measured with respect to a ground-truth solution. This gold-standard solution is constructed in most cases by laboratory experimentation, so it is only available for the problem instances in the training set. Algorithmic performance is then measured in terms of the distance between the solution the algorithm outputs and the ground-truth solution.

Second, the biology algorithms we study all return solutions that maximize some parameterized objective function. Often, there may be multiple solutions that maximize this objective function; we call these solutions *co-optimal*. Although co-optimal solutions have the same objective function value, they may have different utilities. In practice, in any region of the parameter space where the set of co-optimal solutions is invariant, the algorithm's output is invariant as well. We call this type of algorithm *co-optimal constant*. Throughout the remainder of this section, we assume the parameterized algorithms are co-optimal constant.

The results in this chapter are joint work with Nina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, and Tuomas Sandholm [25].

5.1 Global pairwise sequence alignment

Pairwise sequence alignment is a fundamental problem in biological sequence analysis, database search [7], homology detection [158], and many other scientific domains. Pairwise alignment is also a basic operation in many tools for multiple sequence alignment, where the objective is to find correlation between at least three strings [174]. Depending on the application, the goal may be to find a complete alignment of the two sequences, called *global alignment*, or to find the best alignment of any subsequences of the two input sequences, called *local alignment*. In either case, the high level goal is the same: given two sequences, find an alignment that optimizes a given parameterized objective function. While the pairwise sequence alignment problem in general is well studied, most common problem formulations have the same issue: optimizing the objective function's parameters can be challenging. Depending on the application domain, the best parameter values differ greatly between instances.

More formally, let Σ be an alphabet and let S_1 and S_2 be two sequences in Σ of length n. A *sequence alignment* is a pair of sequences $\tau_1, \tau_2 \in (\Sigma \cup \{-\})^*$ such that $|\tau_1| = |\tau_2|$, $\mathrm{del}(\tau_1) = S_1$, and $\mathrm{del}(\tau_2) = S_2$, where del is a function that deletes every $-$, or *gap character*, in the input sequence. We require that a gap character is never paired with a gap character: for all $i \in [|\tau_1|]$, if $\tau_1[i] = -$, then $\tau_2[i] \neq -$ and vice versa. There are many features of a sequence alignment that affect its quality, such as the number of *matches* (indices i where $\tau_1[i] = \tau_2[i]$), *mismatches* (indices i where $\tau_1[i] \neq \tau_2[i]$), *indels* (indices i where $\tau_1[i] = -$ or $\tau_2[i] = -$), and *gaps* (ranges $[i, j]$ where for $k \in \{1, 2\}$, $\tau_k[\ell] = -$ for all $\ell \in [i, j]$, $\tau_k[i-1] \neq -$ or $i = 1$, and $\tau_k[j+1] \neq -$ or $j = n$). We

denote these features by functions $\ell_1, \ldots, \ell_d$, where each maps pairs of sequences (S_1, S_2) and alignments L to real values $\ell_j(S_1, S_2, L) \in \mathbb{R}$.

When aligning two input sequences S_1 and S_2, the goal is to compute the alignment that maximizes the objective function

$$\rho[1] \cdot \ell_1(S_1, S_2, L) + \cdots + \rho[d] \cdot \ell_d(S_1, S_2, L), \tag{5.1}$$

where $\boldsymbol{\rho} \in \mathbb{R}^d$ is a parameter vector. We use the notation $\mathcal{L}_{\boldsymbol{\rho}}(S_1, S_2)$ to denote the set of alignments maximizing Equation (5.1). For each parameter vector $\boldsymbol{\rho}$, we can run a dynamic programming algorithm $A_{\boldsymbol{\rho}}$ which returns an alignment $A_{\boldsymbol{\rho}}(S_1, S_2)$ in $\mathcal{L}_{\boldsymbol{\rho}}(S_1, S_2)$. As we vary the weights, this gives rise to a family of algorithms. Since there is no consensus about what the best weights are, our goal is to automatically learn the best weights for a specific application domain. We assume that the domain expert has a utility function that characterizes an alignment's quality, denoted $u(S_1, S_2, L) \in [0, 1]$. We are agnostic to the specific definition of u. As a concrete example, $u(S_1, S_2, L)$ might measure the distance between L and a "ground truth" alignment of S_1 and S_2, also known as the *developer's accuracy* [175]. In this case, the learning algorithm would require access to the ground truth alignment for every problem instance (S_1, S_2) in the training set. Ground truth is difficult to measure, so these reference alignments are never available for all sequence pairs.

We prove that for any fixed sequence pair, utility as a function of the parameters $\boldsymbol{\rho} \in \mathbb{R}^d$ is piecewise constant, and that the boundaries between pieces are defined by $4^n n^{4n+2}$ hyperplanes. This implies the following pseudo-dimension bound.

Theorem 5.1.1. *Let $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$ be a set of co-optimal-constant algorithms and let u be a utility function mapping tuples (S_1, S_2, L) of sequence pairs and alignments to $\mathbb{R}$. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2)) \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$ mapping sequence pairs $S_1, S_2 \in \Sigma^n$ to $\mathbb{R}$. The pseudo-dimension of $\mathcal{U}$ is $O(d(n \ln n + \ln d))$.*

From Theorem 2.1.3, we know that this pseudo-dimension bound implies the following sample complexity guarantee. Let $[0, 1]$ be the range of each utility function $u_{\boldsymbol{\rho}} : \Sigma^n \times \Sigma^n \to [0, 1]$. For any $\epsilon > 0$ and $\delta \in (0, 1)$, let $N_{\mathcal{U}}(\epsilon, \delta) := \Theta\left(\frac{1}{\epsilon^2}\left(d(n \ln n + \ln d) + \ln \frac{1}{\delta}\right)\right)$. With probability $1 - \delta$ over the draw of $N \geq N_{\mathcal{U}}(\epsilon, \delta)$ samples $\mathcal{S} \sim \mathcal{D}^N$, for all parameter vectors $\boldsymbol{\rho} \in \mathbb{R}^d$, the difference between the average value of $u_{\boldsymbol{\rho}}$ over the samples and the expected value of $u_{\boldsymbol{\rho}}$ is at most ϵ: $\left|\frac{1}{N}\sum_{(S_1,S_2)\in\mathcal{S}} u_{\boldsymbol{\rho}}(S_1, S_2) - \mathbb{E}_{(S_1,S_2)\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(S_1, S_2)\right]\right| \leq \epsilon$.

Tighter guarantees for a structured algorithm subclass: the affine-gap model. A line of prior work [78, 90, 153, 154] analyzed the specific instantiation of the objective function (5.1) where $d = 3$. The goal is to find the alignment L maximizing the objective function

$$\textsc{mt}(S_1, S_2, L) - \rho[1] \cdot \textsc{ms}(S_1, S_2, L) - \rho[2] \cdot \textsc{id}(S_1, S_2, L) - \rho[3] \cdot \textsc{gp}(S_1, S_2, L),$$

where $\textsc{mt}(S_1, S_2, L)$ is the number of columns in the alignment that have the same character (*matches*), $\textsc{ms}(S_1, S_2, L)$ is the number of columns that do not have the same character (*mismatches*), $\textsc{id}(S_1, S_2, L)$ is the total number of gap characters (*indels*, short for insertion/deletion), and $\textsc{gp}(S_1, S_2, L)$ is the number of groups of consecutive gap characters in any one row of the grid (*gaps*). This is known as the *the affine-gap scoring model*. We exploit specific structure exhibited by this algorithm family to obtain an exponential improvement in the sample complexity. This useful structure guarantees that for any pair of sequences (S_1, S_2), there are only $O\left(n^{3/2}\right)$ different alignments the algorithm family $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^3\}$ might produce as we range over parameter

vectors [78, 90, 153]. This implies that for any fixed sequence pair, utility as a function of the parameters $\boldsymbol{\rho} \in \mathbb{R}^3$ is piecewise constant, and that the boundaries between pieces are defined by $O(n^3)$ hyperplanes. This implies the following pseudo-dimension bound.

Theorem 5.1.2. *Let $\left\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^3_{\geq 0}\right\}$ be a set of co-optimal-constant algorithms and let u be a utility function mapping tuples (S_1, S_2, L) of sequence pairs and alignments to $\mathbb{R}$. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u\left(S_1, S_2, A_{\boldsymbol{\rho}}\left(S_1, S_2\right)\right) \mid \boldsymbol{\rho} \in \mathbb{R}^3_{\geq 0}\right\}$ mapping sequence pairs $S_1, S_2 \in \Sigma^n$ to $\mathbb{R}$. The pseudo-dimension of $\mathcal{U}$ is $O(\ln n)$.*

This theorem implies a sample complexity bound of $N_{\mathcal{U}}(\epsilon, \delta) := \Theta\left(\frac{1}{\epsilon^2}\left(\ln n + \ln \frac{1}{\delta}\right)\right)$, which is exponentially tighter than that implied by Theorem 5.1.1. We also prove that our pseudo-dimension bound from Theorem 5.1.2 is tight up to constant factors and generalize our guarantees to the case where we align multiple sequences.

5.2 RNA folding

RNA molecules have many essential roles, including protein coding and enzymatic functions [96]. RNA is assembled as a chain of *bases* denoted using the characters A, U, C, and G. It is often found as a single strand folded onto itself: non-adjacent bases physically bound together. Given an unfolded RNA strand, the goal is to infer the way it would naturally fold, which sheds light on its function. This problem is known as *RNA secondary structure prediction*, or simply *RNA folding*.

Formally, given a sequence $S \in \Sigma^n$ from an alphabet Σ, a *folding* is a set of a pairs (i, j) such that $0 \leq i < j \leq n$, each base is involved in only one pair, and the folding does not contain any pseudoknots (a pair of pairs $(i, j), (i', j')$ such that $i < i' < j < j'$). A commonly-used procedure for finding a folding $\phi \subset \{(i, j) \mid 0 \leq i < j < n\}$ of an input sequence $S \subseteq \Sigma^n$ computes the folding that maximizes the objective function

$$\rho\,|\phi| + (1-\rho) \sum_{(i,j)\in\phi} M_{\binom{S[i],S[j]}{S[i-1],S[j+1]}}^{\mathbb{I}_{\{(i,j),(i-1,j+1)\in\phi\}}} \tag{5.2}$$

where $\rho \in [0, 1]$ is a tunable parameter and M is a fixed, arbitrary, non-negative weight matrix. The parameter ρ trades off between global properties of the folding (the number of binding pairs $|\phi|$) and local properties (the likelihood bases would appear close together in the folding, as indicated by the matrix M). The model described here is a special case of that described in Nussinov and Jacobson [150]. We use the notation $\phi_\rho(S)$ to denote the set of foldings maximizing Equation (5.2). For each parameter ρ, we can run a dynamic programming algorithm A_ρ which returns a folding $A_\rho(S)$ in $\phi_\rho(S)$. As we vary the weight, this gives rise to a family of algorithms. Since there is no consensus about what the best weight is, our goal is to automatically learn the best weight for a specific application domain. As in Section 5.1, we assume that the domain expert has a utility function that characterizes a folding's quality, denoted $u(S, \phi) \in [0, 1]$. We are again agnostic to the specific definition of u, but as a concrete example, $u(S, \phi)$ might measure the fraction of pairs shared between ϕ and a "ground truth" folding ϕ^*. In this case, the learning algorithm would require access to the ground truth folding for every sequence S in the training set.

In the following theorem, we prove that the utility function u, when applied to the output of the algorithm A_ρ, is a piecewise-constant function of the parameter ρ with at most n pieces. This implies the following pseudo-dimension bound.

Theorem 5.2.1. *Let $\{A_\rho \mid \rho \in [0,1]\}$ be a set of co-optimal-constant algorithms and let u be a utility function mapping pairs (S, ϕ) of sequences and foldings to $\mathbb{R}$. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho : S \mapsto u(S, A_\rho(S)) \mid \rho \in [0,1]\}$ mapping sequences S to $\mathbb{R}$. The pseudo-dimension of $\mathcal{U}$ is $O(\log n)$.*

Let $[0,1]$ be the range of the utility function $u_\rho : \Sigma^n \to [0,1]$. Theorems 2.1.3 and 5.2.1 imply that with probability $1-\delta$ over the draw of $N = \Theta\left(\frac{1}{\epsilon^2}\left(\ln n + \ln \frac{1}{\delta}\right)\right)$ samples $\mathcal{S} \sim \mathcal{D}^N$, for all parameters $\rho \in [0,1]$, the difference between the average value of u_ρ over the samples and the expected value of u_ρ is at most ϵ: $\left|\frac{1}{N}\sum_{S\in\mathcal{S}} u_\rho(S) - \mathbb{E}_{S\sim\mathcal{D}}\left[u_\rho(S)\right]\right| \leq \epsilon$.

5.3 Predicting topologically associating domains

Inside a cell, the linear DNA of the genome wraps into three-dimensional structures that influence genome function. Some regions of the genome are closer than others and thereby interact more. One important structure is called a *topological associating domain (TAD)*, which is a contiguous segment of the genome that folds into a compact region. Formally, given a DNA sequence of length n, a TAD set T is a set of non-overlapping intervals from the set $\{1, \ldots, n\}$. If an interval $[a, b]$ is in the TAD set T, then the bases within the corresponding substring physically interact more frequently among one another than with bases from the rest of the genome. Disrupting TAD boundaries can affect the expression of nearby genes, which can trigger diseases such as congenital malformations and cancer [134].

Biological experiments facilitate predicting the location of TADs by measuring the contact frequency of any two locations in the genome [127]. TAD prediction algorithms use these contact frequency measurements to identify regions along the genome that are frequently in contact.

More formally, given the sequence or genome length $n \in \mathbb{N}$, a TAD set is a set

$$T = \{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\} \subset [n] \times [n]$$

such that $i_1 < j_1 < i_2 < j_2 < \cdots < i_t < j_t$. If $(i,j) \in T$, then the region of the genome between the indices i and j is thought to interact frequently.

A TAD prediction algorithm algorithm takes as input a weighted adjacency matrix $M \in \mathbb{R}^{n\times n}$ that measures contact frequency. The goal of TAD-finding is to compute the TAD set T that maximizes the objective function

$$\sum_{(i,j)\in T} s_\rho(i,j) - \mu_\rho(j-i), \tag{5.3}$$

where $\rho \geq 0$ is a tunable parameter,

$$s_\rho(i,j) = \frac{1}{(j-i)^\rho} \sum_{i \leq p < q \leq j} M_{pq}$$

is the scaled density of the subgraph induced by the interactions between genomic loci i and j, and

$$\mu_\rho(d) = \frac{1}{n-d} \sum_{t=0}^{n-d-1} s_\rho(t, t+d).$$

is the mean value of s_ρ over all sub-matrices of length d along the diagonal of M.

We use the notation $\mathcal{T}_\rho(M)$ to denote the set of TAD sets maximizing Equation (5.3). For each parameter ρ, we can run a dynamic programming algorithm A_ρ which returns a labeling $A_\rho(M)$ in $\mathcal{T}_\rho(M)$. As we vary the weight, this gives rise to a family of algorithms. Since there is no

consensus about what the best parameter is, our goal is to automatically learn the best parameter. As before, we assume that the domain expert has a utility function that characterizes the quality of a TAD set T, denoted $u(M, T) \in [0, 1]$. We are again agnostic to the specific definition of u, but as a concrete example, $u(M, T)$ might measure the fraction of TADs in T that are in the correct location given a "ground truth" TAD set T^*. In this case, the learning algorithm would require access to the ground truth TAD set—which may be hand curated—for every matrix M in the training set. We prove that for any fixed matrix M, utility as a function of the parameter ρ is piecewise-constant with $O\left(n^2 4^{n^2}\right)$ pieces, which implies the following pseudo-dimension bound.

Theorem 5.3.1. *Let $\left\{A_\rho \mid \rho \in \mathbb{R}_{\geq 0}\right\}$ be a set of co-optimal-constant algorithms and let u be a utility function mapping pairs (M, T) of matrices and TAD sets to $\mathbb{R}$. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \left\{u_\rho : M \mapsto u\left(M, A_\rho\left(M\right)\right) \mid \rho \in \mathbb{R}_{\geq 0}\right\}$ mapping matrices $M \in \mathbb{R}^{n \times n}$ to $\mathbb{R}$. The pseudo-dimension of $\mathcal{U}$ is $O(n^2)$.*

This theorem together with Theorem 2.1.3 implies that $N = \Theta\left(\frac{1}{\epsilon^2}\left(n^2 + \ln \frac{1}{\delta}\right)\right)$ samples are sufficient to ensure that with probability $1 - \delta$, for all parameters $\rho \in [0, 1]$, the difference between the average value of u_ρ over the samples and the expected value of u_ρ is at most ϵ.

Chapter 6

Mechanism design

In this chapter, we study parameter tuning in the context of mechanism design. In economics, a mechanism is a tool that helps a set of rational agents come to a collective decision. For example, given a set of items and the agents' reported values for those items, a mechanism might determine an allocation of the items to the agents. In order to ensure the agents do not act strategically, and thus report their values truthfully, mechanisms often require agents to pay some amount of money for the items they receive. Given the right payment scheme, one can ensure that the agents are always incentivized to report truthfully. This type of mechanism is known as *incentive compatible*.

Mechanisms can be designed with many different goals in mind. For example, one might wish to design an incentive compatible mechanism with high revenue (the sum of the agents' payments), profit (the revenue minus the cost of producing the items), or social welfare (the sum of the agents' values for the items they receive). As in the case of algorithm configuration, there are myriad different mechanism families, each defined by tunable parameters, and different parameter settings will lead to differing profit, revenue, and social welfare.

In this chapter, we analyze automated parameter tuning under a model that is nearly identical to the previous few chapters of this proposal. There is an unknown distribution over agents' values for a set of values. The mechanism designer receives a training set of values sampled from this distribution. His goal is to use this training set to select mechanism parameters with high expected profit on the underlying distribution. We analyze the sample complexity of this problem in Section 6.1.

Next, in Section 6.2, we analyze a more general mechanism design problem where the agents' values are over an arbitrary set of outcomes. For example, these mechanisms can help agents come to a collective decision about whether or not to build a public good, such as a bridge. In this setting, we provide sample complexity bounds for social welfare maximization.

6.1 Profit maximization

One of the most tantalizing and long-standing open problems in mechanism design is profit maximization in multi-item, multi-buyer settings. Much of the literature surrounding this problem rests on the strong assumption that the mechanism designer knows the distribution over buyers' values. In reality, this information is rarely available. The support of the distribution alone is often doubly exponential, so obtaining and storing the distribution is impractical.

We relax this assumption and instead assume that the mechanism designer only has a set of samples from the distribution [128, 129, 173]. In this work, we develop learning-theoretic foundations of sample-based mechanism design. In particular, we provide *generalization guarantees* which bound the difference between the empirical profit of a mechanism over a set of samples and its expected profit on the unknown distribution.

A substantial body of theory on sample-based mechanism design has developed recently, primarily in single-parameter settings [6, 40, 48, 52, 65, 73, 87, 94, 100, 142, 145, 163]. In this

section, we present a general theory for deriving worst-case generalization guarantees in multi-item settings, as well as data-dependent guarantees when the distribution over buyers' values is well-behaved. We analyze mechanism classes that have not yet been studied in the sample-based mechanism design literature and match or improve over the best-known guarantees for many of the special classes that have been studied.

6.1.1 Our contributions

Our contributions come in three interrelated parts. The results in this section are joint work with Nina Balcan and Tuomas Sandholm [24]. Our existing results appeared in EC 2018.

Worst-case generalization guarantees for profit maximization. We uncover a key structural property shared by a variety of mechanisms which allows us to prove strong generalization guarantees: for any fixed set of bids, profit is a piecewise linear function of the mechanism's parameters. Our main theorem provides generalization guarantees for any class exhibiting this structure. To prove this theorem, we relate the complexity of the partition splitting the parameter space into linear portions to the intrinsic complexity of the mechanism class, which we quantify using *pseudo-dimension*. In turn, pseudo-dimension bounds imply generalization bounds. We prove that many mechanisms throughout economics, artificial intelligence, and theoretical computer science share this structure, and thus our main theorem yields strong learnability guarantees.

We prove that our main theorem applies to randomized mechanisms, making us the first to provide generalization bounds for these mechanisms. Our guarantees apply to lotteries, a general representation of randomized mechanisms, which are extremely important in the intersection of economics and computation (e.g., [16, 39, 47]). Randomized mechanisms are known to generate higher expected revenue than deterministic mechanisms in many settings (e.g., [55, 68]). Our results imply, for example, that if the mechanism designer plans to offer a menu of ℓ lotteries over m items to an additive or unit-demand buyer, then $\tilde{O}\left(H^2\ell m/\epsilon^2\right)$ samples are sufficient to ensure that every menu's expected profit is ϵ-close to its empirical profit, where H is the maximum profit achievable over the support of the buyer's valuation distribution.

We also provide guarantees for pricing mechanisms using our main theorem. These include *item-pricing mechanisms*, also known as *posted-price mechanisms*, where each item has a price and buyers buy their utility-maximizing bundles. These mechanisms are prevalent throughout economics and computation (e.g., [15, 42, 75]). Additionally, we study *multi-part tariffs*, where there is an upfront fee and a price per unit. We are the first to provide generalization bounds for these tariffs and other non-linear pricing mechanisms, which have been studied in economics for decades (e.g., [76, 151, 189]). For instance, our main theorem guarantees that if there are κ units of a single good for sale, then $\tilde{O}\left(H^2\kappa/\epsilon^2\right)$ samples are sufficient to learn a nearly optimal two-part tariff. See Figure 6.1 for an illustration of the partition of the two-part tariff parameter space into piecewise-linear portions.

Our main theorem implies generalization bounds for many auction classes, such as *second price auctions*, which are fundamentally important in economics and beyond (e.g., [44, 62, 184]). We also study generalized VCG auctions, such as *affine maximizer auctions, virtual valuations combinatorial auctions*, and *mixed-bundling auctions*, which have been studied in AI and economics (e.g., [69, 102, 122, 161, 173]).

Next, we combine our analysis with tools from the structured prediction literature in theoretical machine learning. In doing so, we provide more refined upper bounds for several "simple" mechanism classes and answer an open question posed by Morgenstern and Roughgarden [146].

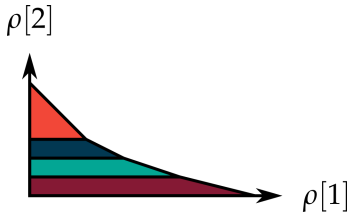


Figure 6.1: This figure illustrates a partition of the *two-part tariff* parameter space into piecewise-linear portions. Under a two-part tariff there are multiple units of a single good for sale. The seller sets an entry fee $\rho[1]$ and a price $\rho[2]$ per unit. If a buyer wishes to buy $t \geq 1$ units, she pays $\rho[1] + \rho[2] \cdot t$ and otherwise she pays nothing. See Example 6.1.1 for more details.

Data-dependent generalization guarantees for profit maximization. We provide several data-dependent tools that strengthen our main theorem when the distribution over buyers' values is "well-behaved." First, we prove generalization guarantees that, surprisingly, are independent of the number of items for item-pricing mechanisms, second price auctions with reserves, and a subset of lottery mechanisms. Under anonymous prices, our bounds do not depend on the number of bidders either. These guarantees hold when the bidders are additive with values drawn from *item-independent distributions* (bidder i_1's value for item j is independent from her value for item j', but her value for item j may be arbitrarily correlated with bidder i_2's value for item j).

Bidders with item-independent value distributions have been studied extensively in prior research (e.g., [16, 41, 42, 46, 85, 93, 192]). Cai and Daskalakis [41] provide learning algorithms for bidders with valuations drawn from product distributions, which are item-independent. Their algorithms return mechanisms whose expected revenue is a constant fraction of the optimal revenue obtainable by any randomized and Bayesian truthful mechanism. Relying on prior research [85, 146], their sample complexity guarantee when the buyers are additive is $O\left(\left(\frac{H}{\epsilon}\right)^2\left(nm\log(nm)+\log\frac{1}{\delta}\right)\right)$, where n is the number of buyers. We improve this sample complexity bound to $O\left(\left(\frac{H}{\epsilon}\right)^2\left(n\log n+\log\frac{1}{\delta}\right)\right)$, completely removing the dependence on the number of items.

Structural profit maximization. Many of the mechanism classes we study exhibit a hierarchical structure. For example, when designing a pricing mechanism, the designer can segment the population into k groups and charge each group a different price. This is prevalent throughout daily life: movie theaters and amusement parks have different admission prices per market segment, with groups such as Child, Student, Adult, and Senior Citizen. In the simplest case, $k = 1$ and the prices are anonymous. If k equals the number of buyers, the prices are non-anonymous, thus forming a hierarchy of mechanisms. In general, the designer should not choose the simplest class to optimize over simply to guarantee good generalization because more complex classes are more likely to contain nearly optimal mechanisms. We show how the mechanism designer can determine the precise level in the hierarchy assuring him the optimal tradeoff between profit maximization and generalization.

6.1.2 Related research

Sample-based mechanism design was introduced in the context of *automated mechanism design* (AMD). In AMD, the goal is to design algorithms that take as input information about a set of

buyers and return a mechanism that maximizes an objective such as revenue [54, 57, 170]. The input information about the buyers in early AMD was an explicit description of the distribution over their valuations. The support of the distribution's prior is often doubly exponential, for example in combinatorial auctions, so obtaining and storing the distribution is impractical. In response, sample-based mechanism design was introduced where the input is a set of samples from this distribution [128, 129, 173]. Those papers also introduced the idea of searching for a high-revenue mechanism in a parameterized space where any parameter vector yields a mechanism that satisfies the individual rationality and incentive-compatibility constraints. This was in contrast to the traditional, less scalable approach of representing mechanism design as an unrestricted optimization problem where those constraints need to be explicitly modeled. The parameterized work studied algorithms for designing combinatorial auctions with high empirical revenue. We follow the parameterized approach, but we study generalization guarantees, which they did not address.

Prior work on the sample complexity of profit maximization has primarily concentrated on the single-item setting, with the exception of work by Balcan et al. [19], Cai and Daskalakis [41], Medina and Vassilvitskii [138], Morgenstern and Roughgarden [146], Syrgkanis [178], and Gonczarowski and Weinberg [88]. We provide a detailed comparison of our results to these papers on multi-item mechanisms in our EC 2018 paper [24]. Earlier work of Balcan et al. [18] addressed sample complexity for revenue maximization in unrestricted supply settings. From an algorithmic perspective, Devanur et al. [65], Hartline and Taggart [94], and Gonczarowski and Nisan [87] provide computationally efficient algorithms for learning nearly-optimal single-item auctions in various settings.

6.1.3 Preliminaries and notation

We study the problem of selling m heterogeneous goods to n buyers. We denote a bundle of goods as a quantity vector $\boldsymbol{q} \in \mathbb{Z}^m_{\geq 0}$. The number of units of item i in the bundle represented by $\boldsymbol{q}$ is denoted by its i^{th} component $q[i]$. Accordingly, the bundle consisting of only one copy of the i^{th} item is denoted by the standard basis vector $\boldsymbol{e}_i$, where $e_i[i] = 1$ and $e_i[j] = 0$ for all $j \neq i$. Each buyer $j \in [n]$ has a valuation function v_j over bundles of goods. If one bundle $\boldsymbol{q}_0$ is contained within another bundle $\boldsymbol{q}_1$ (i.e., $q_0[i] \leq q_1[i]$ for all $i \in [m]$), then $v_j(\boldsymbol{q}_0) \leq v_j(\boldsymbol{q}_1)$ and $v_j(\boldsymbol{0}) = 0$. We denote an allocation as $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$ where $\boldsymbol{q}_j$ is the bundle of goods that buyer j receives under allocation Q. The cost to produce the bundle $\boldsymbol{q}$ is denoted as $c(\boldsymbol{q})$ and the cost to produce the allocation Q is $c(Q) = \sum_{i=1}^{n} c(\boldsymbol{q}_i)$. Suppose there are κ_i units available of item i. Let $K = \prod_{i=1}^{m} \kappa_i$. We use $\boldsymbol{v}_j = \left(v_j(\boldsymbol{q}_1), \ldots, v_j(\boldsymbol{q}_K)\right)$ to denote buyer j's values for all of the K bundles and we use $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$ to denote a vector of buyer values. We also study additive buyers $\left(v_j(\boldsymbol{q}) = \sum_{i=1}^{m} q[i] v_j(\boldsymbol{e}_i)\right)$ and unit-demand buyers $\left(v_j(\boldsymbol{q}) = \max_{i:q[i]\geq 1} v_j(\boldsymbol{e}_i)\right)$. Every auction in the classes we study is incentive compatible, so we assume that the bids equal the bidders' valuations.

There is an unknown distribution $\mathcal{D}$ over buyers' values. We make very few assumptions about this distribution. First of all, we do not assume the distribution belongs to a parametric family. Moreover, we do not assume the buyers' values are independently or identically distributed. In particular, a bidder's values for multiple bundles may be correlated, and multiple bidders may have correlated values as well. We denote the support of $\mathcal{D}$ using the notation $\mathcal{X}$. Therefore, $\mathcal{X}$ is a set of buyers' values.

Our results apply to mechanisms that are parameterized by a vector $\boldsymbol{\rho} \in \mathbb{R}^d$, where the value of d depends on the mechanism class. For example, $\boldsymbol{\rho}$ might equal the prices of the items for

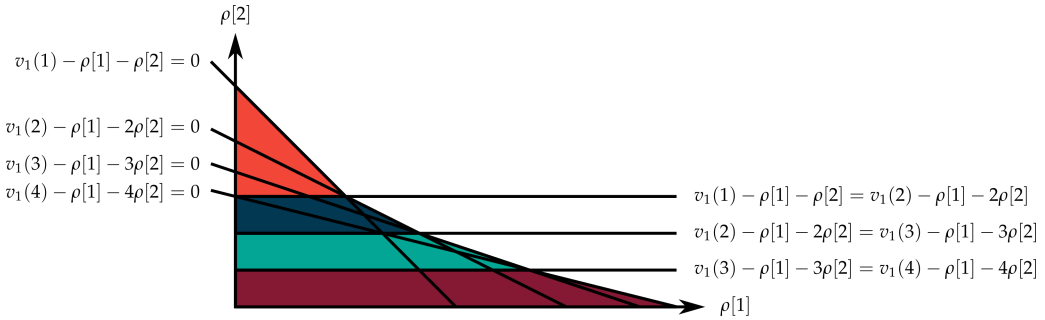


Figure 6.2: This figure illustrates the partition of the two-part tariff parameter space into piecewise-linear portions in the following scenario: there is one buyer whose value for one unit is $v_1(1) = 6$, value for two units is $v_1(2) = 9$, value for three units is $v_1(3) = 11$, and value for $i \geq 4$ units is $v_1(i) = 12$. We assume the seller produces at most four units. The buyer will buy exactly one unit if $v_1(1) - \rho[1] - \rho[2] > v_1(i) - \rho[1] - i \cdot \rho[2]$ for all $i \in \{2, 3, 4\}$ and $v_1(1) - \rho[1] - \rho[2] > 0$. This region of the parameter space is colored orange. By similar logic, the buyer will buy exactly two units in the blue region, exactly three units in the green region, and exactly four units in the red region.

sale. We use the notation $u_{\boldsymbol{\rho}}(\boldsymbol{v})$ to denote the profit of the mechanism parameterized by $\boldsymbol{\rho}$ on the valuation vector $\boldsymbol{v}$.

6.1.4 Worst-case generalization guarantees

Our guarantees apply to mechanism classes where for every valuation vector $\boldsymbol{v}$, profit as a function of the parameters $\boldsymbol{\rho}$ is piecewise linear. We denote this function as $u^*_{\boldsymbol{v}}(\boldsymbol{\rho}) : \mathbb{R}^d \to \mathbb{R}$, where $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = u^*_{\boldsymbol{v}}(\boldsymbol{\rho})$. We begin by illustrating this property via several simple examples.

Example 6.1.1 (Two-part tariffs)**.** In a *two-part tariff*, there are multiple units of a single good for sale. The mechanism is defined by a vector $\boldsymbol{\rho} \in \mathbb{R}^2$. The seller sets an *upfront fee* $\rho[1]$ and a *price per unit* $\rho[2]$. Here, we consider the simple case where there is a single buyer[1]. If the buyer wishes to buy $t \geq 1$ units, she pays the upfront fee $\rho[1]$ plus $\rho[2] \cdot t$, and if she does not want to buy anything, she does not pay anything. Two-part tariffs have been studied extensively by economists [76, 151, 189] and are prevalent throughout daily life. For example, gym and golf membership programs often require an upfront membership fee plus a fee per month. In many cities, purchasing a public transportation card requires a small upfront fee and an additional cost per ride. Many coffee machines, such as those made by Keurig and Nespresso, require specialty coffee pods. Purchasing these pods amounts to paying a fee per unit on top of the upfront fee, which is the cost of the coffee machine.

Suppose there are κ units of the good for sale. The buyer will buy exactly $t \in \{1, \ldots, \kappa\}$ units so long as $v_1(t) - (\rho[1] + \rho[2] \cdot t) > v_1(t') - (\rho[1] + \rho[2] \cdot t')$ for all $t' \neq t$ and $v_1(t) - (\rho[1] + \rho[2] \cdot t) > 0$. Therefore, for a fixed set of buyer values, there are $\binom{\kappa+1}{2}$ hyperplanes splitting $\mathbb{R}^2$ into convex regions such that within any one region, the number of units bought does not vary. So long as the number of units bought is invariant, profit is a linear function of $\rho[1]$ and $\rho[2]$. See Figure 6.2 for an illustration.

Example 6.1.2 (Lotteries)**.** A lottery is defined by a vector $\boldsymbol{\rho} \in \mathbb{R}^{m+1}$, where for $i \in [m]$, $\rho[i]$ is the probability that the bidder receives item i, and $\rho[m+1]$ is the lottery's price. In this example, we

[1]We generalize to multiple buyers in Section 6.1.4.

consider the simple case where there is one additive buyer, no supplier cost, and a single lottery for sale. The buyer's expected utility for the lottery is $\boldsymbol{v} \cdot (\rho[1], \ldots, \rho[m]) - \rho[m+1]$. We know that the buyer will choose to buy the lottery so long as this value is at least 0. If the buyer buys the lottery, she will pay a price of $\rho[m+1]$, and otherwise she will pay nothing. Therefore, there is a single hyperplane breaking the parameter space into regions where profit is linear.

We provide generalization guarantees that are closely dependent on the "complexity" of the partition splitting $\mathbb{R}^d$ into regions such that $u_{\boldsymbol{v}}^*(\boldsymbol{\rho})$ is linear. Inspired by structure exhibited by many mechanism classes, such as Examples 6.1.1 and 6.1.2, we require that this partition be defined by a finite number of hyperplanes. We give the following name to this type of mechanism class:

Definition 6.1.3 ((d,t)-delineable)**.** Fix a set of mechanisms defined parameters $\mathcal{P} \subseteq \mathbb{R}^d$ and let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\right\}$ be the corresponding set of profit functions. The set $\mathcal{U}$ is (d,t)*-delineable* if for any valuation vector $\boldsymbol{v} \in \mathcal{X}$, there is a set $\mathcal{H}$ of t hyperplanes such that for any connected component $\mathcal{P}'$ of $\mathcal{P} \setminus \mathcal{H}$, the function $u_{\boldsymbol{v}}^*(\boldsymbol{\rho})$ is linear over $\mathcal{P}'$.

In Theorem 6.1.4, we relate pseudo-dimension to delineability.

Theorem 6.1.4. *If $\mathcal{U}$ is (d,t)-delineable, the pseudo dimension of $\mathcal{U}$ is $O(d \log(dt))$.*

We now prove that a variety of mechanism classes exhibit this delineability structure, and thus we can apply Theorem 6.1.4. We warm up with the classes from Examples 6.1.1 and 6.1.2.

Theorem 6.1.5. *Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^2\right\}$ be the set of profit functions corresponding to the class of two-part tariffs over a single buyer and κ units of a single good. The set $\mathcal{U}$ is $\left(2, \binom{\kappa+1}{2}\right)$-delineable.*

Theorem 6.1.6. *Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^{m+1}\right\}$ be the set of profit functions corresponding to the class of lottery mechanisms over m items and one additive bidder. Suppose that the cost to produce each item is zero. The set $\mathcal{U}$ is $(m+1,1)$-delineable.*

Lotteries

We now apply Theorem 6.1.4 to *lottery menus*. A *length-ℓ lottery menu* is a set

$$M = \left\{\boldsymbol{\rho}^{(0)}, \boldsymbol{\rho}^{(1)}, \ldots, \boldsymbol{\rho}^{(\ell)}\right\} \subset \mathbb{R}^{m+1},$$

where $\boldsymbol{\rho}^{(0)} = \mathbf{0}$. As is typical in the lottery literature, we assume there is a single additive buyer with values $\boldsymbol{v} \in \mathbb{R}^m$; our results easily generalize to unit-demand buyers and multiple buyers. Under the lottery defined by the parameters $\boldsymbol{\rho}^{(j)}$ the buyer receives each item i with probability $\rho^{(j)}[i]$ and pays a price of $\rho^{(j)}[m+1]$. Their expected utility for this lottery is $\boldsymbol{v} \cdot \left(\rho^{(j)}[1], \ldots, \rho^{(j)}[m]\right) - \rho^{(j)}[m+1]$. Let $\boldsymbol{\rho}_{\boldsymbol{v}} \in M$ be the lottery that maximizes the buyer's expected utility. We use the notation $\boldsymbol{q} \sim \boldsymbol{\rho}_{\boldsymbol{v}}$ to denote a random allocation of the lottery defined by $\boldsymbol{\rho}_{\boldsymbol{v}}$. Specifically, for all $i \in [m]$, $q[i] = 1$ with probability $\rho_{\boldsymbol{v}}[i]$ and $q[i] = 0$ with probability $1 - \rho_{\boldsymbol{v}}[i]$. The expected profit of the mechanism is $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \rho_{\boldsymbol{v}} - \mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{\rho}_{\boldsymbol{v}}}[c(\boldsymbol{q})]]$. Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^{\ell(m+1)}\right\}$.

The key challenge in bounding $\text{Pdim}(\mathcal{U})$ is that $\mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{\rho}_{\boldsymbol{v}}}[c(\boldsymbol{q})]$ is not a piecewise linear function of the parameters $\boldsymbol{\rho}^{(0)}, \ldots, \boldsymbol{\rho}^{(\ell)}$. To overcome this challenge, rather than bounding $\text{Pdim}(\mathcal{U})$, we bound the pseudo-dimension of a related class $\tilde{\mathcal{U}}$. We then show that optimizing over $\tilde{\mathcal{U}}$ amounts to optimizing over $\mathcal{U}$ itself. To motivate the definition of $\tilde{\mathcal{U}}$, notice that if $\boldsymbol{w} \sim$

$U([0,1]^m)$, the probability that $w[j]$ is smaller than $\rho_v[j]$ is $\rho_v[j]$. Therefore, $\mathbb{E}_{\boldsymbol{q}\sim\boldsymbol{\rho}_v}[c(\boldsymbol{q})] = \mathbb{E}_{\boldsymbol{w}}\left[c\left(\sum_{j:w[j]<\rho_v[j]}\boldsymbol{e}_j\right)\right]$. Given lottery parameters $\boldsymbol{\rho}\in\mathbb{R}^{\ell(m+1)}$, we define $\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v},\boldsymbol{w}) := \rho_v[m+1] - c\left(\sum_{j:w[j]<\rho_v[j]}\boldsymbol{e}_j\right)$ and define $\tilde{\mathcal{U}} = \left\{\tilde{u}_{\boldsymbol{\rho}} : \boldsymbol{\rho}\in\mathbb{R}^{\ell(m+1)}\right\}$. The important insight is that the class $\tilde{\mathcal{U}}$ is delineable because for a fixed pair $(\boldsymbol{v},\boldsymbol{w})$, both the lottery the buyer chooses and the bundle $\sum_{j:w[j]<\rho_v[j]}\boldsymbol{e}_j$ are defined by a set of hyperplanes.

Theorem 6.1.7. *For additive and unit-demand buyers, $\tilde{\mathcal{U}}$ is $\left(\ell(m+1),(\ell+1)^2+m\ell\right)$-delineable.*

The following theorem guarantees that optimizing over $\tilde{\mathcal{U}}$ amounts to optimizing over $\mathcal{U}$ itself. It follows from the fact that for all $\boldsymbol{v}$ and parameters $\boldsymbol{\rho}\in\mathbb{R}^{\ell(m+1)}$, $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{w}}[\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v},\boldsymbol{w})]$.

Theorem 6.1.8. *With probability $1-\delta$ over the draw of N samples $\left(\boldsymbol{v}^{(1)},\boldsymbol{w}^{(1)}\right),\ldots,\left(\boldsymbol{v}^{(N)},\boldsymbol{w}^{(N)}\right)\sim\mathcal{D}\times U([0,1])^m$, for all lottery parameters $\boldsymbol{\rho}\in\mathbb{R}^{\ell(m+1)}$,*

$$\left|\frac{1}{N}\sum_{i=1}^{N}\tilde{u}_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(i)},\boldsymbol{w}^{(i)}\right) - \mathop{\mathbb{E}}_{\boldsymbol{v}\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right]\right| = O\left(H\sqrt{\frac{1}{N}\left(\mathrm{Pdim}\left(\tilde{\mathcal{U}}\right)+\log\frac{1}{\delta}\right)}\right).$$

Non-linear pricing mechanisms

Non-linear pricing mechanisms are specifically used to sell multiple units of each good. We assume that the cost function caps the total number of units of each item that the producer will supply. In other words, there is some cap κ_i per item i such that it costs more to produce κ_i units of item i than the buyers will pay. Formally, this means that there exists $(\kappa_1,\ldots,\kappa_m)\in\mathbb{R}^m$ such that for all $\boldsymbol{v}\in\mathcal{X}$ and all allocations $Q=(\boldsymbol{q}_1,\ldots,\boldsymbol{q}_n)$, if there exists an item i such that $\sum_{j=1}^{n} q_j[i] > \kappa_i$, then $\sum_{j=1}^{n} v_j(\boldsymbol{q}_j) - c(Q) < 0$.

Menus of two-part tariffs. Menus of two-part tariffs are a generalization of Example 6.1.1. The seller offers the buyers ℓ different two-part tariffs and each buyer chooses the tariff and number of units that maximizes his utility. If the prices are non-anonymous, then each buyer is presented with a different menu of two-part tariffs.

Theorem 6.1.9. *Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho}\in\mathbb{R}^{2\ell}\right\}$ be the set of profit functions corresponding to the class of anonymous length-ℓ menus of two-part tariffs. The set $\mathcal{U}$ is $\left(2\ell, O\left(n(\kappa\ell)^2\right)\right)$-delineable. When the prices are non-anonymous, the set is $\left(2n\ell, O\left(n(\kappa\ell)^2\right)\right)$-delineable.*

General non-linear pricing mechanisms. We study non-linear pricing under Wilson's *bundling interpretation* [189]: If the prices are anonymous, there is a price per quantity vector $\boldsymbol{q}$ denoted $\rho(\mathbf{q})$. Buyer j will purchase the bundle that maximizes $v_j(\mathbf{q}) - \rho(\mathbf{q})$. If the prices are non-anonymous, there is a price per quantity vector $\boldsymbol{q}$ and buyer $j\in[n]$ denoted $\rho_j(\mathbf{q})$.

Theorem 6.1.10. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous non-linear pricing mechanisms. Let $K=\prod_{i=1}^{m}(\kappa_i+1)$. The set $\mathcal{U}$ is $\left(K, nK^2\right)$-delineable. When the prices are non-anonymous, the set is $\left(nK, nK^2\right)$-delineable.*

Item-pricing mechanisms

We now describe the application of Theorem 6.1.4 to anonymous and non-anonymous item-pricing mechanisms. Under anonymous prices, the seller sets a price per item. Under non-anonymous prices, there is a buyer-specific price per item. We assume that there is some fixed but arbitrary ordering on the buyers such that the first buyer in the ordering arrives first and buys the bundle of goods that maximizes his utility, then the next buyer in the ordering arrives and buys the bundle of remaining goods that maximizes his utility, and so on.

Theorem 6.1.11. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous item-pricing mechanisms with anonymous prices and additive buyers. The set $\mathcal{U}$ is (m, m)-delineable. When the prices are non-anonymous, the set is is (nm, nm)-delineable.*

Auctions

We now present applications of Theorem 6.1.4 to auctions.

Second price item auctions with item reserves. These auctions are only strategy proof for additive bidders, so we restrict our attention to this setting. In the case of non-anonymous reserves, there is a price $\rho_j(\boldsymbol{e}_i)$ for each item i and each bidder j. The bidders submit bids on the items. For each item i, the highest bidder j wins the item if her bid is above $\rho_j(\boldsymbol{e}_i)$. She pays the maximum of the second highest bid and $\rho_j(\boldsymbol{e}_i)$. If the bidder with the highest bid bids below her reserve, the item goes unsold. In the case of anonymous reserves, $\rho_1(\boldsymbol{e}_i) = \rho_2(\boldsymbol{e}_i) = \cdots = \rho_n(\boldsymbol{e}_i)$ for each item i.

Theorem 6.1.12. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous second-price item auctions. The set $\mathcal{U}$ is (m, m)-delineable. If the prices are non-anonymous, the set is is (nm, m)-delineable.*

Mixed bundling auctions with reserve prices (MBARPs). MBARPs [179] are a variation on the VCG mechanism with item reserve prices, with an additional fixed boost to the social welfare of any allocation where some bidder receives the grand bundle. Recall that in a single-item VCG auction (i.e., second-price auction) with a reserve price, the item is only sold if the highest bidder's bid exceeds the reserve price, and the winner must pay the maximum of the second highest bid and the reserve price. To generalize this intuition to the multi-item case, we enlarge the set of agents to include the seller, whose valuation for a set of items is the set's reserve price. An MBARP gives an additional additive boost to the social welfare of any allocation where some bidder receives the grand bundle, and then runs the VCG mechanism over this enlarged set of bidders. The allocation is the boosted social welfare maximizer and the payments are the VCG payments on the boosted social welfare values. Importantly, the seller makes no payments, no matter her allocation.

Formally, MBARPs are defined by a parameter $\gamma \geq 0$ and m reserve prices $\rho(\boldsymbol{e}_1), \ldots, \rho(\boldsymbol{e}_m)$. Let λ be a function such that $\lambda(Q) = \gamma$ if some bidder receives the grand bundle under allocation Q and 0 otherwise. For an allocation Q, let $\boldsymbol{q}_Q$ be the items not allocated. Given a valuation vector $\boldsymbol{v}$, the MBARP allocation is

$$Q^* = (\boldsymbol{q}_1^*, \ldots, \boldsymbol{q}_n^*) = \text{argmax} \left\{ \sum_{j=1}^{n} v_j(\boldsymbol{q}_j) + \sum_{i:q_Q[i]=1} \rho(\boldsymbol{e}_i) + \lambda(Q) - c(Q) \right\}.$$

Using the notation

$$Q^{-j} = \left(\boldsymbol{q}_1^{-j}, \ldots, \boldsymbol{q}_n^{-j}\right) = \text{argmax} \left\{ \sum_{\ell \neq j} v_\ell \left(\boldsymbol{q}_\ell\right) + \sum_{i : q_Q[i]=1} \rho\left(\boldsymbol{e}_i\right) + \lambda\left(Q\right) - c\left(Q\right) \right\},$$

bidder j pays

$$\sum_{\ell \neq j} v_\ell \left(\boldsymbol{q}_\ell^{-j}\right) + \sum_{i : q_{Q^{-j}}[i]=1} \rho\left(\boldsymbol{e}_i\right) + \lambda\left(Q^{-j}\right) - c\left(Q^{-j}\right) - \sum_{\ell \neq j} v_\ell \left(\boldsymbol{q}_\ell^*\right) - \sum_{i : q_{Q^*}[i]=1} \rho\left(\boldsymbol{e}_i\right) - \lambda\left(Q^*\right) + c\left(Q^*\right).$$

Theorem 6.1.13. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of MBARPs. The set $\mathcal{U}$ is $\left(m+1, (n+1)2^{2m}\right)$-delineable.*

Affine maximizer auctions (AMAs). AMAs are an expressive mechanism class: Roberts [161] proved that AMAs are the only *ex post* truthful mechanisms over unrestricted value domains. Later, Lavi et al. [122] proved that under natural assumptions, every truthful multi-item auction is an "almost" AMA, that is, an AMA for sufficiently high values. An AMA is defined by a weight per bidder $w_j \in \mathbb{R}_{>0}$ and a boost per allocation $\lambda\left(Q\right) \in \mathbb{R}_{\geq 0}$. By increasing any w_j or $\lambda\left(Q\right)$, the seller can increase bidder j's bids or increase the likelihood that Q is the auction's allocation. The AMA allocation Q^* is the one which maximizes the weighted social welfare, i.e., $Q^* = (\boldsymbol{q}_1^*, \ldots, \boldsymbol{q}_n^*) = \text{argmax} \left\{ \sum_{j=1}^n w_j v_j \left(\boldsymbol{q}_j\right) + \lambda\left(Q\right) - c\left(Q\right) \right\}$. The payments have the same form as the VCG payments, with the parameters factored in to ensure truthfulness. Formally, using the notation $Q^{-j} = \left(\boldsymbol{q}_1^{-j}, \ldots, \boldsymbol{q}_n^{-j}\right) = \text{argmax} \left\{ \sum_{\ell \neq j} w_\ell v_\ell \left(\boldsymbol{q}_\ell\right) + \lambda\left(Q\right) - c\left(Q\right) \right\}$, each bidder j pays

$$\frac{1}{w_j} \left[\sum_{\ell \neq j} w_\ell v_\ell \left(\boldsymbol{q}_\ell^{-j}\right) + \lambda\left(Q^{-j}\right) - c\left(Q^{-j}\right) - \left(\sum_{\ell \neq j} w_\ell v_\ell \left(\boldsymbol{q}_\ell^*\right) + \lambda\left(Q^*\right) - c\left(Q^*\right) \right) \right].$$

Theorem 6.1.14. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of AMAs. The set $\mathcal{U}$ is $\left(O\left(n(n+1)^m\right), (n+1)^{2m+1}\right)$-delineable.*

Theorem 6.1.14 implies exponentially-many samples are sufficient to ensure that empirical and expected profit are close. Balcan et al. [19] prove an exponential number of samples is also necessary.

Virtual valuation combinational auctions (VVCAs) [128] are a special case of AMAs where each $\lambda\left(Q\right)$ is split into n terms such that $\lambda\left(Q\right) = \sum_{j=1}^n \lambda_j\left(Q\right)$ where $\lambda_j\left(Q\right) = c_{j,\boldsymbol{q}}$ for all allocations Q that give bidder j exactly bundle $\boldsymbol{q}$. We prove a similar theorem for VVCAs.

Theorem 6.1.15. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of VVCAs. The set $\mathcal{U}$ is $\left(O\left(n^2 2^m\right), (n+1)^{2m+1}\right)$-delineable.*

Finally, λ-auctions [102] are a special case of AMAs where the bidder weights equal 1.

Theorem 6.1.16. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of λ-auctions. The set $\mathcal{U}$ is $\left((n+1)^m, (n+1)^{2m+1}\right)$-delineable.*

We now study two hierarchies of AMAs. In Section 6.1.6, we show how to learn which level of the hierarchy optimizes the tradeoff between generalization and profit for the setting at hand.

$\mathcal{Q}$-boosted AMAs and λ-auctions. Let $\mathcal{Q}$ be a set of allocations. The set of $\mathcal{Q}$-boosted AMAs (resp., λ-auctions) consists of all AMAs (resp., λ-auctions) where only allocations in $\mathcal{Q}$ are boosted. In other words, if $\lambda(Q) > 0$, then $Q \in \mathcal{Q}$.

Theorem 6.1.17. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of $\mathcal{Q}$-boosted AMAs. The set $\mathcal{U}$ is $\left(O\left(n\left(n+|\mathcal{Q}|\right)\right),(n+1)^{2(m+1)}\right)$-delineable.*

6.1.5 Data-dependent generalization guarantees

In this section, we provide a data-dependent method of strengthening the results in Section 6.1.4 when the underlying distribution is "well-behaved." This technique applies to bidders whose values are drawn from item-independent distributions and mechanisms whose profit functions decompose additively. For example, under item-pricing mechanisms, the profit function decomposes into the profit obtained from selling item 1, plus the profit obtained by selling item 2, and so on. We obtain surprisingly strong guarantees in this setting: our bounds do not depend on the number of items and under anonymous prices, they do not depend on the number of bidders either.

To obtain our data-dependent guarantees, we move from pseudo-dimension to Rademacher complexity (Section 2.2), which allows us to prove distribution-dependent generalization guarantees. This is the key advantage of Rademacher complexity over pseudo-dimension; pseudo-dimension implies generalization guarantees that are worst-case over the distribution whereas Rademacher complexity implies distribution-dependent guarantees. We prove that this shift to Rademacher complexity from pseudo-dimension is in fact necessary in order to obtain guarantees that are independent of the number of items (Theorem 6.1.22).

In the following corollary of Theorem 6.1.4, we show that if the profit functions of a class $\mathcal{M}$ decompose additively into a number of simpler functions, then we can easily bound $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{M})$ using the Rademacher complexity of those simpler functions. We then demonstrate the power of this corollary by proving stronger guarantees for many well-studied mechanism classes when the bidders are additive and their valuations are drawn from item-independent distributions. This includes bidders with values drawn from product distributions as a special case, which have been extensively studied in the mechanism design literature (e.g., [16, 41, 42, 85, 93, 192]).

We say that a mechanism class parameterized by vectors $\boldsymbol{\rho} \in \mathbb{R}^d$ *decomposes additively* if for all $\boldsymbol{\rho} \in \mathbb{R}^d$, there exist T functions $u_{1,\boldsymbol{\rho}}, \ldots, u_{T,\boldsymbol{\rho}}$ such that the function $u_{\boldsymbol{\rho}}$ can be written as $u_{\boldsymbol{\rho}}(\cdot) = u_{1,\boldsymbol{\rho}}(\cdot) + \cdots + u_{T,\boldsymbol{\rho}}(\cdot)$.

Corollary 6.1.18. *Let $\mathcal{U}$ be the set of profit functions corresponding to a class of additively decomposable mechanisms parameterized by vectors $\boldsymbol{\rho} \in \mathbb{R}^d$. Let $\mathcal{U}_i$ equal the set $\mathcal{U}_i = \left\{u_{i,\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$. Suppose that for all $\boldsymbol{\rho} \in \mathbb{R}^d$, the range of $u_{i,\boldsymbol{\rho}}$ over the support of $\mathcal{D}$ is $[0, H_i]$ and that the class $\mathcal{U}_i$ is (d_i, t_i)-delineable. Then for any set of samples $\mathcal{S} \sim \mathcal{D}^N$,*

$$\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(\sum_{i=1}^{T} H_i \sqrt{\frac{d_i \ln(d_i t_i)}{N}}\right).$$

We now instantiate Corollary 6.1.18 for several mechanism classes.

Theorem 6.1.19. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of second-price auctions with anonymous reserves. Suppose the bidders are additive, $\mathcal{D}$ is item-independent, and the cost function is additive. For any set $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{1/N}\right)$. When the reserves are non-anonymous, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{n \log n / N}\right)$.*

The following theorem follows from the same logic as Theorem 6.1.19.

Theorem 6.1.20. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous item-pricing mechanisms. Suppose the bidders are additive, $\mathcal{D}$ is item-independent, and the cost function is additive. For any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{1/N}\right)$. When the prices are non-anonymous, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{n \log n/N}\right)$.*

Menus of item lotteries. A *length-ℓ item lottery menu* is a set of ℓ lotteries per item. The menu for item i is $M_i = \left\{\boldsymbol{\rho}_i^{(0)}, \boldsymbol{\rho}_i^{(1)}, \ldots, \boldsymbol{\rho}_i^{(\ell)}\right\} \subset \mathbb{R}^2$, where $\boldsymbol{\rho}_i^{(0)} = \mathbf{0}$. The buyer chooses one lottery $\boldsymbol{\rho}_i^{(j_i)}$ per menu M_i, receives each item i with probability $\rho_i^{(j_i)}[1]$, and pays $\sum_{i=1}^m \rho_i^{(j_i)}[2]$.

Theorem 6.1.21. *Let $\mathcal{U}$ be the set profit functions corresponding to the class of length-ℓ item lottery menus. If the bidder is additive, $\mathcal{D}$ is item-independent, and the cost function is additive, then for any set $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) \leq O\left(H\sqrt{\ell \log \ell/N}\right)$.*

Finally, we prove lower bounds showing that one could not hope to prove the generalization guarantees implied by Theorems 6.1.19 and 6.1.20 using pseudo-dimension alone.

Theorem 6.1.22. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous item-pricing mechanisms. The* $\operatorname{Pdim}(\mathcal{U}) \geq m$. *If the prices are non-anonymous, then* $\operatorname{Pdim}(\mathcal{U}) \geq nm$. *The same two lower bounds hold when $\mathcal{U}$ is the set of profit functions corresponding to the classes of anonymous and non-anonymous second-price auctions.*

6.1.6 Structural profit maximization

In this section, we use our results from Section 6.1.4 to provide tools for optimizing the *profit-generalization tradeoff*. Throughout this section, we will use the following notation: for a mechanism class $\mathcal{M}$, let $\mathcal{U}$ be the corresponding set of profit functions, and let $\epsilon_{\mathcal{M}}(N, \delta)$ be defined as $\epsilon_{\mathcal{M}}(N, \delta) = O\left(H\sqrt{\frac{1}{N}\left(\operatorname{Pdim}(\mathcal{U}) + \ln\frac{1}{\delta}\right)}\right)$. By Theorem 2.1.3, with probability $1-\delta$ over the draw of the set $\mathcal{S} \sim \mathcal{D}^N$, for all functions $u_{\boldsymbol{\rho}} \in \mathcal{U}$, $\left|\frac{1}{N}\sum_{\boldsymbol{v}\in\mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) - \mathbb{E}_{\boldsymbol{v}\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right]\right| \leq \epsilon_{\mathcal{M}}(N, \delta)$.

We begin by demonstrating this profit-generalization tradeoff pictorially. For the sake of illustration, suppose that $\mathcal{M}$ is a mechanism class that decomposes into a nested sequence of subclasses $\mathcal{M}_1 \subseteq \cdots \subseteq \mathcal{M}_t = \mathcal{M}$. For example, if $\mathcal{M}$ is the class of AMAs, then $\mathcal{M}_k$ could be the class of all $\mathcal{Q}$-boosted AMAs with $|\mathcal{Q}| = k$. Prior work [19] gave uniform convergence bounds for AMAs without taking advantage of the class's hierarchical structure. We illustrate[2] uniform convergence bounds in the left panel of Figure 6.3 with $t = 4$. On the x-axis, we illustrate the intrinsic complexity of the nested subclasses which can be measured using a metric such as pseudo-dimension. On the y-axis, for $i = 1, 2, 3, 4$, we illustrate the average profit over a fixed set of samples $\mathcal{S}$ of the mechanism $\hat{M}_i \in \mathcal{M}_i$ that maximizes average profit (the solid line). In particular, the dot on the solid line above $\mathcal{M}_i$ illustrates the average profit of $\hat{M}_i$. Since $\mathcal{M}_i \subseteq \mathcal{M}_j$ for $i \leq j$, the average profit of $\hat{M}_i$ is at least as high as the average profit of $\hat{M}_j$, which is why the solid line is increasing. Similarly, the dot on the dotted line above $\mathcal{M}_i$ illustrates the expected profit of $\hat{M}_i$. This dotted line begins decreasing when the complexity of the subclass grows to the point that overfitting occurs: a mechanism's average profit over the samples is no longer indicative of its expected profit on the unknown distribution. We also plot the lower bound on the expected profit of $\hat{M}_i$ which is equal to the average profit of $\hat{M}_i$ over the samples $\mathcal{S}$ minus

[2]These figures are purely illustrative; they are not based on a simulation or real data.

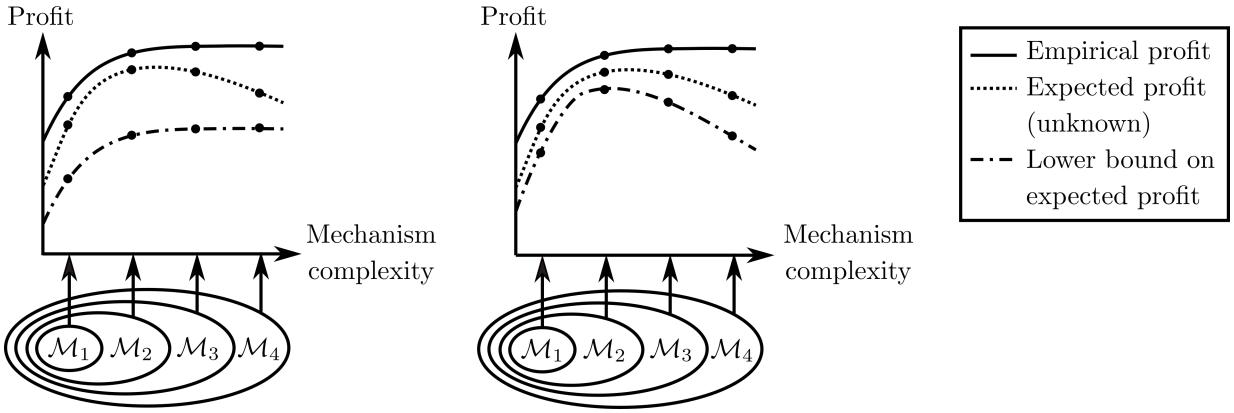


Figure 6.3: An illustration of uniform generalization guarantees (left panel) versus stronger complexity-dependent bounds (right panel) for a mechanism class $\mathcal{M}$ that decomposes into a nested sequence of subclasses $\mathcal{M}_1 \subseteq \mathcal{M}_2 \subseteq \mathcal{M}_3 \subseteq \mathcal{M}_4$. The x-axis is meant to measure each subclass's intrinsic complexity using a metric such as pseudo-dimension. Given a fixed set of samples $\mathcal{S}$, we plot the hypothetical average profit of the mechanism $\hat{M}_i \in \mathcal{M}_i$ that maximizes average profit over the samples. Specifically, the dot on the solid line above $\mathcal{M}_i$ illustrates the average profit of $\hat{M}_i$. Similarly, the dot on the dotted line above $\mathcal{M}_i$ illustrates the expected profit of $\hat{M}_i$. The dashed-dotted line on the left panel illustrates the lower bound on expected profit given by the uniform generalization guarantee, which equals the average profit of $\hat{M}_i$ minus $\epsilon_{\mathcal{M}}(N, \delta)$. Meanwhile, the dashed-dotted line on the right panel illustrates the lower bound on expected profit given by the complexity-dependent generalization guarantee, which equals the average profit of $\hat{M}_i$ minus $\epsilon_{\mathcal{M}_i}(N, \delta)$. We describe the figure in more detail in Section 6.1.6.

$\epsilon_{\mathcal{M}}(N, \delta)$. Since the average profit of $\hat{M}_i$ increases with i, this lower bound also increases with i, so the mechanism designer may erroneously think that $\hat{M}_4$ is the best mechanism to field.

Our general theorem allows us to be more careful since we can easily derive bounds $\epsilon_{\mathcal{M}_i}(N, \delta)$ for each class $\mathcal{M}_i$. Then, we can spread the confidence parameter δ across all subsets $\mathcal{M}_1, \ldots, \mathcal{M}_t$ using a weight function $w : \mathbb{N} \to [0, 1]$ such that $\sum w(i) \leq 1$. More formally, by a union bound, we are guaranteed that with probability at least $1 - \delta$, for all mechanisms $M \in \mathcal{M}$, the difference between the average profit of M over the samples and expected profit of M is at most $\min_{i:M \in \mathcal{M}_i} \epsilon_{\mathcal{M}_i}(N, \delta \cdot w(i))$. This is illustrated in the right panel of Figure 6.3, where for $i = 1, 2, 3, 4$, the lower bound on the expected profit of $\hat{M}_i$ is its average profit minus $\epsilon_{\mathcal{M}_i}(N, \delta \cdot w(i))$. This complexity-dependent lower bound indicates when overfitting begins. By maximizing this complexity-dependent lower bound on expected profit, the designer can correctly determine that $\hat{M}_2$ is a better mechanism to field than $\hat{M}_4$.

Both the decomposition of $\mathcal{M}$ into subsets and the choice of a weight function allow the designer to encode his prior knowledge about the market. For example, if mechanisms in $\mathcal{M}_i$ are likely more profitable than others, he can increase $w(i)$. The larger the weight $w(i)$ assigned to $\mathcal{M}_i$ is, the larger $\delta \cdot w(i)$ is, and a larger $\delta \cdot w(i)$ implies a smaller $\epsilon_{\mathcal{M}_i}(N, \delta \cdot w(i))$, thereby implying tighter guarantees.

We now present an application of this complexity-dependent analysis to item pricing. *Market segmentation* is prevalent throughout daily life: movie theaters, amusement parks, and tourist attractions have different admission prices per market segment, with groups such as Child, Student, Adult, and Senior Citizen. Formally, the designer can segment the buyers into k groups and charge each group a different price. If $k = 1$, the prices are anonymous and if $k = n$, they

are non-anonymous, thus forming a mechanism hierarchy. For $k \in [n]$, let $\mathcal{M}_k$ be the class of non-anonymous pricing mechanisms where there are k price groups. In other words, for all mechanisms in $\mathcal{M}_k$, there is a partition of the buyers $B_1, \ldots, B_k$ such that for all $t \in [k]$, all pairs of buyers $j, j' \in B_t$, and all items $i \in [m]$, $\rho_j(\boldsymbol{e}_i) = \rho_{j'}(\boldsymbol{e}_i)$. We derive the following guarantee for this hierarchy.

Theorem 6.1.23. *Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^{nm}\right\}$ be the set of profit functions corresponding to the class of non-anonymous item-pricing mechanisms over additive bidders. Let $w : [n] \to \mathbb{R}$ be a weight function such that $\sum_{i=1}^{n} w(i) \leq 1$. With probability at least $1 - \delta$ over the draw of a set $\mathcal{S} \sim \mathcal{D}^N$, for any $k \in [n]$ and any mechanism in $\mathcal{M}_k$ with parameters $\boldsymbol{\rho} \in \mathbb{R}^{nm}$,*

$$\left| \frac{1}{N} \sum_{\boldsymbol{v} \in \mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) - \mathop{\mathbb{E}}_{\boldsymbol{v} \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right] \right| = O\left(H \sqrt{\frac{1}{N}\left(km \log(nm) + \ln \frac{1}{\delta \cdot w(k)} \right)} \right).$$

We also prove the following theorem for the hierarchy of AMAs defined by the classes of $\mathcal{Q}$-boosted AMAs. For an AMA M, let $\mathcal{Q}_M$ be the set of all allocations Q such that $\lambda(Q) > 0$.

Theorem 6.1.24. *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of AMAs. Let w be a weight function that maps sets of allocations $\mathcal{Q}$ to $[0, 1]$ such that $\sum w(\mathcal{Q}) \leq 1$. With probability $1 - \delta$ over the draw of a set $\mathcal{S} \sim \mathcal{D}^N$, for any AMA M with parameters $\boldsymbol{\rho}$,*

$$\left| \frac{1}{N} \sum_{\boldsymbol{v} \in \mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) - \mathop{\mathbb{E}}_{\boldsymbol{v} \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right] \right| = O\left(H \sqrt{\frac{1}{N}\left(nm\left(n + |\mathcal{Q}_M|\right) \ln n + \ln \frac{1}{\delta \cdot w(\mathcal{Q}_M)} \right)} \right).$$

6.2 Social welfare maximization

A fundamental problem in economics is designing protocols that help groups of agents come to collective decisions. For example, the literature on partnership dissolution [59, 137] investigates questions such as: when a jointly-owned company must be dissolved, which partner should buy the others out, and for how much? When a couple divorces or children inherit an estate, how should they divide the property? How should a town decide which public projects to take on? There is no one policy that best answers these questions; the optimal protocol depends on the setting at hand. For example, splitting a family estate equally may seem "fair", but it may be impossible if the estate is not evenly divisible, and it may not be *efficient* if one family member values the estate much more than another.

In this section, we study an infinite, well-studied family of *mechanisms*, each of which takes as input a set of agents' stated values for each possible outcome and returns one of those outcomes. A mechanism can thus be thought of as an algorithm that the agents use to arrive at a single outcome. This family is known as the class of *neutral affine maximizers (NAMs)* [140, 147, 161]. There several appealing properties that NAMs satisfy. First, each mechanism in this infinite class is *incentive compatible*, which means that each agent is incentivized to report her values truthfully. In other words, she cannot gain by lying. In order to satisfy incentive compatibility, each agent may have to make a payment in addition to the benefit she accrues or loss she suffers from the mechanism's outcome. Otherwise, the agents could wildly misreport their valuations and suffer no consequences. This raises the question: who should receive this payment? Should it be split evenly among the agents? Should it be discarded? These questions motivate the second property that the class of NAMs satisfies: *budget balance*. A mechanism is budget-balanced if the aggregated payments are somehow distributed among the agents. A line of research [140, 147,

161] has shown that under natural assumptions, every incentive-compatible, budget-balanced mechanism is a NAM, roughly speaking.

We study a setting where there is a set $\{1, \ldots, m\}$ of m alternatives and a set of n agents. Each agent i has a value $v_i(j) \in \mathbb{R}$ for each alternative $j \in [m]$. We denote all m of his values as $\boldsymbol{v}_i \in \mathbb{R}^m$ and all n agents' values as $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \in \mathbb{R}^{nm}$.

A NAM takes as input a set of bids from each agent i. Since NAMs are incentive compatible, we assume the bids equal the agents true values $\boldsymbol{v}$. Every NAM is defined by a *social choice function* and a set of *payment functions*. A social choice function $f : \mathbb{R}^{nm} \to [m]$ uses the valuations to choose an alternative $f(\boldsymbol{v}) \in [m]$. Moreover, for each agent $i \in [n]$, there is a payment function $p_i : \mathbb{R}^{nm} \to \mathbb{R}$ which maps the values $\boldsymbol{v}$ to a value $p_i(\boldsymbol{v}) \in \mathbb{R}$ that agent i either pays or receives (if $p_i(\boldsymbol{v}) > 0$, then the agent pays that value, and if $p_i(\boldsymbol{v}) < 0$, then the agent receives that value).

A *neutral affine maximizer* mechanism [140, 147, 161], defined as follows, is incentive compatible and budget balanced (meaning that the sum of the agents' payments equals zero: $\sum_{i=1}^{n} p_i(\boldsymbol{v}) = 0$).

Definition 6.2.1 (Neutral affine maximizer with sink agents)**.** A *neutral affine maximizer* (NAM) mechanism is defined by n parameters (one per agent) $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n) \in \mathbb{R}^n_{\geq 0}$ such that at least one agent is assigned a weight of zero ($\{i : \rho_i = 0\} \neq \emptyset$). The social choice function is defined as $f_{\boldsymbol{\rho}}(\boldsymbol{v}) = \operatorname{argmax}_{j \in [m]} \sum_{i=1}^{n} \rho_i v_i(j)$. Let $j^* = f_{\boldsymbol{\rho}}(\boldsymbol{v})$ and for each agent i, let $j_{-i} = \operatorname{argmax}_{j \in [m]} \sum_{i' \neq i} \rho_{i'} v_{i'}(j)$. The payment function is defined as

$$p_i(\boldsymbol{v}) = \begin{cases} \frac{1}{\rho_i} \left(\sum_{i' \neq i} \rho_{i'} v_{i'}(j^*) - \sum_{i' \neq i}^{n} \rho_{i'} v_{i'}(j_{-i}) \right) & \text{if } \rho_i \neq 0 \\ -\sum_{i' \neq i} p_{i'}(\boldsymbol{v}) & \text{if } i = \min\{i' : \rho_{i'} = 0\} \\ 0 & \text{otherwise.} \end{cases}$$

Each agent i such that $\rho_i = 0$ is known as *sink agents* because his values do not influence the outcome.

Our high-level goal is to find a NAM that nearly maximizes the expected social welfare ($\sum_{i=1}^{n} v_i(j^*)$). The expectation is over the draw of a valuation vector $\boldsymbol{v} \sim \mathcal{D}$. Thus we define

$$u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^{n} v_i(j^*) \tag{6.1}$$

where $j^* = \operatorname{argmax}_{j \in [m]} \sum_{i=1}^{n} \rho_i v_i(j)$. We prove that for any fixed valuation vector $\boldsymbol{v}$, utility is a piecewise constant function of the parameters $\boldsymbol{\rho}$. This implies the following pseudo-dimension bound.

Theorem 6.2.2. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \left\{ u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}_{\geq 0}, \{\rho_i \mid i = 0\} \neq \emptyset \right\}$ where $u_{\boldsymbol{\rho}}$ is defined by Equation* (6.1)*. The pseudo-dimension of $\mathcal{U}$ is $O(n \ln(nm))$.*

We also prove that the pseudo-dimension of $\mathcal{U}$ is $\frac{n}{2}$, which means that our pseudo-dimension upper bound is tight up to log factors.

Theorem 6.2.3. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \left\{ u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}, \{\rho_i \mid i = 0\} \neq \emptyset \right\}$ where $u_{\boldsymbol{\rho}}$ is defined by Equation* (6.1)*. The pseudo-dimension of $\mathcal{U}$ is at least $\frac{n}{2}$.*

The results in this section are joint work with Nina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, and Tuomas Sandholm [25].

Chapter 7

A general theory of sample complexity for algorithm configuration

In this section, we present a general theory which unifies our results so far in this proposal. Throughout this proposal, we have analyzed algorithms parameterized by some set $\mathcal{P} \subseteq \mathbb{R}^d$ of vectors. We have bounded the pseudo-dimension of the class of functions $u_{\boldsymbol{\rho}} : \mathcal{Z} \to \mathbb{R}$ that measures, abstractly, the performance of the algorithm parameterized by $\boldsymbol{\rho} \in \mathcal{P}$ on input problem instances from a set $\mathcal{Z}$. In turn, classic results from learning theory allow us to use pseudo-dimension in order to derive sample complexity guarantees, as demonstrated in Chapter 2.

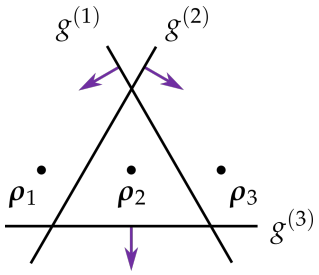
As we have seen in the previous chapters, the class $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\right\}$ is gnarly, so bounding its pseudo-dimension is not straight-forward. For example, in the case of integer programming algorithm configuration (Chapter 3), the domain of every function in $\mathcal{U}$ consists of integer programs, so it is unclear how to visualize or plot these functions, and there are no obvious notions of Lipschitzness or smoothness to rely on.

Rather than analyze the functions $u_{\boldsymbol{\rho}}$ directly, we have seen that it can be enlightening to analyze the algorithm's performance as a function of $\boldsymbol{\rho}$ on a fixed input z. We refer to this function as a *dual function*. The dual functions have a simple, Euclidean domain (namely, $\mathcal{P} \subseteq \mathbb{R}^d$), they are typically easy to visualize and plot, and they often have ample structure we can use to bound the intrinsic complexity of the class $\mathcal{U}$. For example, in most of the preceding chapters, we have seen that the dual functions are piecewise-constant or -linear. Broadly speaking, these dual functions are *piecewise-structured*. In this chapter, we define what it means for a dual function class to be piecewise-structured, and we provide pseudo-dimension guarantees for any algorithm family exhibiting this structure. Surprisingly, this abstraction does not introduce any slack: we are able to match all pseudo-dimension bounds proven thus far in this proposal.

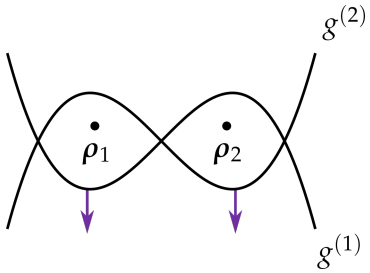
The results in this chapter are joint work with Nina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, and Tuomas Sandholm [25].

7.1 Problem statement

We assume there is an application-specific distribution $\mathcal{D}$ over problem instances in $\mathcal{Z}$. Our goal is to find a parameter vector in $\mathcal{P}$ with high performance in expectation over the distribution $\mathcal{D}$. As one step in this process, we analyze the number of samples necessary for *uniform convergence* to hold. Specifically, for any $\epsilon, \delta \in (0, 1)$ and any distribution $\mathcal{D}$ over problem instances, we bound the number N of samples sufficient to ensure that with probability at least $1 - \delta$ over the draw of N samples $\mathcal{S} = \{z_1, \ldots, z_N\} \sim \mathcal{D}^N$, uniformly for all parameters $\boldsymbol{\rho} \in \mathcal{P}$, the difference between the average utility of $\boldsymbol{\rho}$ and the expected utility of $\boldsymbol{\rho}$ is at most ϵ: $\left|\frac{1}{N}\sum_{i=1}^N u_{\boldsymbol{\rho}}(z_i) - \mathbb{E}_{z\sim\mathcal{D}}[u_{\boldsymbol{\rho}}(z)]\right| \leq \epsilon$. Classic results from learning theory guarantee that if uniform convergence holds and $\hat{\boldsymbol{\rho}}$ is a parameter vector that maximizes average utility over the samples $\left(\hat{\boldsymbol{\rho}} \in \operatorname{argmax}\left\{\frac{1}{N}\sum_{i=1}^N u_{\boldsymbol{\rho}}(z_i)\right\}\right)$, then $\hat{\boldsymbol{\rho}}$ is nearly optimal in expectation as well. In particular, with probability at least $1 - \delta$ over the draw $\mathcal{S} \sim \mathcal{D}^N$, $\max_{\rho\in\mathcal{P}} \mathbb{E}_{z\sim\mathcal{D}}[u_{\boldsymbol{\rho}}(z)] - \mathbb{E}_{z\sim\mathcal{D}}[u_{\hat{\boldsymbol{\rho}}}(z)] \leq 2\epsilon$.



(a) The arrows indicate on which side of each linear separator $g^{(i)}$ we have that $g^{(i)}(\boldsymbol{\rho}) = 0$ and on which side $g^{(i)}(\boldsymbol{\rho}) = 1$. For example, $g^{(1)}(\boldsymbol{\rho}_1) = 1$, $g^{(1)}(\boldsymbol{\rho}_2) = 1$, and $g^{(1)}(\boldsymbol{\rho}_3) = 0$.



(b) The arrows indicate on which side of each polynomial separator $g^{(i)}$ we have that $g^{(i)}(\boldsymbol{\rho}) = 0$ and on which side $g^{(i)}(\boldsymbol{\rho}) = 1$. For example, $g^{(1)}(\boldsymbol{\rho}_1) = 1$ and $g^{(1)}(\boldsymbol{\rho}_2) = 0$.

Figure 7.1: Figures 7.1a and 7.1b illustrate boundary functions partitioning $\mathbb{R}^2$.

7.2 Dual functions

Dual classes have been studied before in learning theory, beginning with research by Assouad [11], but never in the context of algorithm configuration. Below, we formally define the class of dual functions, adapted to our setting.

Definition 7.2.1 (Dual class)**.** The *dual class* of $\mathcal{U} = \{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\}$ is defined as

$$\mathcal{U}^* = \{u_z^* : \mathcal{U} \to \mathbb{R} \mid z \in \mathcal{Z}\}$$

where $u_z^*(u_{\boldsymbol{\rho}}) = u_{\boldsymbol{\rho}}(z)$. Each function $u_z^* \in \mathcal{U}^*$ fixes an input $z \in \mathcal{Z}$ and maps each function $u_{\boldsymbol{\rho}} \in \mathcal{U}$ to $u_{\boldsymbol{\rho}}(z)$. We refer to the class $\mathcal{U}$ as the *primal class.*

Each dual function u_z^* measures algorithmic performance as a function of the parameter vector $\boldsymbol{\rho}$ (or equivalently, as a function of $u_{\boldsymbol{\rho}}$) on the fixed input $z \in \mathcal{Z}$.

7.3 General theory of sample complexity for algorithm configuration

In this proposal, we show that a large number of algorithm configuration problems share a clear-cut, useful structure: for each problem instance $z \in \mathcal{Z}$, the function u_z^* is a piecewise structured. For example, each function u_z^* might be a piecewise-constant function of $\boldsymbol{\rho}$ with a small number of pieces. We use this piecewise structure of the dual class to bound the pseudo-dimension of the primal class $\mathcal{U}$. We then apply Theorem 2.1.3 to bound the number of samples sufficient to ensure that uniform convergence holds.

Before stating our main result, we introduce notation that we will use to more formally define the notion of piecewise-structured functions. Let $h : \mathcal{X} \to \mathbb{R}$ be a function mapping an abstract domain $\mathcal{X}$ to the real line. Intuitively, the function h is piecewise structured if we can partition the domain $\mathcal{X}$ into subsets $\mathcal{X}_1, \ldots, \mathcal{X}_m$ such that when we restrict h to a single piece $\mathcal{X}_i$, h equals some piece-specific function $f : \mathcal{X} \to \mathbb{R}$. In other words, for all $x \in \mathcal{X}_i$, $h(x) = f(x)$. We describe the partition $\mathcal{X}_1, \ldots, \mathcal{X}_m$ using a collection of *boundary functions* $g^{(1)}, \ldots, g^{(k)} : \mathcal{X} \to \{0, 1\}$. Each boundary function $g^{(i)}$ divides the domain $\mathcal{X}$ into two sets: the points it labels 0 and the points it labels 1. Figure 7.1 illustrates two partitions of $\mathbb{R}^2$ by boundary functions. Together, the k boundary functions partition the domain $\mathcal{X}$ into at most 2^k regions, each one corresponding to a bit vector $\boldsymbol{b} \in \{0, 1\}^k$ describing on which side of each boundary the region belongs. For each

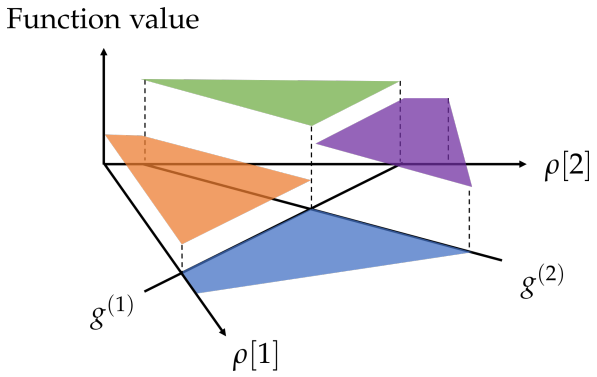


Figure 7.2: Example of a piecewise *structured* function. Here there are two functions that partition the space into 4 regions, and within each region the function value stays constant.

region, we specify a *piece function* $f_{\boldsymbol{b}} : \mathcal{X} \to \mathbb{R}$ defining the function values of h restricted to that region, where $\boldsymbol{b} \in \{0,1\}^k$ is the bit vector describing the region. More formally, the function h can be written as $h(x) = f_{\boldsymbol{b}_x}(x)$, where $\boldsymbol{b}_x = \left(g^{(1)}(x), \ldots, g^{(k)}(x)\right) \in \{0,1\}^k$ is the bit vector identifying the region in the partition containing the element x. Figure 7.2 shows an example of a piecewise-structured function with two boundary functions and four piece functions.

In many algorithm configuration problems, every function in the dual class is piecewise structured. Moreover, across dual functions, the corresponding boundary functions come from a single, fixed class, as do the piece functions. For example, the boundary functions might always be halfspace indicator functions, while the piece functions might always be linear functions. The following definition formalizes this structure.

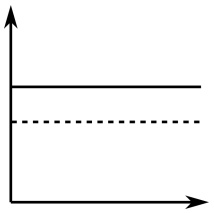
Definition 7.3.1 (($\mathcal{F}, \mathcal{G}, k$)-piecewise decomposable)**.** A class of functions $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ mapping some domain $\mathcal{X}$ to $\mathbb{R}$ is $(\mathcal{F}, \mathcal{G}, k)$-*piecewise decomposable* for a class $\mathcal{G} \subseteq \{0,1\}^{\mathcal{X}}$ of boundary functions and a class $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{X}}$ of piece functions if the following holds: for every function $h \in \mathcal{H}$, there exist k boundary functions $g^{(1)}, \ldots, g^{(k)} \in \mathcal{G}$ and a piece function $f_{\boldsymbol{b}} \in \mathcal{F}$ for each bit vector $\boldsymbol{b} \in \{0,1\}^k$ such that for all $x \in \mathcal{X}$, $h(x) = f_{\boldsymbol{b}_x}(x)$, where $\boldsymbol{b}_x = \left(g^{(1)}(x), \ldots, g^{(k)}(x)\right) \in \{0,1\}^k$.

Our main theorem shows that whenever a class $\mathcal{Q}$ of functions has a $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable dual function class $\mathcal{Q}^*$, we can bound the pseudo-dimension of $\mathcal{Q}$ in terms of the VC-dimension of $\mathcal{G}^*$ and the pseudo-dimension of $\mathcal{F}^*$. In other words, if the boundary and piece functions both have dual classes with low complexity, then the pseudo-dimension of $\mathcal{Q}$ is small. In Section 7.4, we show that for many common boundary and piece classes $\mathcal{F}$ and $\mathcal{G}$, we can easily bound the complexity of their dual classes.

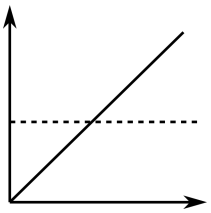
Theorem 7.3.2 (Main sample complexity theorem)**.** *Let $\mathcal{Q} \subseteq \mathbb{R}^{\mathcal{Y}}$ be a class of functions mapping an abstract domain $\mathcal{Y}$ to the real line. Suppose that the dual function class $\mathcal{Q}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-decomposable with boundary functions $\mathcal{G} \subseteq \{0,1\}^{\mathcal{Q}}$ and piece functions $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{Q}}$. Denote the VC-dimension of $\mathcal{G}^*$ as $d_{\mathcal{G}^*}$ and the pseudo-dimension of $\mathcal{F}^*$ as $d_{\mathcal{F}^*}$. The pseudo-dimension of $\mathcal{Q}$ is bounded as follows:*

$$\operatorname{Pdim}(\mathcal{Q}) \leq 4\left(d_{\mathcal{F}^*} + d_{\mathcal{G}^*}\right) \ln\left(4ek\left(d_{\mathcal{F}^*} + d_{\mathcal{G}^*}\right)\right).$$

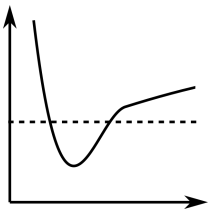
Relating these concepts back to algorithm configuration, $\mathcal{Q}$ equals the function class $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P}\}$, where every function in $\mathcal{Q}$ is defined by a set of parameters $\boldsymbol{\rho}$ and maps problem instances z to real-valued utilities $u_{\boldsymbol{\rho}}(z)$. Moreover, the dual class $\mathcal{Q}^*$ is equivalent to the function



(a) Constant function (zero oscillations).



(b) Linear function (one oscillation).



(c) Inverse-quadratic function (two oscillations).

Figure 7.3: Examples of functions with bounded oscillations. Each solid line is a function with bounded oscillations and each dotted line represents an arbitrary threshold.

class $\{u_z^* \mid z \in \mathcal{Z}\}$. Every function in $\mathcal{Q}^*$ is defined by a problem instance z and maps functions u_ρ (or equivalently, parameters $\boldsymbol{\rho}$) to utilities $u_z^*\left(u_\rho\right) = u_\rho(z)$.

7.4 Applications of main theorem to representative function classes

In this section, we instantiate our main result, Theorem 7.3.2, in settings inspired by algorithm configuration problems.

7.4.1 One-dimensional functions with a bounded number of oscillations

Let $\mathcal{U} = \left\{u_\rho \mid \rho \in \mathcal{P} \subseteq \mathbb{R}\right\}$ be a class of utility functions defined over a single-dimensional parameter space. We often find that the dual class contains functions that are piecewise constant, linear, or polynomial in the parameter. More generally, the functions in the dual class are piecewise-structured, and we can guarantee that the structured functions oscillate a fixed number of times. In the language of decomposability, this means that the dual function class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-decomposable, where the boundary functions $\mathcal{G} \subseteq \{0,1\}^{\mathcal{U}}$ are thresholds and the piece functions $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{U}}$ oscillate a bounded number of times, as we formalize below.

Definition 7.4.1. We say that a function $h : \mathbb{R} \to \mathbb{R}$ has at most B oscillations if for every $\theta \in \mathbb{R}$, the function $\rho \mapsto \mathbb{I}_{\{h(\rho) \geq \theta\}}$ is piecewise constant with at most B discontinuities.

For example, constant functions have zero oscillations (see Figure 7.3a), linear functions have one oscillation (see Figure 7.3b), and inverse-quadratic functions (of the form $h(x) = \frac{a}{x^2} + bx + c$) have at most two oscillations (see Figure 7.3c). Throughout this thesis, we analyze piecewise-structured functions whose the piece functions come from these three families. For example, in the case of integer programming, we analyzed dual functions that are piecewise constant (Chapter 3) and piecewise inverse-quadratic (Chapter 4). The computational biology algorithms (Chapter 5) and social-welfare maximizing mechanisms (Section 6.2) we analyzed have piecewise constant dual functions. The results in this section recover our guarantees from all of those settings.

In the following lemma, we bound the pseudo-dimension of classes with bounded oscillations.

Lemma 7.4.2. *Let $\mathcal{H}$ be a class of functions mapping $\mathbb{R}$ to $\mathbb{R}$, each of which has at most B oscillations. Then* $\operatorname{Pdim}(\mathcal{H}^*) = O(\ln B)$.

Lemma 7.4.2 implies the following pseudo-dimension bound for the case where the dual function class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-decomposable, where the boundary functions $\mathcal{G} \subseteq \{0,1\}^{\mathcal{U}}$ are thresholds and the piece functions $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{U}}$ oscillate a bounded number of times.

Corollary 7.4.3. *Let $\mathcal{U} = \{u_\rho \mid \rho \in \mathcal{P} \subseteq \mathbb{R}\}$ be a class of utility functions defined over a single-dimensional parameter space. Suppose the dual function class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-decomposable, where the boundary functions $\mathcal{G} = \{f_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ are thresholds $g_a : u_\rho \mapsto \mathbb{I}_{\{a \leq \rho\}}$. Moreover, suppose that for each function $f \in \mathcal{F}$, the function $\rho \mapsto f(u_\rho)$ has at most B oscillations. Then* $\text{Pdim}(\mathcal{U}) = O(\ln(B)\ln(k\ln(B)))$.

7.4.2 Multi-dimensional piecewise linear functions

Generalizing to multi-dimensional parameter spaces, we often find that the boundary functions correspond to halfspace thresholds and the piece functions correspond to constant or linear functions. We handle this case in the following lemma.

Lemma 7.4.4. *Let $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^d\}$ be a class of utility functions defined over a d-dimensional parameter space. Suppose the dual function class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-decomposable, where the boundary functions $\mathcal{G} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ are halfspace thresholds $g_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a}\cdot\boldsymbol{\rho} \leq \theta\}}$ and the piece functions $\mathcal{F} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \mathbb{R} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ are linear functions $f_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \boldsymbol{a} \cdot \boldsymbol{\rho} + \theta$. Then* $\text{Pdim}(\mathcal{U}) = O(d\ln(dk))$.

This lemma recovers our guarantees from Chapter 3, where we studied integer programming algorithms, and Section 6.1, where we studied revenue maximizing mechanisms.

Chapter 8

Data-dependent guarantees

In this chapter, we observe that for some configuration problems, the dual functions may not be piecewise-structured themselves (as in the previous chapter), can be closely approximated by "simple" functions, as in Figure 8.1. This raises the question: can we exploit this structure to provide strong generalization guarantees? We show that if the dual functions are approximated by simple functions under the L^∞-norm (meaning the maximum distance between the functions is small), then we can provide strong generalization guarantees. However, this is no longer true when the approximation only holds under the L^p-norm for $p < \infty$: we present a set of functions whose duals are well-approximated by a simple constant function under the L^p-norm, but which are not learnable.

We provide an algorithm that finds approximating simple functions in the following widely-applicable setting: the dual functions are piecewise-constant with a large number of pieces, but can be approximated by simpler piecewise-constant functions with few pieces, as in Figure 8.1.

In our experiments, we demonstrate significant practical implications of our analysis. We configure CPLEX, one of the most widely-used integer programming solvers. Integer programming has diverse applications throughout science. In Chapter 3, we showed that the dual functions associated with various CPLEX parameters are piecewise constant and provide generalization bounds that grow with the number of pieces. However, the number of pieces can be so large that these bounds can be quite loose. We show that these dual functions can be approximated under the L^∞-norm by simple functions (as in Figure 8.1), so our theoretical results imply strong generalization guarantees. In our experiments, we demonstrate that in order to obtain the same generalization bound, the training set size required under our analysis is up to 700 times smaller than that of Chapter 3. Improved sample complexity guarantees imply faster learning algorithms, since the learning algorithm needs to analyze fewer training instances.

The results in this section are joint work with Nina Balcan and Tuomas Sandholm and appeared in ICML 2020 [29].

8.1 Notation and background

8.1.1 Integer programming algorithm configuration

We use integer programming algorithm configuration as a running example, though our results are much more general. In this section, we briefly recall the integer programming algorithm configuration problem from Chapter 3. An *integer program (IP)* is defined by a matrix $A \in \mathbb{R}^{m \times n}$, a constraint vector $\boldsymbol{b} \in \mathbb{R}^m$, an objective vector $\boldsymbol{c} \in \mathbb{R}^n$, and a set of indices $I \subseteq [n]$. The goal is to find a vector $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{c} \cdot \boldsymbol{x}$ is maximized, subject to the constraints that $A\boldsymbol{x} \leq \boldsymbol{b}$ and for every index $i \in I$, $x[i] \in \{0, 1\}$.

In our experiments, we tune the parameters of branch-and-bound (B&B) [119], the most widely-used algorithm for solving IPs. It is used under the hood by commercial solvers such as CPLEX and Gurobi. We provide a brief, high-level overview of B&B, and refer the reader to

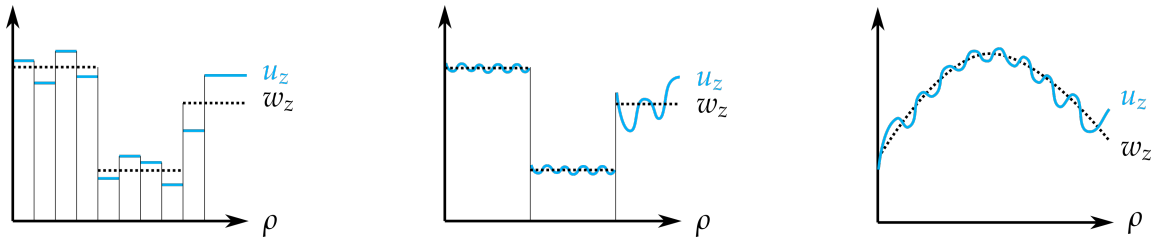


Figure 8.1: Examples of dual functions $u_z : \mathbb{R} \to \mathbb{R}$ (solid blue lines) which are approximated by simpler functions w_z (dotted black lines).

the textbook by Nemhauser and Wolsey [148] for more details. B&B builds a search tree to solve an input IP z. At the tree's root is the original IP z. At each round, B&B chooses a leaf of the search tree, which represents an IP z'. It does so using a *node selection policy*; common choices include depth- and best-first search. Then, it chooses an index $i \in I$ using a *variable selection policy*. It next *branches* on $x[i]$: it sets the left child of z' to be that same integer program z', but with the additional constraint that $x[i] = 0$, and it sets the right child of z' to be that same integer program, but with the additional constraint that $x[i] = 1$. It solves both LP relaxations, and if either solution satisfies the integrality constraints of the original IP z, it constitutes a feasible solution to z. B&B *fathoms* a leaf—which means that it never will branch on that leaf—if it can guarantee that the optimal solution does not lie along that path. B&B terminates when it has fathomed every leaf. At that point, we can guarantee that the best solution to z found so far is optimal. In our experiments, we tune the parameters of the variable selection policy, which we describe in more detail in Section 8.2.2.

In keeping with Chapter 3, we define $u_{\boldsymbol{\rho}}(z)$ to be 0 minus the size of the B&B tree CPLEX builds given the parameter setting $\boldsymbol{\rho}$ and input IP z, normalized to fall in $[-1, 0]$. The learning algorithms we study take as input a training set of IPs sampled from $\mathcal{D}$ and return a parameter vector. Since our goal is to minimize tree size, ideally, the size of the trees CPLEX builds using that parameter setting should be small in expectation over $\mathcal{D}$.

8.1.2 Dual functions

Our goal is to provide generalization guarantees for a function class $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P}\}$, where $\mathcal{P} \subseteq \mathbb{R}^d$ is a set of parameters and each function $u_{\boldsymbol{\rho}}$ maps a set of problem instances $\mathcal{Z}$ to $[0, 1]$. To do so, we use structure exhibited by the *dual function class*, where each dual function measures algorithmic performance as a function of the parameters. In this chapter, we slightly simplify the notation from Section 7.2 as follows: every function in the dual class is defined by an element $z \in \mathcal{Z}$, denoted $u_z : \mathcal{P} \to [0, 1]$. Naturally, $u_z(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(z)$. The dual class $\mathcal{U}^* = \{u_z \mid z \in \mathcal{Z}\}$ is the set of all dual functions.

The dual functions are intuitive in our integer programming example. For any IP z, the dual function u_z measures the size of the tree CPLEX builds (normalized to lie in the interval $[0, 1]$) when given x as input, as a function of the CPLEX parameters.

In Chapter 7, we showed that when the dual functions are piecewise-simple—for example, they are piecewise-constant with a small number of pieces—it is possible to provide strong generalization bounds. In many settings, however, we find that the dual functions themselves are not simple, but are approximated by simple functions, as in Figure 8.1. We formally define this concept as follows.

Definition 8.1.1 ((γ, p)-approximate)**.** Let $\mathcal{U} = \{u_\rho \mid \rho \in \mathcal{P}\}$ and $\mathcal{W} = \{w_\rho \mid \rho \in \mathcal{P}\}$ be two sets of functions mapping $\mathcal{Z}$ to $[0,1]$. We assume that all dual functions u_z and w_z are integrable over the domain $\mathcal{P}$. We say that the dual class $\mathcal{W}^*$ (γ, p)-*approximates* the dual class $\mathcal{U}^*$ if for every element z, the distance between the functions u_z and w_z is at most γ under the L^p-norm. For $p \in [1, \infty)$, this means that $\|u_z - w_z\|_p := \sqrt[p]{\int_{\mathcal{R}} |u_z(\rho) - w_z(\rho)|^p \, d\rho} \leq \gamma$ and when $p = \infty$, this means that $\|u_z - w_z\|_\infty := \sup_{\rho \in \mathcal{P}} |u_z(\rho) - w_z(\rho)| \leq \gamma$.

8.2 Learnability and approximability

In this section, we investigate the connection between learnability and approximability. In Section 8.2.1, we prove that when the dual functions are approximable under the L_∞-norm by simple functions, we can provide strong generalization bounds. In Section 8.2.2, we empirically evaluate these improved guarantees in the context of integer programming. Finally, in Section 8.2.3, we prove that it is not possible to provide non-trivial generalization guarantees (in the worst case) when the norm under which the dual functions are approximable is the L_p-norm for $p < \infty$.

8.2.1 Data-dependent generalization guarantees

We now show that if the dual class $\mathcal{U}^*$ is (γ, ∞)-approximated by the dual of a "simple" function class $\mathcal{W}$, we can provide strong generalization bounds for the class $\mathcal{U}$. Specifically, we show that if the dual class $\mathcal{U}^*$ is (γ, ∞)-approximated by the dual of a class $\mathcal{W}$ with small Rademacher complexity, then the Rademacher complexity of $\mathcal{U}$ is also small.

Theorem 8.2.1. *Let $\mathcal{U} = \{u_\rho \mid \rho \in \mathcal{P}\}$ and $\mathcal{W} = \{w_\rho \mid \rho \in \mathcal{P}\}$ consist of functions mapping $\mathcal{Z}$ to $[0,1]$. For any $\mathcal{S} \subseteq \mathcal{Z}$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) \leq \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) + \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \|u_z - w_z\|_\infty$.*

If the class $\mathcal{W}^*$ (γ, ∞)-approximates the class $\mathcal{U}^*$, then $\frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \|u_z - w_z\|_\infty$ is at most γ. If this term is smaller than γ for most sets $\mathcal{S} \sim \mathcal{D}^N$, then the bound on $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U})$ in Theorem 8.2.1 will often be even better than $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) + \gamma$.

Theorems 2.2.2 and 8.2.1 imply that with probability $1 - \delta$ over the draw of the set $\mathcal{S} \sim \mathcal{D}^N$, for all parameter vectors $\rho \in \mathcal{R}$, the difference between the empirical average value of f_ρ over $\mathcal{S}$ and its expected value is at most $\tilde{O}\left(\frac{1}{N} \sum_{x \in \mathcal{S}} \|u_z - w_z\|_\infty + \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) + \sqrt{\frac{1}{N}}\right)$. In our integer programming experiments, we show that this data-dependent generalization guarantee can be much tighter than the best-known worst-case guarantee.

Algorithm for finding approximating functions. We provide a dynamic programming (DP) algorithm for the widely-applicable case where the dual functions u_z are piecewise constant with a large number of pieces. Given an integer k, the algorithm returns a piecewise-constant function w_z with at most k pieces such that $\|u_z - w_z\|_\infty$ is minimized, as in Figure 8.1. As we describe in Section 8.2.2, when k and $\|u_z - w_z\|_\infty$ are small, Theorem 8.2.1 implies strong guarantees. We use this DP algorithm in our integer programming experiments.

Structural risk minimization. Theorem 8.2.1 illustrates a fundamental tradeoff in machine learning. The simpler the class $\mathcal{W}$, the smaller its Rademacher complexity, but (broadly speaking) the worse functions from its dual will be at approximating functions in $\mathcal{U}^*$. In other words, the simpler $\mathcal{W}$ is, the worse the approximation $\frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \|u_z - w_z\|_\infty$ will likely be. Therefore,

there is a tradeoff between generalizability and approximability. It may not be *a priori* clear how to balance this tradeoff. *Structural risk minimization (SRM)* is a classic, well-studied approach for optimizing tradeoffs between complexity and generalizability which we use in our experiments.

Our SRM approach is based on the following corollary of Theorem 8.2.1. Let $\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3, \ldots$ be a countable sequence of function classes where each $\mathcal{W}_j = \{w_{j,\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P}\}$ is a set of functions mapping $\mathcal{Z}$ to $[0,1]$. We use the notation $w_{j,z}$ to denote the duals of the functions in $\mathcal{W}_j$, so $w_{j,z}(\boldsymbol{\rho}) = w_{j,\boldsymbol{\rho}}(z)$.

Corollary 8.2.2. *With probability $1-\delta$ over the draw of the set $\mathcal{S} \sim \mathcal{D}^N$, for all $\boldsymbol{\rho} \in \mathcal{P}$ and all $j \geq 1$,*

$$\left| \frac{1}{N} \sum_{z \in \mathcal{S}} u_{\boldsymbol{\rho}}(z) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(z) \right] \right| = \tilde{O}\left(\frac{1}{N} \sum_{z \in \mathcal{S}} \left\| u_z - w_{j,z} \right\|_\infty + \widehat{\mathscr{R}}_{\mathcal{S}}\left(\mathcal{W}_j\right) + \sqrt{\frac{1}{N}} \right). \tag{8.1}$$

In our experiments, each dual class $\mathcal{W}_j^*$ consists of piecewise-constant functions with at most j pieces. This means that as j grows, the class $\mathcal{W}_j^*$ becomes more complex, or in other words, the Rademacher complexity $\widehat{\mathscr{R}}_{\mathcal{S}}\left(\mathcal{W}_j\right)$ also grows. Meanwhile, the more pieces a piecewise-constant function w_z has, the better it is able to approximate the dual function u_z. In other words, as j grows, the approximation term $\frac{1}{N}\sum_{x \in \mathcal{S}} \left\| u_z - w_{j,z} \right\|_\infty$ shrinks. SRM is the process of finding the level j in the nested hierarchy that minimizes the sum of these two terms, and therefore obtains the best generalization guarantee via Equation (8.1).

Remark 8.2.3. We conclude by noting that the empirical average $\frac{1}{N}\sum_{z \in \mathcal{S}} \left\| u_z - w_{j,z} \right\|_\infty$ in Equation (8.1) can be replaced by the expectation $\mathbb{E}_{z \sim \mathcal{D}}\left[\left\| u_z - w_{j,z} \right\|_\infty \right]$.

8.2.2 Improved integer programming guarantees

In this section, we demonstrate that our data-dependent generalization guarantees from Section 8.2.1 can be much tighter than worst-case generalization guarantees provided in prior research. We demonstrate these improvements in the context of integer programming algorithm configuration, which we introduced in Section 8.1.1. Our formal model is the same as that from Chapter 3, where we studied worst-case generalization guarantees. Each element of the set $\mathcal{Z}$ is an IP. The set $\mathcal{P}$ consists of CPLEX parameter settings. We assume there is an upper bound $\bar{\kappa}$ on the size of the largest tree we allow B&B to build before we terminate, as in prior research [21, 101, 109, 110]. In the full version of the paper, we describe our methodology for choosing $\bar{\kappa}$. Given a parameter setting ρ and an IP z, we define $u_\rho(z)$ to be the size of the tree CPLEX builds, capped at $\bar{\kappa}$, divided by $\bar{\kappa}$ (this way, $u_\rho(z) \in [0,1]$). We define the set $\mathcal{U} = \{u_\rho \mid \rho \in \mathcal{P}\}$.

We tune the parameter of B&B's variable selection policy (VSP). We described the purpose of VSPs in Section 8.1.1. We study *score-based VSPs*, defined as follows. Let score be a function that takes as input a partial B&B tree $\mathcal{T}$, a leaf of $\mathcal{T}$ representing an IP z, and an index $i \in [n]$, and returns a real-valued $\text{score}(\mathcal{T}, z, i)$. Let V be the set of variables that have not been branched on along the path from the root of $\mathcal{T}$ to z. A *score-based VSP* branches on the variable $\text{argmax}_{x[i] \in V}\{\text{score}(\mathcal{T}, z, i)\}$ at the node z.

We study how to learn a high-performing convex combination of any two scoring rules. We focus on four scoring rules in our experiments. To define them, we first recall some notation (first defined in Chapter 3). For an IP z with objective function $\boldsymbol{c} \cdot \boldsymbol{x}$, we denote an optimal solution to the LP relaxation of z as $\check{\boldsymbol{x}}_z = (\check{x}_z[1], \ldots \check{x}_z[n])$. We also use the notation $\check{c}_z = \boldsymbol{c} \cdot \check{\boldsymbol{x}}_z$. Finally,

we use the notation z_i^+ (resp., z_i^-) to denote the IP z with the additional constraint that $x[i] = 1$ (resp., $x[i] = 0$).[1]

We study four scoring rules score_L and score_S, score_A, and score_P:

- $\text{score}_L(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$. Under score_L, B&B branches on the variable leading to the **L**argest change in the LP objective value.
- $\text{score}_S(\mathcal{T}, z, i) = \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$. Under score_S, B&B branches on the variable leading to the **S**mallest change.
- $\text{score}_A(\mathcal{T}, z, i) = \frac{1}{6}\text{score}_L(\mathcal{T}, z, i) + \frac{5}{6}\text{score}_S(\mathcal{T}, z, i)$. This is a scoring rule that Achterberg [2] recommended. It balances the optimistic approach to branching under score_L with the pessimistic approach under score_S.
- $\text{score}_P(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, 10^{-6}\right\} \cdot \max\left\{\check{c}_z - \check{c}_{z_i^-}, 10^{-6}\right\}$. This is known as the ***P**roduct scoring rule*. Comparing $\check{c}_z - \check{c}_{z_i^-}$ and $\check{c}_z - \check{c}_{z_i^+}$ to 10^{-6} allows the algorithm to compare two variables even if $\check{c}_z - \check{c}_{z_i^-} = 0$ or $\check{c}_z - \check{c}_{z_i^+} = 0$. After all, suppose the scoring rule simply calculated the product $\left(\check{c}_z - \check{c}_{z_i^-}\right) \cdot \left(\check{c}_z - \check{c}_{z_i^+}\right)$ without comparing to 10^{-6}. If $\check{c}_z - \check{c}_{z_i^-} = 0$, then the score equals 0, canceling out the value of $\check{c}_z - \check{c}_{z_i^+}$ and thus losing the information encoded by this difference.

Fix any two scoring rules score_1 and score_2. We define $u_\rho(z)$ to be 0 minus the size of the tree B&B builds (normalized to lie in $[-1, 0]$) when it uses the score-based VSP defined by $(1-\rho)\text{score}_1 + \rho\text{score}_2$. Our goal is to learn the best convex combination of the two scoring rules. When $\text{score}_1 = \text{score}_L$ and $\text{score}_2 = \text{score}_S$, prior research has proposed several alternative settings for the parameter ρ [2, 32, 33, 81, 130], though no one setting is optimal across all applications.

Lemma 3.3.5 from Chapter 3 implies the following worst-case generalization bound: with probability $1 - \delta$ over the draw of N samples $\mathcal{S} \sim \mathcal{D}^N$, for all $\rho \in [0, 1]$,

$$\left|\frac{1}{N}\sum_{z\in\mathcal{S}} u_\rho(z) - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_\rho(z)\right]\right| \leq 2\sqrt{\frac{2\ln(N\left(n^{2(\bar{\kappa}+1)}-1\right)+1)}{N}} + 3\sqrt{\frac{1}{2N}\ln\frac{2}{\delta}}. \tag{8.2}$$

This worst-case bound can be large when $\bar{\kappa}$ is large. We find that although the duals u_z are piecewise-constant with many pieces, they can be approximated piecewise-constant functions with few pieces, as in Figure 8.1. As a result, we improve over Equation (8.2) via Theorem 8.2.1, our data-dependent bound.

To make use of Theorem 8.2.1, we now formally define the function class whose dual (γ, ∞)-approximates $\mathcal{U}^*$. We first define the dual class, then the primal class. To this end, fix some integer $j \geq 1$ and let $\mathcal{H}_j$ be the set of all piecewise-constant functions mapping $[0, 1]$ to $[-1, 0]$ with at most j pieces. For every IP z, we define $w_{j,z} \in \text{argmin}_{h\in\mathcal{H}_j} \|u_z - h\|_\infty$, breaking ties in some fixed but arbitrary manner. The function $w_{j,z}$ can be found via dynamic programming. We define the dual class $\mathcal{W}_j^* = \left\{w_{j,z} \mid z \in \mathcal{Z}\right\}$. Therefore, the dual class $\mathcal{W}_j^*$ is consists of piecewise-constant functions with at most j pieces. In keeping with the definition of primal and dual functions from Section 8.1.2, for every parameter $\rho \in [0, 1]$ and IP z, we define $w_{j,\rho}(x) = w_{j,z}(\rho)$. Finally, we define the primal class $\mathcal{W}_j = \left\{w_{j,\rho} \mid \rho \in [0, 1]\right\}$.

[1] If z_i^+ (resp., z_i^-) is infeasible, then we define $\check{c}_z - \check{c}_{z_i^+}$ (resp., $\check{c}_z - \check{c}_{z_i^-}$) to be some large number greater than $||\boldsymbol{c}||_1$.

To apply our results from Section 8.2.1, we must bound the Rademacher complexity of the set $\mathcal{W}_j$. Doing so is simple due to the structure of the dual class $\mathcal{W}_j^*$.

Lemma 8.2.4. *For any set $\mathcal{S} \subseteq \mathcal{Z}$ of integer programs, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}_j) \leq \sqrt{\frac{2\ln(|\mathcal{S}|(j-1)+1)}{|\mathcal{S}|}}$.*

This lemma together with Remark 8.2.3 and Corollary 8.2.2 imply that with probability $1-\delta$ over $\mathcal{S} \sim \mathcal{D}^N$, for all parameters $\rho \in [0,1]$ and $j \geq 1$, $\left|\frac{1}{N}\sum_{z\in\mathcal{S}} u_\rho(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[u_\rho(z)\right]\right|$ is upper-bounded by the minimum of Equation (8.2) and

$$2\gamma_j + 2\sqrt{\frac{2\ln(N(j-1)+1)}{N}} + \sqrt{\frac{2}{N}\ln\frac{2(\pi j)^2}{3\delta}}, \tag{8.3}$$

where $\gamma_j = \mathbb{E}_{z\sim\mathcal{D}}\left[\left\|u_z - w_{j,z}\right\|_\infty\right]$. As j grows, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}_j)$ grows, but the dual class $\mathcal{W}_j^*$ is better able to approximate $\mathcal{U}^*$. In our experiments, we optimize this tradeoff between generalizability and approximability.

Experiments. We analyze distributions over IPs formulating the combinatorial auction winner determination problem under the OR-bidding language [169], which we generate using the Combinatorial Auction Test Suite (CATS) [124]. We use the "arbitrary" generator with 200 bids and 100 goods, resulting in IPs with 200 variables, and the "regions" generator with 400 bids and 200 goods, resulting in IPs with 400 variables.

We use the same code as in Chapter 3 to compute the functions u_z. It overrides the default VSP of CPLEX 12.8.0.0 using the C API. All experiments were run on a 64-core machine with 512 GB of RAM.

In Figures 8.2a-8.2c, we select $\text{score}_1, \text{score}_2 \in \{\text{score}_L, \text{score}_S, \text{score}_A, \text{score}_P\}$ and compare the worst-case and data-dependent bounds. First, we plot the worst-case bound from Equation (8.2), with $\delta = 0.01$, as a function of the number of training examples N. This is the black, dotted line in Figures 8.2a-8.2c.

Next, we plot the data-dependent bound, which is the red, solid line in Figures 8.2a-8.2c. To calculate the data-dependent bound in Equation (8.3), we have to estimate $\mathbb{E}_{z\sim\mathcal{D}}\left[\left\|u_z - w_{j,z}\right\|_\infty\right]$ for all $j \in [1600]$.[2] To do so, we draw $M = 6000$ IPs $z_1, \ldots, z_M$ from the distribution $\mathcal{D}$. We estimate $\mathbb{E}_{z\sim\mathcal{D}}\left[\left\|u_z - w_{j,z}\right\|_\infty\right]$ via the empirical average $\frac{1}{M}\sum_{i=1}^M \left\|u_{z_i} - w_{j,z_i}\right\|_\infty$. A Hoeffding bound guarantees that with probability 0.995, for all $j \in [1600]$,

$$\mathbb{E}\left[\left\|u_z - w_{j,z}\right\|_\infty\right] \leq \frac{1}{M}\sum_{i=1}^M \left\|u_{z_i} - w_{j,z_i}\right\|_\infty + \frac{1}{40}. \tag{8.4}$$

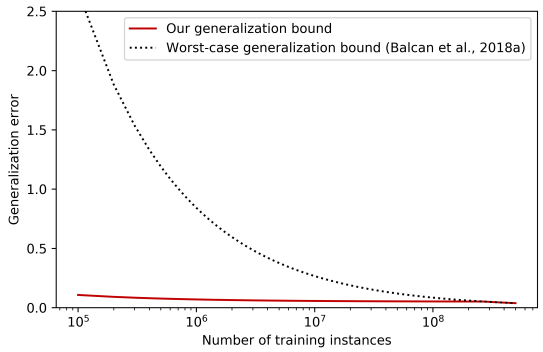
With thereby estimate our data-dependent bound Equation (8.3) using Equation (8.5), below:

$$\min_{j\in[1600]} \left\{ 2\left(\frac{1}{M}\sum_{i=1}^M \left\|u_{z_i} - w_{j,z_i}\right\|_\infty + \frac{1}{40}\right) + 2\sqrt{\frac{2\ln(N(j-1)+1)}{N}} + \sqrt{\frac{2}{N}\ln\frac{(20\pi j)^2}{3}} \right\}. \tag{8.5}$$

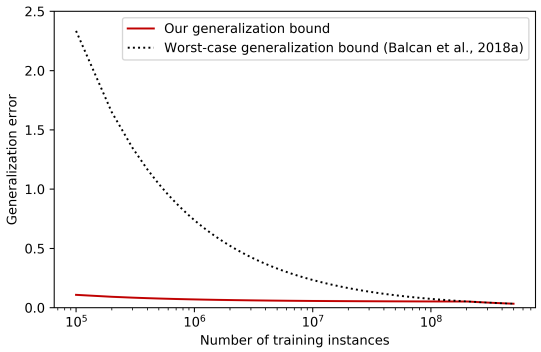
The only difference between these bounds is that Equation (8.3) relies on the left-hand-side of Equation (8.4) and Equation (8.5) relies on the right-hand-side of Equation (8.4) and sets $\delta = 0.005$.[3] In Figures 8.2a-8.2c, the red solid line equals the minimum of Equations (8.2) and (8.5) as a function of the number of training examples N.

[2]We choose the range $j \in [1600]$ because under these distributions, the functions u_z generally have at most 1600 pieces.

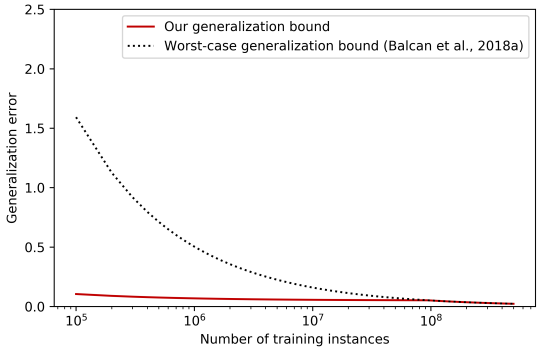
[3]Like the worst-case bound, Equation (8.5) holds with probability 0.99, because with probability 0.995, Equation (8.4) holds, and with probability 0.995, the bound from Equation (8.3) holds.



(a) Results using SRM on the CATS "regions" generator with $\text{score}_1 = \text{score}_L$ and $\text{score}_2 = \text{score}_S$.



(b) Results using SRM on the CATS "arbitrary" generator with $\text{score}_1 = \text{score}_L$ and $\text{score}_2 = \text{score}_S$.



(c) Results using SRM on the CATS "arbitrary" generator with $\text{score}_1 = \text{score}_P$ and $\text{score}_2 = \text{score}_A$.

Figure 8.2: Experimental results.

In Figures 8.2a, 8.2b, and 8.2c, we see that our bound significantly beats the worst-case bound up until the point there are approximately 100,000,000 training instances. At this point, the worst-case guarantee is better than the data-dependent bound, which makes sense because it goes to zero as N goes to infinity, whereas the term $\frac{1}{M}\sum_{i=1}^{M}\left\|u_{z_i} - w_{j,z_i}\right\|_\infty + \frac{1}{40}$ in our bound (Equation (8.5)) is a constant.

Figures 8.2a-8.2c also demonstrate that even when there are only 10^5 training instances, our bound provides a generalization guarantee of approximately 0.1. Meanwhile, in Figure 8.2a, $7 \cdot 10^7$ training instances are necessary to provide a generalization guarantee of 0.1 under the worst-case bound, so the sample complexity implied by our analysis is 700 times better. Similarly, in Figure 8.2b, our sample complexity is 500 times better, and in Figure 8.2c, it is 250 times better.

8.2.3 Rademacher complexity lower bound

In this section, we show that (γ, p)-approximability with $p < \infty$ does not necessarily imply strong generalization guarantees of the type we saw in Section 8.2.1. We show that it is possible for a dual class $\mathcal{U}^*$ to be well-approximated by the dual of a class $\mathcal{W}$ with $\widehat{\mathcal{R}}_S(\mathcal{W}) = 0$, yet for the primal $\mathcal{U}$ to have high Rademacher complexity.

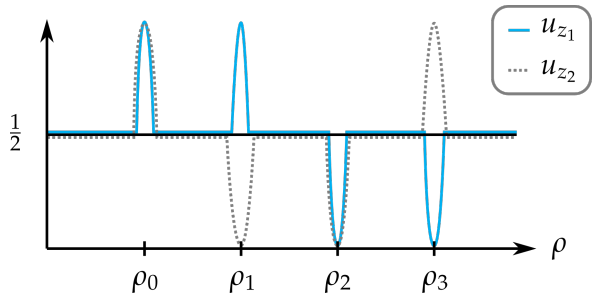


Figure 8.3: The dual functions u_{z_1} and u_{z_2} are well-approximated by the constant function $\rho \mapsto \frac{1}{2}$ under, for example, the L^1-norm because the integrals $\int_{\mathcal{P}} \left|u_{z_i}(\rho) - \frac{1}{2}\right| \, dr$ are small; for most ρ, $u_{z_i}(\rho) = \frac{1}{2}$. The approximation is not strong under the L^∞-norm, since $\max_{\rho \in \mathcal{P}} \left|u_{z_i}(\rho) - \frac{1}{2}\right| = \frac{1}{2}$. The function class $\mathcal{U}$ corresponding to these duals has a large Rademacher complexity.

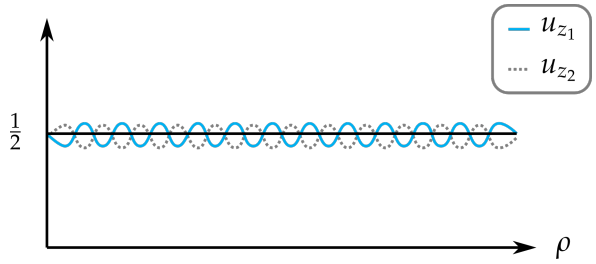


Figure 8.4: The dual functions u_{z_1} and u_{z_2} are well-approximated by the constant function $\rho \mapsto \frac{1}{2}$ under the L^∞-norm since $\max_{\rho \in \mathcal{P}} \left|u_{z_i}(\rho) - \frac{1}{2}\right|$ is small. The function class $\mathcal{U}$ corresponding to these duals has a small Rademacher complexity.

Figures 8.3 and 8.4 help explain why there is this sharp constrast between the L^∞- and L^p-norms for $p < \infty$. Figure 8.3 illustrates two dual functions u_{z_1} (the blue solid line) and u_{z_2} (the grey dotted line). Let $\mathcal{W}$ be the extremely simple function class $\mathcal{W} = \left\{w_\rho : \rho \in \mathbb{R}\right\}$ where $w_\rho(z) = \frac{1}{2}$ for every $z \in \mathcal{Z}$. It is easy to see that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{W}) = 0$ for any set $\mathcal{S}$. Moreover, every dual function w_z is also simple, because $w_z(\rho) = w_\rho(z) = \frac{1}{2}$. From Figure 8.3, we can see that the functions u_{z_1} and u_{z_2} are well approximated by the constant function $w_{z_1}(\rho) = w_{z_2}(\rho) = \frac{1}{2}$ under, for example, the L^1-norm because the integrals $\int_{\mathcal{P}} \left|u_{z_i}(\rho) - \frac{1}{2}\right| \, dr$ are small. However, the approximation is not strong under the L^∞-norm, since $\max_{\rho \in \mathcal{P}} \left|u_{z_i}(\rho) - \frac{1}{2}\right| = \frac{1}{2}$ for $i \in \{1, 2\}$.

Moreover, despite the fact that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{W}) = 0$, we have that $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$ when $\mathcal{S} = \{z_1, z_2\}$, which makes Theorem 2.2.2 meaningless. At a high level, this is because when $\sigma_1 = 1$, we can ensure that $\sigma_1 u_\rho(z_1) = \sigma_1 u_{z_1}(\rho) = 1$ by choosing $\rho \in \{\rho_0, \rho_1\}$ and when $\sigma_1 = -1$, we can ensure that $\sigma_1 u_\rho(z_1) = 0$ by choosing $\rho \in \{\rho_2, \rho_3\}$. A similar argument holds for σ_2. In summary, (γ, p)-approximability for $p < \infty$ does not guarantee low Rademacher complexity.

Meanwhile, in Figure 8.4, $w_{z_i}(\rho) = \frac{1}{2}$ and $u_{z_i}(\rho)$ are close under the L^∞-norm for every parameter ρ. As a result, for any noise vector $\boldsymbol{\sigma} \in \{-1, 1\}^2$, $\sup_{\rho \in \mathbb{R}} \left\{\sigma_1 u_{z_1}(\rho) + \sigma_2 u_{z_2}(\rho)\right\}$ is close to $\sup_{\rho \in \mathbb{R}} \left\{\sigma_1 w_{z_1}(\rho) + \sigma_2 w_{z_2}(\rho)\right\}$. This implies that the Rademacher complexities $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{W})$ and $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{U})$ are close. This illustration exemplifies Theorem 8.2.1: (γ, ∞)-approximability implies strong Rademacher bounds.

We now prove that (γ, p)-approximability by a simple class for $p < \infty$ does not guarantee low Rademacher complexity.

Theorem 8.2.5. *For any $\gamma \in (0, 1/4)$ and any $p \in [1, \infty)$, there exist function classes $\mathcal{U}, \mathcal{W} \subset [0,1]^{\mathcal{X}}$ such that the dual class $\mathcal{W}^*$ (γ, p)-approximates $\mathcal{U}^*$ and for any $N \geq 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) = 0$ and $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$.*

Remark 8.2.6. Suppose, for example, that $\mathcal{P} = [0,1]^d$. Theorem 8.2.5 implies that even if the difference $|u_z(\boldsymbol{\rho}) - w_z(\boldsymbol{\rho})|$ is small for all z in expectation over $\boldsymbol{\rho} \sim \text{Uniform}(\mathcal{P})$, the function class $\mathcal{U}$ may not have Rademacher complexity close to $\mathcal{W}$.

Chapter 9

Estimating approximate incentive compatibility

In this chapter, we return to sample complexity questions in the context of mechanism design, though in a new context. In Chapter 6, we analyzed the number of samples sufficient to learn an incentive compatible mechanism with high revenue, profit, or social welfare. The mechanisms used in practice, however, are often not incentive compatible; they are *manipulable*. This is the case in many settings for selling, buying, matching (such as school choice), voting, and so on. In this chapter, we present techniques for estimating how far a mechanism is from incentive compatible. Given samples from the agents' type distribution, we show how to estimate the extent to which an agent can improve his utility by misreporting his type.

Examples of manipulable mechanisms abound in practice. For instance, most real-world auctions are implemented using the first-price mechanism. In multi-unit sales settings, the U.S. Treasury has used discriminatory auctions, a variant of the first-price auction, to sell treasury bills since 1929 [116]. Similarly, electricity generators in the U.K. use discriminatory auctions to sell their output [116]. Sponsored search auctions are typically implemented using variants of the generalized second-price auction [72, 183]. In the past year, many major display ad exchanges including AppNexus, Rubicon, and Index Exchange have transitioned to first-price auctions, driven by ad buyers who believe it offers a higher degree of transparency [91, 156]. Finally, nearly all fielded combinatorial auctions are manipulable. Essentially all combinatorial sourcing auctions are implemented using the first-price mechanism [171]. Combinatorial spectrum auctions are conducted using a variety of manipulable mechanisms. Even the "incentive auction" used to source spectrum licenses back from low-value broadcasters—which has sometimes been hailed as obviously incentive compatible—is manipulable once one takes into account the fact that many owners own multiple broadcasting stations, or the fact that stations do not only have the option to keep or relinquish their license, but also the option to move to (two) less desirable spectrum ranges [149].

Many reasons have been suggested why manipulable mechanisms are used in practice. First, the rules are often easier to explain. Second, incentive compatibility ceases to hold even in the relatively simple context of the Vickrey auction when determining one's own valuation is costly (for example, due to computation or information gathering effort) [168]. Third, bidders may have even more incentive to behave strategically when they can conduct computation or information gathering on each others' valuations, and if they can incrementally decide how to allocate valuation-determination effort [120, 121]. Fourth, in combinatorial settings, well-known incentive-compatible mechanisms such as the Vickrey-Clarke-Groves (VCG) mechanism require bidders to submit bids for every bundle, which generally requires a prohibitive amount of valuation computation (solving a local planning problem for potentially every bundle, and each planning problem, itself, can be NP-complete) or information acquisition [53, 155, 167, 172]. Fifth, in settings such as sourcing, single-shot incentive compatible mechanisms such as the VCG are generally not incentive compatible when the bid-taker uses bids from one auction to adjust the parameters of later auctions in later years [171]. Sixth, incentive compatible mechanisms may leak the agents' sensitive private information [162]. Seventh, incentive compatibility typically

ceases to hold if agents are not risk neutral (i.e., the utility functions are not quasi-linear). There are also sound theoretical reasons why the designer sometimes prefers manipulable mechanisms. Specifically, there exist settings where the designer does better than under any incentive compatible mechanism if the agents cannot solve hard computational or communication problems, and equally well if they can [56, 152].

Due in part to the ubiquity of manipulable mechanisms, a growing body of additional research has explored mechanisms that are not incentive compatible [10, 14, 58, 64, 70, 71, 77, 86, 114, 133, 139]. A popular and widely-studied relaxation of incentive compatibility is γ-*incentive compatibility* [10, 14, 64, 70, 114, 133, 139], which requires that no agent can improve his utility by more than γ when he misreports his type.

The results in this section are joint work with Nina Balcan and Tuomas Sandholm [26]. Our current results were published in EC 2019.

9.0.1 Our contributions

Much of the literature on γ-incentive compatibility rests on the strong assumption that the agents' type distribution is known. In reality, this information is rarely available. We relax this assumption and instead assume we only have samples from the distribution [128, 129, 173]. We present techniques with provable guarantees that the mechanism designer can use to estimate how far a mechanism is from incentive compatible. We analyze both the *ex-interim* and *ex-ante*[1] settings: in the *ex-interim* case, we bound the amount any agent can improve his utility by misreporting his type, in expectation over the other agents' types, no matter his true type. In the weaker *ex-ante* setting, the expectation is also over the agent's true type as well. In the *ex-interim* case, we adopt the common assumption that each agent's type is drawn independently from the other agents' types. In the *ex-ante* case, we drop this assumption: the agents' types may be arbitrarily correlated.

Our estimate is simple: it measures the maximum utility an agent can gain by misreporting his type on average over the samples, whenever his true and reported types are from a finite subset of the type space. We bound the difference between our incentive compatibility estimate and the true incentive compatibility approximation factor γ.

We apply our estimation technique to a variety of auction classes. We begin with the first-price auction, in both single-item and combinatorial settings. Our guarantees can be used by display ad exchanges, for instance, to measure the extent to which incentive compatibility will be compromised if the exchange transitions to using first-price auctions [91, 156]. In the single-item setting, we prove that the difference between our estimate and the true incentive compatibility approximation factor is $\tilde{O}\left(\left(n+\kappa^{-1}\right)/\sqrt{N}\right)$, where n is the number of bidders, N is the number of samples, and $[0,\kappa]$ contains the range of the density functions defining the agents' type distribution. We prove the same bound for the second-price auction with *spiteful bidders* [38, 144, 177, 180], where each bidder's utility not only increases when his surplus increases but also decreases when the other bidders' surpluses increase.

In a similar direction, we analyze the class of generalized second-price auctions [72], where m sponsored search slots are for sale. The mechanism designer assigns a real-valued weight per bidder, collects a bid per bidder indicating their value per click, and allocates the slots in order of the bidders' weighted bids. In this setting, we prove that the difference between our

[1]We do not study *ex-post* or *dominant-strategy* approximate incentive compatibility because they are worst-case, distribution-independent notions. Therefore, we cannot hope to measure the *ex-post* or dominant-strategy approximation factors using samples from the agents' type distribution.

incentive compatibility estimate and the true incentive compatibility approximation bound is $\tilde{O}\left(\left(n^{3/2}+\kappa^{-1}\right)/\sqrt{N}\right)$.

We also analyze multi-parameter mechanisms beyond the first-price combinatorial auction, namely, the uniform-price and discriminatory auctions, which are used extensively in markets around the world [116]. In both, the auctioneer has m identical units of a single good to sell. Each bidder submits m bids indicating their value for each additional unit of the good. The number of goods the auctioneer allocates to each bidder equals the number of bids that bidder has in the top m bids. Under a discriminatory auction, each agent pays the amount she bid for the number of units she is allocated. Under a uniform-price auction, the agents pay the market-clearing price, meaning the total amount demanded equals the total amount supplied. In both cases, we prove that the difference between our incentive compatibility estimate and the true incentive compatibility approximation bound is $\tilde{O}\left(\left(nm^2+\kappa^{-1}\right)/\sqrt{N}\right)$.

A strength of our estimation techniques is that they are application-agnostic. For example, they can be used as a tool in *incremental mechanism design* [58], a subfield of *automated mechanism design* [54, 170], where the mechanism designer gradually adds incentive compatibility constraints to her optimization problem until she has met a desired incentive compatibility guarantee.

Key challenges. To prove our guarantees, we must estimate the value γ defined such that no agent can misreport his type in order to improve his expected utility by more than γ, no matter his true type. We propose estimating γ by measuring the extent to which an agent can improve his utility by misreporting his type on average over the samples, whenever his true and reported types are from a finite subset $\mathcal{G}$ of the type space. We denote this estimate as $\hat{\gamma}$. The challenge is that by searching over a subset of the type space, we might miss pairs of types θ and $\hat{\theta}$ where an agent with type θ can greatly improve his expected utility by misreporting his type as $\hat{\theta}$. Indeed, utility functions are often volatile in mechanism design settings. For example, under the first- and second-price auctions, nudging an agent's bid from below the other agents' largest bid to above will change the allocation, causing a jump in utility. Thus, there are two questions we must address: which finite subset should we search over and how do we relate our estimate $\hat{\gamma}$ to the true approximation factor γ?

We provide two approaches to constructing the cover $\mathcal{G}$. The first is to run a greedy procedure based off a classic algorithm from theoretical machine learning. This approach is extremely versatile: it provides strong guarantees no matter the setting. However, it may be difficult to implement.

Meanwhile, implementing our second approach is straightforward: the cover is simply a uniform grid over the type space (assuming the type space equals $[0,1]^m$ for some integer m). The efficacy of this approach depends on a "niceness" property that holds under mild assumptions. To analyze this second approach, we must understand how the edge-length of the grid effects our error bound relating the estimate $\hat{\gamma}$ to the true approximation factor γ. To do so, we rely on the notion of *dispersion* [22]. Roughly speaking, a set of piecewise Lipschitz functions is (w,k)-dispersed if every ball of radius w in the domain contains at most k of the functions' discontinuities. Given a set of N samples from the distribution over agents' types, we analyze the dispersion of function sequences from two related families. In both families, the function sequences are of length N: each function is defined by one of the N samples. In the first family, each sequence is defined by a true type $\boldsymbol{\theta}$ and measures a fixed agent's utility as a function of her reported type $\hat{\boldsymbol{\theta}}$, when the other agents' bids are defined by the samples. In the second family, each sequence is defined by a reported type $\hat{\boldsymbol{\theta}}$ and measures the agent's utility as a function of

her true type θ. We show that if all function sequences from both classes are (w, k)-dispersed, then we can use a grid with edge-length w to discretize the agents' type space.

We show that for a wide range of mechanism classes, dispersion holds under mild assumptions. As we describe more in Section 9.2.1, this requires us to prove that with high probability, each function sequence from several infinite families of sequences is dispersed. This facet of our analysis is notably different from prior research by Balcan et al. [22]: in their applications, it is enough to show that with high probability, a single, finite sequence of functions is dispersed. Our proofs thus necessitate that we carefully examine the structure of the utility functions that we analyze. In particular, we prove that across all of the infinitely-many sequences, the functions' discontinuities are invariant. As a result, it is enough to prove dispersion for single generic sequence in order to guarantee dispersion for all of the sequences.

Finally, we prove that for both choices of the cover $\mathcal{G}$, so long as the intrinsic complexities of the agents' utility functions are not too large (as measured by the learning-theoretic notion of *pseudo-dimension* [159]), then our estimate $\hat{\gamma}$ quickly converges to the true approximation factor γ as the number of samples grows.

9.0.2 Additional related research

Strategy-proofness in the large. Azevedo and Budish [14] propose a variation on approximate incentive compatibility called *strategy-proofness in the large (SP-L)*. SP-L requires that it is approximately optimal for agents to report their types truthfully in sufficiently large markets. As in this chapter, SP-L is a condition on *ex-interim* γ-incentive compatibility. The authors argue that SP-L approximates, in large markets, attractive properties of a mechanism such as strategic simplicity and robustness. They categorize a number of mechanisms as either SP-L or not. For example, they show that the discriminatory auction is manipulable in the large whereas the uniform-price auction is SP-L. Measuring a mechanism's SP-L approximation factor requires knowledge of the distribution over agents' types, whereas we only require sample access to this distribution. Moreover, we do not make any large-market assumptions: our guarantees hold regardless of the number of agents.

Comparing mechanisms by their vulnerability to manipulation. Pathak and Sönmez [157] analyze *ex-post* incentive compatibility without any connection to approximate incentive compatibility. They say that one mechanism M is at least as manipulable as another M' if every type profile that is vulnerable to manipulation under M is also vulnerable to manipulation under M'. They apply their formalism in the context of school assignment mechanisms, the uniform-price auction, the discriminatory auction, and several keyword auctions. We do not study *ex-post* approximate incentive compatibility because it is a worst-case, distribution-independent notion. Therefore, we cannot hope to measure an *ex-post* approximation factor using samples from the agents' type distribution. Rather, we are concerned with providing data-dependent bounds on the *ex-interim* and *ex-ante* approximation factors. Another major difference is that our work provides quantitative results on manipulability while theirs provides boolean comparisons as to the relative manipulability of mechanisms. Finally, our measure applies to all mechanisms while theirs cannot rank all mechanisms because in many settings, pairs of mechanisms are incomparable according to their boolean measure.

Mechanism design via deep learning. In mechanism design via deep learning [71, 77, 86], the learning algorithm receives samples from the distribution over agents' types. The resulting allocation and payment functions are characterized by neural networks, and thus the corresponding

mechanism may not be incentive compatible. In an attempt to make these mechanisms nearly incentive compatible, the authors of these works add constraints to the deep learning optimization problem enforcing that the resulting mechanism be incentive compatible over a set of buyer values sampled from the underlying, unknown distribution.

In research that was conducted simultaneously, Dütting et al. [71] provide guarantees on the extent to which the resulting mechanism—characterized by a neural network—is approximately incentive compatible, in the sense of *expected ex-post incentive compatibility*: in expectation over all agents' types, what is the maximum amount that an agent can improve her utility by misreporting her type? This is a different notion of approximate incentive compatibility than those we study. In short, we bound the maximum expected change in utility an agent can induce by misreporting her type (Definition 9.1.1), no matter her true type, where the expectation is over the other agents' types. In contrast, Dütting et al. [71] bound the expected maximum change in utility an agent can induce, where the expectation is over all agents' types. As a result, our bounds are complementary, but not overlapping. Moreover, the sets of mechanisms we analyze and Dütting et al. [71] analyze are disjoint: Dütting et al. [71] provide bounds for mechanisms represented as neural networks, whereas we instantiate our guarantees for single-item and combinatorial first-price auctions, generalized second-price auction, discriminatory auction, uniform-price auction, and second-price auction with spiteful bidders.

Sample complexity of revenue maximization. A long line of research has studied revenue maximization via machine learning from a theoretical perspective (cited in Section 6.1). The mechanism designer receives samples from the type distribution which she uses to find a mechanism that is, ideally, nearly optimal in expectation. That research has only studied incentive compatible mechanism classes. Moreover, in this chapter, it is not enough to provide generalization guarantees; we must both compute our estimate of the incentive compatibility approximation factor and bound our estimate's error. This type of error has nothing to do with revenue functions, but rather utility functions. These factors in conjunction mean that our research differs significantly from prior research on generalization guarantees in mechanism design.

In a related direction, Chawla et al. [48, 49, 50] study counterfactual revenue estimation. Given two auctions, they provide techniques for estimating one of the auction's equilibrium revenue from the other auction's equilibrium bids. They also study social welfare in this context. Thus, their research is tied to selling mechanisms, whereas we study more general mechanism design problems from an application-agnostic perspective.

Incentive compatibility from a buyer's perspective. Lahaie et al. [118] also provide tools for estimating approximate incentive compatibility, but from the buyer's perspective rather than the mechanism designer's perspective. As such, the type of information available to their estimation tools versus ours is different. Moreover, they focus on ad auctions, whereas we study mechanism design in general and apply our techniques to a wide range of settings and mechanisms.

9.1 Preliminaries and notation

There are n agents who each have a *type*. We denote agent i's type as θ_i, which is an element of a (potentially infinite) set Θ_i. A mechanism takes as input the agents' *reported types*, which it uses to choose an outcome. We denote agent i's reported type as $\hat{\theta}_i \in \Theta_i$. We denote all n agents' types as $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ and reported types as $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_n)$. For $i \in [n]$, we use the standard notation $\boldsymbol{\theta}_{-i} \in \times_{j \neq i} \Theta_j$ to denote all $n-1$ agents' types except agent i. Using this

notation, we denote the type profile $\boldsymbol{\theta}$ representing all n agents' types as $\boldsymbol{\theta} = (\theta_i, \boldsymbol{\theta}_{-i})$. Similarly, $\hat{\boldsymbol{\theta}} = \left(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}\right)$. We assume there is a distribution $\mathcal{D}$ over all n agents' types, and thus the support of $\mathcal{D}$ is contained in $\times_{i=1}^{n}\Theta_i$. We use $\mathcal{D}|_{\theta_i}$ to denote the conditional distribution given θ_i, so the support of $\mathcal{D}|_{\theta_i}$ is contained in $\times_{j\neq i}\Theta_j$. We assume that we can draw samples $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ independently from $\mathcal{D}$. This is the same assumption made in a long line of work on mechanism design via machine learning (including, for example, work by Balcan et al. [18], Elkind [73], and Cole and Roughgarden [52]).

Given a mechanism M and agent $i \in [n]$, we use the notation $u_{i,M}\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}\right)$ to denote the utility agent i receives when the agents have types $\boldsymbol{\theta}$ and reported types $\hat{\boldsymbol{\theta}}$. We assume[2] it maps to $[-1, 1]$. When $\boldsymbol{\theta}_{-i} = \hat{\boldsymbol{\theta}}_{-i}$, we use the simplified notation $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) := u_{i,M}\left(\left(\theta_i, \boldsymbol{\theta}_{-i}\right), \left(\hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)\right)$.

At a high level, a mechanism is incentive compatible if no agent can ever increase her utility by misreporting her type. A mechanism is γ-*incentive compatible* if each agent can increase her utility by an additive factor of at most γ by misreporting her type [10, 14, 58, 64, 70, 71, 77, 86, 114, 133, 139]. In this chapter, we concentrate on *ex-interim* approximate incentive compatibility [14, 133].

Definition 9.1.1. A mechanism M is ex-interim γ-*incentive compatible* if for each $i \in [n]$ and all $\theta_i, \hat{\theta}_i \in \Theta_i$, agent i with type θ_i can increase her expected utility by an additive factor of at most γ by reporting her type as $\hat{\theta}_i$, so long as the other agents report truthfully. In other words, $\mathbb{E}_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}|_{\theta_i}}\left[u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right)\right] \geq E_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}|_{\theta_i}}\left[u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)\right] - \gamma$.

In our EC 2019 paper [26], we also study *ex-ante* approximate incentive compatibility, where the above definition holds in expectation over the draw of the types $\boldsymbol{\theta} \sim \mathcal{D}$.

We do not study *ex-post*[3] or *dominant-strategy*[4] approximate incentive compatibility because they are worst-case, distribution-independent notions. Therefore, we cannot hope to measure the *ex-post* or dominant-strategy approximation factors using samples from the agents' type distribution.

9.2 Estimating approximate ex-interim incentive compatibility

In this section, we show how to estimate the *ex-interim* incentive compatibility approximation guarantee using data. We assume there is an unknown distribution $\mathcal{D}$ over agents' types, and we operate under the common assumption [14, 16, 41, 42, 85, 93, 133, 192] that the agents' types are independently distributed. In other words, for each agent $i \in [n]$, there exists a distribution ϕ_i over their type space Θ_i such that $\mathcal{D} = \times_{i=1}^{n}\phi_i$. For each agent $i \in [n]$, we receive a set $\mathcal{S}_{-i}$ of samples independently drawn from $\mathcal{D}_{-i} = \times_{j\neq i}\phi_j$. For each mechanism $M \in \mathcal{M}$, we show how to use the samples to estimate a value $\hat{\gamma}_M$ such that:

> *With probability* $1-\delta$ *over the draw of the* n *sets of samples* $\mathcal{S}_{-1}, \ldots, \mathcal{S}_{-n}$*, for any agent* $i \in [n]$ *and all pairs* $\theta_i, \hat{\theta}_i \in \Theta_i$, $\mathbb{E}_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}_{-i}}\left[u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right)\right] \leq \hat{\gamma}_M$.

[2]Our estimation error only increases by a multiplicative factor of H if the range of the utility functions is $[-H, H]$ instead of $[-1, 1]$.

[3]A mechanism M is *ex-post incentive compatible* if for each $i \in [n]$ and all $\theta_i, \hat{\theta}_i \in \Theta_i$, and all $\boldsymbol{\theta}_{-i} \in \times_{j\neq i}\Theta_j$, agent i with type θ_i cannot increase her utility by reporting her type as $\hat{\theta}_i$, so long as the other agents report truthfully. In other words, $u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right) \geq u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$.

[4]A mechanism M is *dominant-strategy incentive compatible* if for each $i \in [n]$ and all $\theta_i, \hat{\theta}_i \in \Theta_i$, agent i with type θ_i cannot increase her utility by reporting her type as $\hat{\theta}_i$, no matter which types the other agents report. In other words, for all $\boldsymbol{\theta}_{-i}, \hat{\boldsymbol{\theta}}_{-i} \in \times_{j\neq i}\Theta_j$, $u_{i,M}\left(\left(\theta_i, \boldsymbol{\theta}_{-i}\right), \left(\theta_i, \hat{\boldsymbol{\theta}}_{-i}\right)\right) \geq u_{i,M}\left(\left(\theta_i, \boldsymbol{\theta}_{-i}\right), \left(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}\right)\right)$.

To this end, one simple approach, informally, is to estimate $\hat{\gamma}_M$ by measuring the extent to which any agent i with any type θ_i can improve his utility by misreporting his type, averaged over all profiles in $\mathcal{S}_{-i}$. In other words, we can estimate $\hat{\gamma}_M$ by solving the following optimization problem:

$$\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\}. \tag{9.1}$$

Unfortunately, in full generality, there might not be a finite-time procedure to solve this optimization problem, so in Section 9.2.1, we propose more nuanced approaches based on optimizing over finite subsets of $\Theta_i \times \Theta_i$. As a warm-up and a building block for our main theorems in that section, we prove that with probability $1 - \delta$, for all mechanisms $M \in \mathcal{M}$,

$$\begin{aligned} &\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}} \left[u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right) \right] \right\} \\ &\leq \max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + \epsilon_{\mathcal{M}}(N, \delta), \end{aligned} \tag{9.2}$$

where $\epsilon_{\mathcal{M}}(N, \delta)$ is an error term that converges to zero as the number N of samples grows. Its convergence rate depends on the *intrinsic complexity* of the utility functions corresponding to the mechanisms in $\mathcal{M}$, which we formalize using the learning-theoretic tool *pseudo-dimension.* We define pseudo-dimension below for an abstract class $\mathcal{A}$ of functions mapping a domain $\mathcal{X}$ to $[-1, 1]$.

We now use Theorem 2.1.3 to prove that the error term $\epsilon_{\mathcal{M}}(N, \delta)$ in Equation (9.2) converges to zero as N increases. To this end, for any mechanism M, any agent $i \in [n]$, and any pair of types $\theta_i, \hat{\theta}_i \in \Theta_i$, let $u_{i,M,\theta_i,\hat{\theta}_i} : \times_{j \neq i} \Theta_j \to [-1, 1]$ be a function that maps the types $\boldsymbol{\theta}_{-i}$ of the other agents to the utility of agent i with type θ_i and reported type $\hat{\theta}_i$ when the other agents report their types truthfully. In other words, $u_{i,M,\theta_i,\hat{\theta}_i}(\boldsymbol{\theta}_{-i}) = u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$. Let $\mathcal{F}_{i,\mathcal{M}}$ be the set of all such functions defined by mechanisms M from the class $\mathcal{M}$. In other words, $\mathcal{F}_{i,\mathcal{M}} = \left\{ u_{i,M,\theta_i,\hat{\theta}_i} \middle| \, \theta_i, \hat{\theta}_i \in \Theta_i, M \in \mathcal{M} \right\}$. We now analyze the convergence rate of the error term $\epsilon_{\mathcal{M}}(N, \delta)$.

Theorem 9.2.1. *With probability $1 - \delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{ \boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)} \right\} \sim \mathcal{D}_{-i}^{N}$, for all mechanisms $M \in \mathcal{M}$ and agents $i \in [n]$,*

$$\begin{aligned} &\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}} \left[u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right) \right] \right\} \\ &\leq \max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + \epsilon_{\mathcal{M}}(N, \delta), \end{aligned}$$

where $\epsilon_{\mathcal{M}}(N, \delta) = 2\sqrt{\frac{2d_i}{N} \ln \frac{eN}{d_i}} + 2\sqrt{\frac{1}{2N} \ln \frac{2n}{\delta}}$ and $d_i = \operatorname{Pdim}(\mathcal{F}_{i,\mathcal{M}})$.

9.2.1 Incentive compatibility guarantees via finite covers

In the previous section, we presented an empirical estimate of the *ex-interim* incentive compatibility approximation factor (Equation (9.1)) and we showed that it quickly converges to the true

Algorithm 4 Greedy cover construction

Input: Mechanism $M \in \mathcal{M}$, set of samples $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\}$, accuracy parameter $\epsilon > 0$.

1: Let U be the set of vectors $U \leftarrow \left\{ \frac{1}{N} \begin{pmatrix} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \\ \vdots \\ u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(N)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(N)}\right) \end{pmatrix} : \theta_i, \hat{\theta}_i \in \Theta_i \right\}$.

2: Let $V \leftarrow \emptyset$ and $\Gamma \leftarrow \emptyset$.

3: **while** $U \setminus \left(\bigcup_{v \in V} B_1(\boldsymbol{v}, \epsilon)\right) \neq \emptyset$ **do**

4: Select an arbitrary vector $\boldsymbol{v}' \in U \setminus \left(\bigcup_{v \in V} B_1(\boldsymbol{v}, \epsilon)\right)$.

5: Let $\theta_i, \hat{\theta}_i \in \Theta_i$ be the types such that $\boldsymbol{v}' = \frac{1}{N} \begin{pmatrix} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \\ \vdots \\ u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(N)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(N)}\right) \end{pmatrix}$.

6: Add $\boldsymbol{v}'$ to V and $\left(\theta_i, \hat{\theta}_i\right)$ to Γ.

Output: The cover $\Gamma \subseteq \Theta_i \times \Theta_i$.

approximation factor. However, there may not be a finite-time procedure for computing Equation (9.1) in its full generality, restated below:

$$\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\}. \tag{9.3}$$

In this section, we address that challenge. A simple alternative approach is to fix a finite *cover* of $\Theta_i \times \Theta_i$, which we denote as $\Gamma \subset \Theta_i \times \Theta_i$, and approximate Equation (9.3) by measuring the extent to which any agent i can improve his utility by misreporting his type when his true and reported types are elements of the cover Γ, averaged over all profiles in $\mathcal{S}_{-i}$. In other words, we estimate Equation (9.3) as:

$$\max_{\left(\theta_i, \hat{\theta}_i\right) \in \Gamma} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\}. \tag{9.4}$$

This raises two natural questions: how do we select the cover Γ and how close are the optimal solutions to Equations (9.3) and (9.4)? We provide two simple, intuitive approaches to selecting the cover Γ. The first is to run a greedy procedure (see Section 9.2.1) and the second is to create a uniform grid over the type space (assuming $\Theta_i = [0,1]^m$ for some integer m; see Section 9.2.1).

Covering via a greedy procedure

In this section, we show how to construct the cover Γ of $\Theta_i \times \Theta_i$ greedily, based off a classic learning-theoretic algorithm. We then show that when we use the cover Γ to estimate the incentive compatibility approximation factor (via Equation (9.4)), the estimate quickly converges to the true approximation factor. This greedy procedure is summarized by Algorithm 4. For any $\boldsymbol{v} \in \mathbb{R}^N$, we use the notation $B_1(\boldsymbol{v}, \epsilon) = \{\boldsymbol{v}' : \|\boldsymbol{v}' - \boldsymbol{v}\|_1 \leq \epsilon\}$. To simplify notation, let U be the

set of vectors defined in Algorithm 4:

$$U = \left\{ \frac{1}{N} \begin{pmatrix} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \\ \vdots \\ u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(N)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(N)}\right) \end{pmatrix} : \theta_i, \hat{\theta}_i \in \Theta_i \right\}.$$

Note that the solution to Equation (9.3) equals $\max_{v \in U} \sum_{i=1}^{N} v[i]$. The algorithm greedily selects a set of vectors $V \subseteq U$, or equivalently, a set of type pairs $\Gamma \subseteq \Theta_i \times \Theta_i$ as follows: while $U \setminus \left(\bigcup_{v \in V} B_1(\boldsymbol{v}, \epsilon)\right)$ is non-empty, it chooses an arbitrary vector $\boldsymbol{v}'$ in the set, adds it to V, and adds the pair $\left(\theta_i, \hat{\theta}_i\right) \in \Theta_i \times \Theta_i$ defining the vector $\boldsymbol{v}'$ to Γ. Classic results from learning theory [9] guarantee that this greedy procedure will repeat for at most $(8eN/(\epsilon d_i))^{2d_i}$ iterations, where $d_i = \text{Pdim}(\mathcal{F}_{i,\mathcal{M}})$.

We now relate the true incentive compatibility approximation factor to the solution to Equation (9.4) when the cover is constructed using Algorithm 4.

Theorem 9.2.2. *Given a set $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\}$, a mechanism $M \in \mathcal{M}$, and accuracy parameter $\epsilon > 0$, let $\Gamma(\mathcal{S}_{-i}, M, \epsilon)$ be the cover returned by Algorithm 4. With probability $1 - \delta$ over the draw of the n sets $\mathcal{S}_{-i} \sim \mathcal{D}_{-i}^{N}$, for every mechanism $M \in \mathcal{M}$ and every agent $i \in [n]$,*

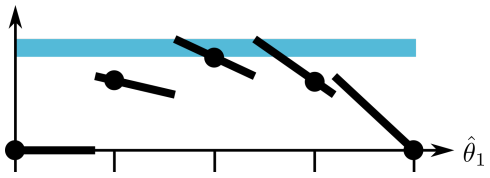
$$\begin{aligned} &\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}} \left[u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right) \right] \right\} \\ \leq &\max_{\left(\theta_i, \hat{\theta}_i\right) \in \Gamma(\mathcal{S}_{-i}, M, \epsilon)} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + \epsilon + \tilde{O}\left(\sqrt{\frac{d_i}{N}}\right), \end{aligned} \tag{9.5}$$

where $d_i = \text{Pdim}(\mathcal{F}_{i,\mathcal{M}})$. Moreover, with probability 1, $|\Gamma(\mathcal{S}_{-i}, M, \epsilon)| \leq (8eN/(\epsilon d_i))^{2d_i}$.

Covering via a uniform grid

The greedy approach in Section 9.2.1 is extremely versatile: no matter the type space Θ_i, when we use the resulting cover to estimate the incentive compatibility approximation factor (via Equation (9.4)), the estimate quickly converges to the true approximation factor. However, implementing the greedy procedure (Algorithm 4) might be computationally challenging because at each round, it is necessary to check if $U \setminus \left(\bigcup_{v \in V} B_1(\boldsymbol{v}, \epsilon)\right)$ is nonempty and if so, select a vector from the set. In this section, we propose an alternative, extremely simple approach to selecting a cover Γ: using a uniform grid over the type space. The efficacy of this approach depends on a "niceness" assumption that holds under mild assumptions, as we prove in Section 9.2.2. Throughout this section, we assume that $\Theta_i = [0,1]^m$ for some integer m. We propose covering the type space using a *w-grid* Γ_w over $[0,1]^m$, by which we mean a finite set of vectors in $[0,1]^m$ such that for all $\boldsymbol{p} \in [0,1]^m$, there exists a vector $\boldsymbol{p}' \in \Gamma_w$ such that $\|\boldsymbol{p} - \boldsymbol{p}'\|_1 \leq w$. For example, if $m = 1$, we could define $\Gamma_w = \left\{0, \frac{1}{\lfloor 1/w \rfloor}, \frac{2}{\lfloor 1/w \rfloor}, \ldots, 1\right\}$. We will estimate the expected incentive compatibility approximation factor using Equation (9.4) with $\Gamma = \Gamma_w \times \Gamma_w$. Throughout the rest of this chapter, we discuss how the choice of w effects the error bound. To do so, we will use the notion of *dispersion*, defined below.

Definition 9.2.3 (Balcan et al. [22])**.** Let $a_1, \ldots, a_N : \mathbb{R}^d \to \mathbb{R}$ be a set of functions where each a_i is piecewise Lipschitz with respect to the ℓ_1-norm over a partition $\mathcal{P}_i$ of $\mathbb{R}^d$. We say that $\mathcal{P}_i$ splits



(a) Average of dispersed functions.

(b) Average of functions that are not dispersed.

Figure 9.1: Illustration of Example 9.2.4. The lines in Figure 9.1a depict the average of four utility functions corresponding to the first-price auction. The maximum of this average falls at the top of the blue region. We evaluate the function in Figure 9.1a on the grid $\left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\}$, as depicted by the dots. The maximum over the grid falls at the bottom of the blue region. In Figure 9.1b, we illustrate the same concepts but for a different set of utility functions that are not as dispersed as those in Figure 9.1a. As illustrated by the blue regions, the approximation over the grid is better for the dispersed functions than the non-dispersed functions.

a set $A \subseteq \mathbb{R}^d$ if A intersects with at least two sets in $\mathcal{P}_i$. The set of functions is (w,k)-*dispersed* if for every point $\boldsymbol{p} \in \mathbb{R}^d$, the ball $\left\{\boldsymbol{p}' \in \mathbb{R}^d : \|\boldsymbol{p} - \boldsymbol{p}'\|_1 \leq w\right\}$ is split by at most k of the partitions $\mathcal{P}_1, \ldots, \mathcal{P}_N$.

The smaller w is and the larger k is, the more "dispersed" the functions' discontinuities are. Moreover, the more jump discontinuities a set of functions has, the more difficult it is to approximately optimize its average using a grid. We illustrate this phenomenon in Example 9.2.4.

Example 9.2.4. Suppose there are two agents and M is the first-price single-item auction. For any trio of types $\theta_1, \theta_2, \hat{\theta}_1 \in [0,1]$, $u_{1,M}\left(\theta_1, \hat{\theta}_1, \theta_2\right) = \mathbf{1}_{\left\{\hat{\theta}_1 > \theta_2\right\}}\left(\theta_1 - \hat{\theta}_1\right)$. Suppose $\theta_1 = 1$. Figure 9.1a displays the function $\frac{1}{4}\sum_{\theta_2 \in \mathcal{S}_2} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right)$ where $\mathcal{S}_2 = \left\{\frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}\right\}$ and Figure 9.1b displays the function $\frac{1}{4}\sum_{\theta_2 \in \mathcal{S}_2'} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right)$ where $\mathcal{S}_2' = \left\{\frac{31}{40}, \frac{32}{40}, \frac{33}{40}, \frac{34}{40}\right\}$.

In Figure 9.1, we evaluate each function on the grid $\Gamma = \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\}$. In Figure 9.1a, the maximum over Γ better approximates the the maximum over $[0,1]$ compared to Figure 9.1b. In other words,

$$\max_{\hat{\theta}_1 \in [0,1]} \sum_{\theta_2 \in \mathcal{S}_2} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right) - \max_{\hat{\theta}_1 \in \Gamma} \sum_{\theta_2 \in \mathcal{S}_2} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right) \tag{9.6}$$

$$< \max_{\hat{\theta}_1 \in [0,1]} \sum_{\theta_2 \in \mathcal{S}_2'} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right) - \max_{\hat{\theta}_1 \in \Gamma} \sum_{\theta_2 \in \mathcal{S}_2'} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right). \tag{9.7}$$

Intuitively, this is because the functions we average over in Figure 9.1a are more dispersed than the functions we average over in Figure 9.1b. The differences described by Equations (9.6) and (9.7) are represented by the shaded regions in Figures 9.1a and 9.1b, respectively.

We now state a helpful lemma which we use to prove this section's main theorem. Informally, it shows that we can measure the average amount that any agent can improve his utility by misreporting his type, even if we discretize his type space, so long as his utility function applied to the samples demonstrates dispersion.

Lemma 9.2.5. *Suppose that for each agent $i \in [n]$, there exist $L_i, k_i, w_i \in \mathbb{R}$ such that with probability $1 - \delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^N$, for each mechanism $M \in \mathcal{M}$ and agent $i \in [n]$, the following conditions hold:*

1. *For any type* $\boldsymbol{\theta}_i \in [0,1]^m$*, the functions* $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ *are piecewise* L_i*-Lipschitz and* (w_i, k_i)*-dispersed.*

2. *For any reported type* $\hat{\boldsymbol{\theta}}_i \in [0,1]^m$*, the functions* $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(N)}\right)$ *are piecewise* L_i*-Lipschitz and* (w_i, k_i)*-dispersed.*

Then with probability $1-\delta$ *over the draw of the* n *sets* $\mathcal{S}_{-i} \sim \mathcal{D}_{-i}^N$*, for all mechanisms* $M \in \mathcal{M}$ *and agents* $i \in [n]$*,*

$$\begin{aligned}
&\max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in [0,1]^m} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} \\
&\leq \max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in \Gamma_{w_i}} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + 4L_i w_i + \frac{8k_i}{N}. \qquad (9.8)
\end{aligned}$$

In Section 9.2.2, we prove that under mild assumptions, for a wide range of mechanisms, Conditions 1 and 2 in Lemma 9.2.5 hold with $L_i = 1$, $w_i = O\left(1/\sqrt{N}\right)$, and $k_i = \tilde{O}\left(\sqrt{N}\right)$, ignoring problem-specific multiplicands. Thus, we find that $4L_i w_i + 8k_i/N$ quickly converges to 0 as N grows.

Theorem 9.2.1 and Lemma 9.2.5 immediately imply this section's main theorem. At a high level, it states that any agent's average utility gain when restricted to types on the grid is a close estimation of the true incentive compatibility approximation factor, so long as his utility function applied to the samples demonstrates dispersion.

Theorem 9.2.6. *Suppose that for each agent* $i \in [n]$*, there exist* $L_i, k_i, w_i \in \mathbb{R}$ *such that with probability* $1-\delta$ *over the draw of the* n *sets* $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^N$*, for each mechanism* $M \in \mathcal{M}$ *and agent* $i \in [n]$*, the following conditions hold:*

1. *For any* $\boldsymbol{\theta}_i \in [0,1]^m$*, the functions* $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ *are piecewise* L_i*-Lipschitz and* (w_i, k_i)*-dispersed.*

2. *For any* $\hat{\boldsymbol{\theta}}_i \in [0,1]^m$*, the functions* $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(N)}\right)$ *are piecewise* L_i*-Lipschitz and* (w_i, k_i)*-dispersed.*

Then with probability $1-2\delta$*, for every mechanism* $M \in \mathcal{M}$ *and every agent* $i \in [n]$*,*

$$\begin{aligned}
&\max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in \Theta_i} \left\{ \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}} \left[u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}\right) \right] \right\} \\
&\leq \max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in \Gamma_{w_i}} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + \epsilon,
\end{aligned}$$

where $\epsilon = 4L_i w_i + \frac{8k_i}{N} + 2\sqrt{\frac{2d_i}{N} \ln \frac{eN}{d_i}} + 2\sqrt{\frac{1}{2N} \ln \frac{2n}{\delta}}$ *and* $d_i = \operatorname{Pdim}\left(\mathcal{F}_{i,\mathcal{M}}\right)$*.*

We conclude with one final comparison of the greedy approach in Section 9.2.1 and the uniform grid approach in this section. In Section 9.2.1, we use the functional form of the utility functions in order to bound the cover size, as quantified by pseudo-dimension. When we use the approach based on dispersion, we do not use these functional forms to the fullest extent possible; we only use simple facts about the functions' discontinuities and Lipschitzness.

Mechanism(s)	Upper bound on estimation error $\lvert\hat{\gamma} - \gamma\rvert$ based on dispersion
First-price single-item auction	$\tilde{O}\left(\frac{\kappa^{-1}+n}{\sqrt{N}}\right)$
First-price combinatorial auction over ℓ items	$\tilde{O}\left(\frac{\kappa^{-1}+(n+1)^{2\ell}\sqrt{\ell}}{\sqrt{N}}\right)$
Generalized second-price auction	$\tilde{O}\left(\frac{\kappa^{-1}+n^{3/2}}{\sqrt{N}}\right)$
Discriminatory auction over m units of a single good	$\tilde{O}\left(\frac{\kappa^{-1}+nm^2}{\sqrt{N}}\right)$
Uniform-price auction over m units of a single good	$\tilde{O}\left(\frac{\kappa^{-1}+nm^2}{\sqrt{N}}\right)$
Second-price auction with spiteful bidders	$\tilde{O}\left(\frac{\kappa^{-1}+n}{\sqrt{N}}\right)$

Table 9.1: Our estimation error upper bounds based on dispersion. The value κ is defined such that $[0, \kappa]$ contains the range of the density functions defining the agents' type distribution.

9.2.2 Dispersion and pseudo-dimension guarantees

We now provide dispersion and pseudo-dimension guarantees for a variety of mechanism classes. The theorems in this section allow us to instantiate the bounds from the previous section and thus understand how well our empirical incentive compatibility approximation factor matches the true approximation factor. We summarize our guarantees in Table 9.1.

Given an agent $i \in [n]$, a true type $\boldsymbol{\theta}_i \in [0,1]^m$, and a set of samples $\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)} \sim \mathcal{D}_{-i}$, we often find that the discontinuities of each function $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(j)}\right)$ are highly dependent on the vector $\boldsymbol{\theta}_{-i}^{(j)}$. For example, under the first-price single-item auction, the discontinuities of the function $u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(j)}\right)$ occur when $\hat{\theta}_i = \left\|\boldsymbol{\theta}_{-i}^{(j)}\right\|_\infty$. Since each vector $\boldsymbol{\theta}_{-i}^{(j)}$ is a random draw from the distribution $\mathcal{D}_{-i}$, these functions will not be dispersed if these random draws are highly-concentrated. For this reason, we focus on type distributions with *κ-bounded density functions*. A density function $\phi : \mathbb{R} \to \mathbb{R}$ is κ-bounded if $\max\{\phi(x)\} \leq \kappa$. For example, the density function of any normal distribution with standard deviation $\sigma \in \mathbb{R}$ is $\frac{1}{\sigma\sqrt{2\pi}}$-bounded.

The challenge we face in this section is that it is not enough to show that for some $\boldsymbol{\theta}_i \in [0,1]^m$, the functions $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are dispersed. Rather, we must prove that for all type vectors, the dispersion property holds. This facet of our analysis is notably different from prior work by Balcan et al. [22]: in their applications, it is enough to show that with high probability, a single, finite sequence of functions is dispersed. In contrast, we show that under mild assumptions, with high probability, each function sequence from an infinite family is dispersed.

In a bit more detail, there are two types of sequences we must prove are dispersed. The first are defined by functions over reported types $\hat{\boldsymbol{\theta}}_i$, with one sequence for each true type $\boldsymbol{\theta}_i$; the second are defined by functions over true types $\boldsymbol{\theta}_i$ with one sequence for each reported type $\hat{\boldsymbol{\theta}}_i$. In other words, sequences of the first type have the form $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ and sequences of the second type have the form $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(N)}\right)$. For the first

family of sequences, we show that discontinuities occur only when agent i's shift in her reported type causes the allocation to change. These change-points have nothing to do with the true type $\boldsymbol{\theta}_i$ (which the mechanism does not see), so all the sequences have the same discontinuities (see, for example in Lemma 9.2.7). As a result, it is enough prove dispersion for a single generic sequence, defined by an arbitrary $\boldsymbol{\theta}_i$—as we do, for example, in Theorem 9.2.8—in order to guarantee dispersion for all of the sequences. Throughout our applications, the second type of sequence is even simpler. These functions have no discontinuities at all, because the allocation is invariant as we vary the true type $\boldsymbol{\theta}_i$ (because, again, the mechanism does not see $\boldsymbol{\theta}_i$). Therefore, these functions are either constant if the player was not allocated anything, or linear otherwise (see, for example, Theorem 9.2.9). As a result, all sequences from this second family are immediately dispersed.

First-price single-item auction

To develop intuition, we begin by analyzing the first-price auction for a single item. We then analyze the first-price combinatorial auction. Under this auction, each agent $i \in [n]$ has a value $\theta_i \in [0,1]$ for the item and submits a bid $\hat{\theta}_i \in [0,1]$. The agent with the highest bid wins the item and pays his bid. Therefore, agent i's utility function is $u_{i,M}\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}\right) = \mathbf{1}_{\left\{\hat{\theta}_i > \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right\}}\left(\theta_i - \hat{\theta}_i\right)$.

To prove our dispersion guarantees, we begin with the following helpful lemma.

Lemma 9.2.7. *For any agent $i \in [n]$, any set $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\}$, and any type $\theta_i \in [0,1]$, the functions $u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are piecewise 1-Lipschitz and their discontinuities fall in the set $\left\{\left\|\boldsymbol{\theta}_{-i}^{(j)}\right\|_\infty\right\}_{j \in [N]}$.*

We now use Lemma 9.2.7 to prove our first dispersion guarantee.

Theorem 9.2.8. *Suppose each agent's type has a κ-bounded density function. With probability $1 - \delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^N$, we have that for all agents $i \in [n]$ and types $\theta_i \in [0,1]$, the functions $u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are piecewise 1-Lipschitz and $\left(O\left(1/\left(\kappa\sqrt{N}\right)\right), \tilde{O}\left(n\sqrt{N}\right)\right)$-dispersed.*

Theorem 9.2.9. *For all agents $i \in [n]$, reported types $\hat{\theta}_i \in [0,1]$, and type profiles $\boldsymbol{\theta}_{-i} \in [0,1]^{n-1}$, the function $u_{i,M}\left(\cdot, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz.*

Next, we prove the following pseudo-dimension bound.

Theorem 9.2.10. *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is 2.*

First-price combinatorial auction

Under this auction, there are ℓ items for sale and each agent's type $\boldsymbol{\theta}_i \in [0,1]^{2^\ell}$ indicates his value for each bundle $b \subseteq [\ell]$. We denote his value and bid for bundle b as $\theta_i(b)$ and $\hat{\theta}_i(b)$, respectively. The allocation $(b_1^*, \ldots, b_n^*)$ is the solution to the *winner determination problem*:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^n \hat{\theta}_i\left(b_i\right) \\ \text{subject to} & b_i \cap b_{i'} = \emptyset \quad \forall i, i' \in [n], i \neq i'. \end{array}$$

Each agent $i \in [n]$ pays $\hat{\theta}_i\left(b_i^*\right)$.

We begin with dispersion guarantees.

Theorem 9.2.11. *Suppose that for each pair of agents $i, i' \in [n]$ and each pair of bundles $b, b' \subseteq [\ell]$, the values $\theta_i(b)$ and $\theta_{i'}(b')$ have a κ-bounded joint density function. With probability $1-\delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^{N}$, we have that for all agents $i \in [n]$ and types $\boldsymbol{\theta}_i \in [0,1]^{2^\ell}$, the functions $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are piecewise 1-Lipschitz and $\left(O\left(1/\left(\kappa\sqrt{N}\right)\right), \tilde{O}\left((n+1)^{2^\ell}\sqrt{N\ell}\right)\right)$-dispersed.*

Theorem 9.2.12. *For all agents $i \in [n]$, reported types $\hat{\boldsymbol{\theta}}_i \in [0,1]^{2^\ell}$, and type profiles $\boldsymbol{\theta}_{-i} \in [0,1]^{(n-1)2^\ell}$, the function $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz.*

Next, we prove the following pseudo-dimension bound.

Theorem 9.2.13. *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is $O\left(\ell 2^\ell \log n\right)$.*

Generalized second-price auction

A *generalized second-price auction* allocates m advertising slots to a set of $n > m$ agents. Each slot s has a probability $\alpha_{s,i}$ of being clicked if agent i's advertisement is in that slot. We assume $\alpha_{s,i}$ is fixed and known by the mechanism designer. The mechanism designer assigns a weight $\omega_i \in (0,1]$ per agent i. Each agent has a value $\theta_i \in [0,1]$ for a click and submits a bid $\hat{\theta}_i \in [0,1]$. The mechanism allocates the first slot to the agent with the highest weighted bid $\omega_i\hat{\theta}_i$, the second slot to the agent with the second highest weighted bid, and so on. Let $\pi(s)$ be the agent allocated slot s. If slot s is clicked on, agent $\pi(s)$ pays the lowest amount that would have given him slot s, which is $\omega_{\pi(s+1)}\hat{\theta}_{\pi(s+1)}/\omega_{\pi(s)}$. Agent $\pi(s)$'s expected utility is thus $u_{\pi(s),M}\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}\right) = \alpha_{s,\pi(s)}\left(\theta_{\pi(s)} - \omega_{\pi(s+1)}\hat{\theta}_{\pi(s+1)}/\omega_{\pi(s)}\right)$.

For $r \in \mathbb{Z}_{\geq 1}$, let $\mathcal{M}_r$ be the set of auctions defined by agent weights from the set $\left\{\frac{1}{r}, \frac{2}{r}, \ldots, 1\right\}$. We begin by proving dispersion guarantees.

Theorem 9.2.14. *Suppose each agent's type has a κ-bounded density function. With probability $1-\delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^{N}$, we have that for all agents $i \in [n]$, types $\theta_i \in [0,1]$, and mechanisms $M \in \mathcal{M}_r$, the functions $u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are piecewise 0-Lipschitz and $\left(O\left(1/\left(r\kappa\sqrt{N}\right)\right), \tilde{O}\left(\sqrt{n^3 N}\right)\right)$-dispersed.*

Theorem 9.2.15. *For all agents $i \in [n]$, all reported types $\hat{\theta}_i \in [0,1]$, all type profiles $\boldsymbol{\theta}_{-i} \in [0,1]^{n-1}$, and all generalized second-price auctions M, the function $u_{i,M}\left(\cdot, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz.*

We now provide the following pseudo-dimension guarantee.

Theorem 9.2.16. *For any agent $i \in [n]$ and $r \in \mathbb{Z}_{\geq 1}$, $\operatorname{Pdim}\left(\mathcal{F}_{i,\mathcal{M}_r}\right) = O\left(n \log n\right)$.*

Discriminatory auction

Under the discriminatory auction, there are m identical units of a single item for sale. For each agent $i \in [n]$, his type $\boldsymbol{\theta}_i \in [0,1]^m$ indicates how much he is willing to pay for each additional unit. Thus, $\theta_i[1]$ is the amount he is willing to pay for one unit, $\theta_i[1] + \theta_i[2]$ is the amount he is willing to pay for two units, and so on. We assume that $\theta_i[1] \geq \theta_i[2] \geq \cdots \geq \theta_i[m]$. The auctioneer collects nm bids $\hat{\theta}_i[\mu]$ for $i \in [n]$ and $\mu \in [m]$. If exactly m_i of agent i's bids are among the m highest of all nm bids, then agent i is awarded m_i units and pays $\sum_{\mu=1}^{m_i}\hat{\theta}_i[\mu]$.

We begin with dispersion guarantees.

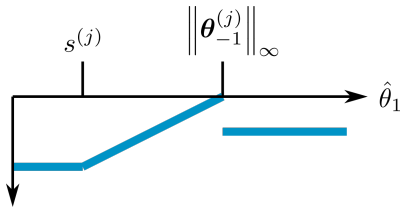


Figure 9.2: Graph of $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ for a spiteful agent under a second-price auction with $\alpha_1 = \theta_1 = \frac{1}{2}$, $\|\boldsymbol{\theta}_{-1}\|_\infty = \frac{3}{4}$, and $s^{(j)} = \frac{1}{4}$.

Theorem 9.2.17. *Suppose that each agent's value for each marginal unit has a κ-bounded density function. With probability $1-\delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^N$, for all agents $i \in [n]$ and types $\boldsymbol{\theta}_i \in [0,1]^m$, the functions $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are piecewise 1-Lipschitz and $\left(O\left(1/\left(\kappa\sqrt{N}\right)\right), \tilde{O}\left(nm^2\sqrt{N}\right)\right)$-dispersed.*

Theorem 9.2.18. *For all agents $i \in [n]$, reported types $\hat{\boldsymbol{\theta}}_i \in [0,1]^m$, and type profiles $\boldsymbol{\theta}_{-i} \in [0,1]^{(n-1)m}$, the function $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz.*

Next, we prove the following pseudo-dimension bound.

Theorem 9.2.19. *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is $O\left(m \log(nm)\right)$.*

Uniform-price auction

Under this auction, the allocation rule is the same as in the discriminatory auction. However, all m units are sold at a "market-clearing" price, meaning the total amount demanded equals the total amount supplied. We obtain the same bounds as we do for the discriminatory auction.

Second-price auction with spiteful agents

We conclude by studying *spiteful agents* [38, 144, 177, 180], where each bidder's utility not only increases when his surplus increases, but also decreases when the other bidders' surpluses increase. Formally, given a *spite parameter* $\alpha_i \in [0,1]$, agent i's utility under the second-price auction M is $u_{i,M}\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}\right) = \alpha_i \mathbf{1}_{\left\{\hat{\theta}_i > \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right\}}\left(\theta_i - \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right) - (1-\alpha_i)\sum_{i' \neq i} \mathbf{1}_{\left\{\hat{\theta}_{i'} > \left\|\hat{\boldsymbol{\theta}}_{-i'}\right\|_\infty\right\}}\left(\theta_{i'} - \left\|\hat{\boldsymbol{\theta}}_{-i'}\right\|_\infty\right)$. The closer α_i is to zero, the more spiteful bidder i is.

We begin with the following lemma, which follows from the form of $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ (see Figure 9.2).

Lemma 9.2.20. *For any agent $i \in [n]$, any $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\}$, and any type $\theta_i \in [0,1]$, the functions $u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ are piecewise 1-Lipschitz and their discontinuities fall in the set $\left\{\left\|\boldsymbol{\theta}_{-i}^{(j)}\right\|_\infty\right\}_{j \in [N]}$.*

Lemma 9.2.20 implies the following dispersion guarantee.

Theorem 9.2.21. *Suppose each agent's type has a κ-bounded density function. With probability $1-\delta$ over the draw of the n sets $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)}\right\} \sim \mathcal{D}_{-i}^N$, we have that for all agents $i \in [n]$*

and types $\theta_i \in [0,1]$*, the functions* $u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(1)}\right), \ldots, u_{i,M}\left(\theta_i, \cdot, \boldsymbol{\theta}_{-i}^{(N)}\right)$ *are piecewise* 1*-Lipschitz and* $\left(O\left(1/\left(\kappa\sqrt{N}\right)\right), \tilde{O}\left(n\sqrt{N}\right)\right)$*-dispersed.*

Theorem 9.2.22. *For all agents* $i \in [n]$*, reported types* $\hat{\theta}_i \in [0,1]$*, and type profiles* $\boldsymbol{\theta}_{-i} \in [0,1]^{n-1}$*, the function* $u_{i,M}\left(\cdot, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ *is* 1*-Lipschitz.*

We conclude with the following pseudo-dimension bound.

Theorem 9.2.23. *For any agent* $i \in [n]$*, the pseudo-dimension of the class* $\mathcal{F}_{i,M}$ *is* $O(1)$*.*

Part II

Learning algorithms

Chapter 10

Batch learning algorithms

Recently, two lines of research have emerged that explore the theoretical underpinnings of algorithm configuration. One—which we covered extensively in Part I—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 [20, 21, 89]. 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 [109, 110, 185, 186]. 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: in Chapter 3, we presented 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 10.4.

This chapter 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 [109, 110, 185, 186], 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, in Chapter 3, we 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 [36]. Animation studios with thousands of meshes require carefully tuned solvers which, thus far, domain

experts have handcrafted [37]. Our algorithm can be used to find configurations that minimize the branch-and-bound tree size. Balcan et al. [20] 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 [136]. 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 $\ell_{\boldsymbol{\rho}}(z)$ measures the computational resources (running time, for example) required to solve problem instance z using the algorithm parameterized by the vector $\boldsymbol{\rho}$. Let OPT be the smallest expected loss[1] $\mathbb{E}_{z\sim\mathcal{D}}\left[\ell_{\boldsymbol{\rho}}(z)\right]$ of any parameter $\boldsymbol{\rho}$, where $\mathcal{D}$ 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 10.0.1 (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.

The results in this section are joint work with Nina Balcan and Tuomas Sandholm [28]. Our current results were published in AAAI 2020.

10.1 Problem definition

The algorithm configuration model we adopt is a generalization of the model from prior research by Kleinberg et al. [109, 110] and Weisz et al. [185, 186]. There is a set $\mathcal{Z}$ of problem instances and an unknown distribution $\mathcal{D}$ over $\mathcal{Z}$. 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 $U:\mathcal{P}\times\mathcal{Z}\times\mathbb{Z}_{\geq 0}\rightarrow$

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

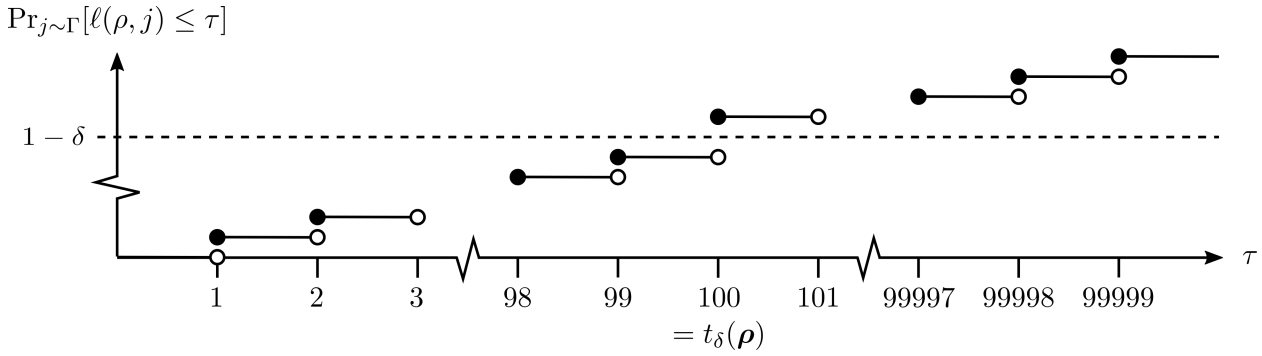


Figure 10.1: Fix a parameter vector $\boldsymbol{\rho}$. The figure is a hypothetical illustration of the cumulative density function of $\ell_{\boldsymbol{\rho}}(z)$ when z is sampled from $\mathcal{D}$. For each value τ along the x-axis, the solid line equals $\Pr_{z\sim\mathcal{D}}\left[\ell_{\boldsymbol{\rho}}(z) \leq \tau\right]$. The dotted line equals the constant function $1 - \delta$. Since 100 is the largest integer such that $\Pr_{z\sim\mathcal{D}}\left[\ell_{\boldsymbol{\rho}}(z) \geq 100\right] \geq \delta$, we have that $t_\delta(\boldsymbol{\rho}) = 100$.

$\{0, 1\}$, where $U(\boldsymbol{\rho}, z, \tau) = 1$ if and only if the algorithm parameterized by $\boldsymbol{\rho}$ returns a solution to the instance z given a budget of τ. We make the natural assumption that the algorithm is more likely to find a solution the higher its budget: $U(\boldsymbol{\rho}, z, \tau) \geq U(\boldsymbol{\rho}, z, \tau')$ for $\tau \geq \tau'$. Finally, for every parameter vector $\boldsymbol{\rho} \in \mathcal{P}$, there is a loss function $\ell_{\boldsymbol{\rho}} : \mathcal{Z} \to \mathbb{Z}_{\geq 0}$ which measures the minimum budget the algorithm requires to find a solution. Specifically, $\ell_{\boldsymbol{\rho}}(z) = \infty$ if $U(\boldsymbol{\rho}, z, \tau) = 0$ for all τ, and otherwise, $\ell_{\boldsymbol{\rho}}(z) = \operatorname{argmin}\{\tau : U(\boldsymbol{\rho}, z, \tau) = 1\}$. In Section 10.1.1, we provide several examples of this problem definition instantiated for combinatorial problems.

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

$$\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\ell_{\hat{\boldsymbol{\rho}}}(z)\right] \leq (1+\epsilon) \inf_{\boldsymbol{\rho}\in\mathcal{P}} \left\{ \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\ell_{\boldsymbol{\rho}}(z)\right] \right\}. \tag{10.1}$$

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 10.1.1. [Weisz et al. [186]] 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: $\ell_1(z) = 2$ for all $z \in \mathcal{Z}$. Meanwhile, suppose that $\ell_2(z) = \infty$ with probability δ for some $\delta \in (0, 1)$ and $\ell_2(z) = 1$ with probability $1 - \delta$. In this case, $\mathbb{E}_{z\sim\mathcal{D}}[\ell_1(z)] = 2$ and $\mathbb{E}_{z\sim\mathcal{D}}[\ell_2(z)] = \infty$. 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 z such that $\ell_2(z) = \infty$. 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 (10.1).

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. [185, 186], building off of work by Kleinberg et al. [109, 110], 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 $\ell_{\boldsymbol{\rho}}(z)$ is greater than τ is at least δ. Mathematically, $t_\delta(\boldsymbol{\rho}) = \operatorname{argmax}_{\tau\in\mathbb{Z}}\left\{\Pr_{z\sim\mathcal{D}}[\ell_{\boldsymbol{\rho}}(z) \geq \tau] \geq \delta\right\}$. 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 10.1. We now define the relaxed notion of approximate optimality by Weisz et al. [185].

Definition 10.1.2 ($(\epsilon, \delta, \mathcal{P})$-optimality)**.** A parameter vector $\hat{\boldsymbol{\rho}}$ is $(\epsilon, \delta, \mathcal{P})$-*optimal* if

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

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 $OPT_{c\delta} := \inf_{\boldsymbol{\rho}\in\mathcal{P}}\left\{\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell_{\boldsymbol{\rho}}(z), t_{c\delta}(\boldsymbol{\rho})\right\}\right]\right\}$. 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. [186] provide one such algorithm, CapsAndRuns.

10.1.1 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 [17, 21, 67, 95, 101, 107, 108, 117, 190]. Given parameters $\boldsymbol{\rho}$ and a problem instance z, 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 $u(\boldsymbol{\rho}, z, \tau) = 1$ if and only if the algorithm terminates, having found the optimal solution, after building a tree of size τ. The loss $\ell_{\boldsymbol{\rho}}(z)$ equals the size of the entire tree built by the algorithm parameterized by $\boldsymbol{\rho}$ given the instance z 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. [20] 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:

[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 [185].

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 [12, 166, 187] and come with theoretical guarantees. Balcan et al. [20] 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: the best-known algorithm's runtime is $O(n^2 \log n)$, where n is the number of datapoints [136]. 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 $c : \mathcal{P} \times \mathcal{Z} \times \mathbb{Z} \to \mathbb{R}$ where $c(\rho, z, \tau)$ measures the quality of the clustering τ-capped ρ-linkage returns, given the instance z as input. We assume there is a threshold θ_z where the clustering is admissible if and only if $c(\rho, z, \tau) \leq \theta_z$, which means the utility function is defined as $U(\rho, z, \tau) = \mathbf{1}_{\{c(\rho,z,\tau)\leq\theta_z\}}$. For example, $c(\rho, z, \tau)$ might measure the clustering's k-means objective value, and θ_z might equal the optimal k-means objective value (obtained only for the training instances via an expensive computation) plus an error term.

10.2 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 [186], 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 10.2.1 (**(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 $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell_{\hat{\boldsymbol{\rho}}}(z), t_{\delta/2}\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \leq \sqrt{1+\epsilon} \cdot OPT_{\delta/4}$.

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 10.2.2. *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 $(\epsilon', \delta, \bar{\mathcal{P}})$-optimal. Then $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell_{\hat{\boldsymbol{\rho}}}(z), t_{\delta}\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \leq (1+\epsilon) \cdot OPT_{\delta/4}$.*

10.3 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 z, the loss function is a piecewise-constant function of the parameters. This structure has been observed throughout a diverse array of configuration problems, as we demonstrated in Part I. More formally, this structure holds if for any problem instance $z \in \mathcal{Z}$ and cap $\tau \in \mathbb{Z}_{\geq 0}$, 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 $\boldsymbol{\rho}, \boldsymbol{\rho}' \in R$, $\min\left\{\ell_{\boldsymbol{\rho}}(z), \tau\right\} = \min\left\{\ell_{\boldsymbol{\rho}'}(z), \tau\right\}$.

Algorithm 5 Algorithm for learning (ϵ, δ)-optimal subsets

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

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 $\mathcal{G} \leftarrow \emptyset$.

while $2^{t-3}\delta < T$ **do**

  Set $\mathcal{S}_t \leftarrow \{z\}$, where $z \sim \mathcal{D}$.

  while $\eta\delta$ is smaller than

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

  do Draw $z \sim \mathcal{D}$ and add z to $\mathcal{S}_t$.

  Compute tuples $\left(\mathcal{P}_1, r_1, \boldsymbol{\tau}_1\right), \ldots, \left(\mathcal{P}_k, r_k, \boldsymbol{\tau}_k\right) \leftarrow \textsc{GetPartition}\left(\mathcal{S}_t, 2^t\right)$.

  for $i \in \{1, \ldots, k\}$ with $r_i \geq 1 - 3\delta/8$ **do**

    Set $\mathcal{G} \leftarrow \mathcal{G} \cup \{\mathcal{P}_i\}$.

    Sort the elements of $\boldsymbol{\tau}_i$: $\tau_1 \leq \cdots \leq \tau_{|\mathcal{S}_t|}$.

    Set $T' \leftarrow \frac{1}{|\mathcal{S}_t|} \sum_{m=1}^{|\mathcal{S}_t|} \min\left\{\tau_m, \tau_{\lfloor |\mathcal{S}_t|(1-3\delta/8)\rfloor}\right\}$.

    if $T' < T$ **then** Set $T \leftarrow T'$.

  $t \leftarrow t + 1$.

For each set $\mathcal{P}' \in \mathcal{G}$, select a vector $\boldsymbol{\rho}_{\mathcal{P}'} \in \mathcal{P}'$.

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

To exploit this piecewise-constant structure, we require access to a function GetPartition 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 $\left(\mathcal{P}_1, r_1, \boldsymbol{\tau}_1\right), \ldots, \left(\mathcal{P}_k, r_k, \boldsymbol{\tau}_k\right) \in 2^{\mathcal{P}} \times [0,1] \times \mathbb{Z}^{|\mathcal{S}|}$ 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 $\boldsymbol{\rho}, \boldsymbol{\rho'} \in \mathcal{P}_i$, $\frac{1}{|\mathcal{S}|} \sum_{z \in \mathcal{S}} \mathbf{1}_{\left\{\ell_{\boldsymbol{\rho}}(z) \leq \tau\right\}} = \frac{1}{|\mathcal{S}|} \sum_{z \in \mathcal{S}} \mathbf{1}_{\left\{\ell_{\boldsymbol{\rho}'}(z) \leq \tau\right\}} = r_i$.
3. For all subsets $\mathcal{P}_i$, all $\boldsymbol{\rho}, \boldsymbol{\rho'} \in \mathcal{P}_i$, and all $z \in \mathcal{S}$, $\min\left\{\ell_{\boldsymbol{\rho}}(z), \tau\right\} = \min\left\{\ell_{\boldsymbol{\rho}'}(z), \tau\right\} = \tau_i[j]$.

We assume the number of tuples GetPartition returns is monotone: if $\tau \leq \tau'$, then

$$\left|\textsc{GetPartition}(\mathcal{S}, \tau)\right| \leq \left|\textsc{GetPartition}(\mathcal{S}, \tau')\right|$$

and if $\mathcal{S} \subseteq \mathcal{S}'$, then $\left|\textsc{GetPartition}(\mathcal{S}, \tau)\right| \leq \left|\textsc{GetPartition}(\mathcal{S}', \tau)\right|$.

Results from prior research imply guidance for implementing GetPartition in the contexts of clustering and integer programming. For example, in the clustering application we describe in Section 10.1.1, the distribution $\mathcal{D}$ is over clustering instances. Suppose n is an upper bound on the number of points in each instance. Balcan et al. [20] prove that for any set $\mathcal{S}$ of samples and any cap τ, in the worst case, $\left|\textsc{GetPartition}(\mathcal{S}, \tau)\right| = O\left(|\mathcal{S}| n^8\right)$, though empirically, $\left|\textsc{GetPartition}(\mathcal{S}, \tau)\right|$ is often several orders of magnitude smaller [27]. Balcan et al. [20, 27] provide guidance for implementing GetPartition.

High-level description of algorithm. We now describe our algorithm for learning (ϵ, δ)-optimal subsets. See Algorithm 5 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 $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell_{\boldsymbol{\rho}}(z), 2^t\right\}\right]$ 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* [31, 113].

Next, the algorithm evaluates the function GetPartition $\left(\mathcal{S}_t, 2^t\right)$ to obtain the tuples

$$\left(\mathcal{P}_1, r_1, \boldsymbol{\tau}_1\right), \ldots, \left(\mathcal{P}_k, r_k, \boldsymbol{\tau}_k\right) \in 2^{\mathcal{P}} \times [0,1] \times \mathbb{Z}^{|\mathcal{S}_t|}.$$

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 $z \in \mathcal{S}_t$ with $\ell_{\boldsymbol{\rho}}(z) \leq 2^t$ is equal to the fraction of instances $z \in \mathcal{S}_t$ with $\ell_{\boldsymbol{\rho}'}(z) \leq 2^t$. In other words, $\frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t} \mathbf{1}_{\left\{\ell_{\boldsymbol{\rho}}(z)\leq 2^t\right\}} = \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t} \mathbf{1}_{\left\{\ell_{\boldsymbol{\rho}'}(z)\leq 2^t\right\}} = r_i$. 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 5'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 10.3.1. *With probability $1 - \zeta$, the following conditions hold, with $\eta = \min\left\{\frac{\left(\sqrt[4]{1+\epsilon}-1\right)}{8}, \frac{1}{9}\right\}$ and $c = \frac{16\sqrt[4]{1+\epsilon}}{\delta}$:*

1. *Algorithm 5 terminates after $\bar{t} = O\left(\log\left(c \cdot OPT_{\delta/4}\right)\right)$ iterations.*
2. *Algorithm 5 returns an (ϵ, δ)-optimal set of parameters of size at most*

$$\sum_{t=1}^{\bar{t}} \left|\textsc{GetPartition}\left(\mathcal{S}_t, c \cdot OPT_{\delta/4}\right)\right|.$$

3. *The sample complexity on round $t \in [\bar{t}]$, $|\mathcal{S}_t|$, is*

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

10.4 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 10.1.1.

Uniformly sampling configurations. Prior research provides algorithms for finding nearly-optimal configurations from a finite set [109, 110, 185, 186]. 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. (This is Theorem 3.3.1, restated here for convenience.)

Theorem 10.4.1. *For any $\frac{1}{3} < a < b < \frac{1}{2}$ and $n \geq 6$, there are infinitely-many distributions $\mathcal{D}$ over IPs with n variables and a B&B parameter with range $[0, 1]$ such that:*

1. *If $\rho \leq a$, then $\ell_\rho(z) = 2^{(n-5)/4}$ with probability $\frac{1}{2}$ and $\ell_\rho(z) = 8$ with probability $\frac{1}{2}$.*
2. *If $\rho \in (a, b)$, then $\ell_\rho(z) = 8$ with probability 1.*
3. *If $\rho \geq b$, then $\ell_\rho(z) = 2^{(n-4)/2}$ with probability $\frac{1}{2}$ and $\ell_\rho(z) = 8$ with probability $\frac{1}{2}$.*

Here, $\ell_\rho(z)$ 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 [109, 110, 185, 186], 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.

Theorem 10.4.2. *For the B&B configuration problem in Theorem 10.4.1, 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 5 quickly terminates, having found an optimal parameter, as we describe below.

Theorem 10.4.3. *For the configuration problem in Theorem 10.4.1, Algorithm 5 terminates after $\tilde{O}\left(\log \frac{1}{\delta}\right)$ 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).*

Similarly, Balcan et al. [23] exemplify clustering configuration problems—which we overview in Section 10.1.1—where the optimal parameters lie within an arbitrarily small region, and any other parameter leads to significantly worse performance. As in Theorem 10.4.2, this means a uniform sampling of the parameters will fail to find optimal parameters.

Chapter 11

Beyond batch learning

Throughout this proposal, we have studied algorithm configuration in the classic batch learning model: there is a distribution over problem instances, and the learning algorithm uses a training set sampled from this distribution to learn a high-performing parameter setting. In this chapter, we study an online approach to algorithm design, where there is no distribution over problem instances. Rather, problem instances arrive one-by-one, and the goal is to use structure shared across instances to gradually speed up the time it takes to find an optimal solution.

Our model is motivated by the following scenario. Consider computing the shortest path from home to work every morning. The shortest path may vary from day to day—sometimes side roads beat the highway; sometimes the bridge is closed due to construction. However, although San Francisco and New York are contained in the same road network, it is unlikely that a San Francisco-area commuter would ever find New York along her shortest path—the edge times in the graph do not change *that* dramatically from day to day.

With this motivation in mind, we study a learning problem where the goal is to speed up repeated computations when the sequence of instances share common substructure. Examples include repeatedly computing the shortest path between the same two nodes on a graph with varying edge weights, repeatedly computing string matchings, and repeatedly solving linear programs with mildly varying objectives. Our work is in the spirit of recent work in batch learning for algorithm configuration (discussed in Part I) and online learning [43], although with some key differences, which we discuss below.

The basis of this work is the observation that for many realistic instances of repeated problems, vast swaths of the search space may never contain an optimal solution—perhaps the shortest path is always contained in a specific region of the road network; large portions of a DNA string may never contain the patterns of interest; a few key linear programming constraints may be the only ones that bind. Algorithms designed to satisfy worst-case guarantees may thus waste substantial computation time on futile searching. For example, even if a single, fixed path from home to work were best every day, Dijkstra's algorithm would consider all nodes within distance d_i from home on day i, where d_i is the length of the optimal path on day i, as illustrated in Figure 11.1.

We develop a simple solution, inspired by online learning, that leverages this observation to the maximal extent possible. On each problem, our algorithm typically searches over a small, pruned subset of the solution space, which it learns over time. This pruning is the minimal subset containing all previously returned solutions. These rounds are analogous to "exploit" rounds in online learning. To learn a good subset, our algorithm occasionally deploys a worst-case-style algorithm, which explores a large part of the solution space and guarantees correctness on any instance. These rounds are analogous to "explore" rounds in online learning. If, for example, a single fixed path were always optimal, our algorithm would almost always immediately output that path, as it would be the only one in its pruned search space. Occasionally, it would run a full Dijkstra's computation to check if it should expand the pruned set. Roughly speaking, we prove that our algorithm's solution is almost always correct, but its cumulative runtime is not

Figure 11.1: A standard algorithm computing the shortest path from the upper to the lower star will explore many nodes (grey), even nodes in the opposite direction. Our algorithm learns to prune to a subgraph (black) of nodes that have been included in prior shortest paths.

much larger than that of running an optimal algorithm on the maximally-pruned search space in hindsight. Our results hold for worst-case sequences of problem instances, and we do not make any distributional assumptions.

In a bit more detail, let $f : \mathcal{Z} \to Y$ be a function that takes as input a problem instance $z \in \mathcal{Z}$ and returns a solution $y \in Y$. Our algorithm receives a sequence of inputs from $\mathcal{Z}$. Our high-level goal is to correctly compute f on almost every round while minimizing runtime. For example, each $z \in \mathcal{Z}$ might be a set of graph edge weights for some fixed graph $G = (V, E)$ and $f(z)$ might be the shortest s-t path for some vertices s and t. Given a sequence $z_1, \ldots, z_T \in \mathcal{Z}$, a worst-case algorithm would simply compute and return $f(z_i)$ for every instance z_i. However, in many application domains, we have access to other functions mapping $\mathcal{Z}$ to Y, which are faster to compute. These simpler functions are defined by subsets S of a universe $\mathcal{U}$ that represents the entire search space. We call each subset a "pruning" of the search space. For example, in the shortest paths problem, $\mathcal{U}$ equals the set E of edges and a pruning $S \subset E$ is a subset of the edges. The function corresponding to S, which we denote $f_S : \mathcal{Z} \to Y$, also takes as input edge weights z, but returns the shortest path from s to t using only edges from the set S. By definition, the function that is correct on every input is $f = f_{\mathcal{U}}$. We assume that for every z, there is a set $S^*(z) \subseteq \mathcal{U}$ such that $f_S(z) = f(z)$ if and only if $S \supseteq S^*(z)$ – a mild assumption we discuss in more detail later on.

Given a sequence of inputs $z_1, \ldots, z_T$, our algorithm returns the value $f_{S_i}(z_i)$ on round i, where S_i is chosen based on the first i inputs $z_1, \ldots, z_i$. Our goal is two fold: first, we hope to minimize the size of each S_i (and thereby maximally prune the search space), since $|S_i|$ is often monotonically related to the runtime of computing $f_{S_i}(z_i)$. For example, a shortest path computation will typically run faster if we consider only paths that use a small subset of edges. To this end, we prove that if S^* is the smallest set such that $f_{S^*}(z_i) = f(z_i)$ for all i (or equivalently, $S^* = \bigcup_{i=1}^{T} S^*(z_i)$), then

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}|S_i|\right] \leq |S^*| + \frac{|\mathcal{U}| - |S^*|}{\sqrt{T}},$$

where the expectation is over the algorithm's randomness. At the same time, we seek to minimize the the number of mistakes the our algorithm makes (i.e., rounds i where $f(z_i) \neq f_{S_i}(z_i)$). We prove that the expected fraction of rounds i where $f_{S_i}(z_i) \neq f(z_i)$ is $O(|S^*|/\sqrt{T})$. Finally, the expected runtime[1] of the algorithm is the expected time required to compute $f_{S_i}(z_i)$ for $i \in [T]$,

[1]As we will formalize, when determining $S_1, \ldots, S_T$, our algorithm must compute the smallest set S such that

plus $O(|S^*|\sqrt{T})$ expected time to determine the subsets $S_1, \ldots, S_T$.

We instantiate our algorithm and corresponding theorem in three diverse settings—shortest-path routing, linear programming, and string matching—to illustrate the flexibility of our approach. We present experiments on real-world maps and economically-motivated linear programs. In the case of shortest-path routing, our algorithm's performance is illustrated in Figure 11.1. Our algorithm explores up to five times fewer nodes on average than Dijkstra's algorithm, while sacrificing accuracy on only a small number of rounds. In the case of linear programming, when the objective function is perturbed on each round but the constraints remain invariant, we show that it is possible to significantly prune the constraint matrix, allowing our algorithm to make fewer simplex iterations to find solutions that are nearly always optimal.

The results in this section are joint work with Daniel Alabi, Adam Tauman Kalai, Katrina Ligett, Cameron Musco, and Christos Tzamos [4]. Our current results were published in COLT 2019.

11.1 Related work

The results in this chapter advance a recent line of research studying the foundations of algorithm configuration. Many of these works study a distributional setting, as in Part I: there is a distribution over problem instances and the goal is to use a set of samples from this distribution to determine an algorithm from some fixed class with the best expected performance. In our setting, there is no distribution over instances: they may be adversarially selected.

Several papers provide guarantees for online algorithm configuration without distributional assumptions from a theoretical perspective [22, 51, 89]. Before the arrival of any problem instance, the learning algorithm fixes a class of algorithms to learn over. The classes of algorithms that Gupta and Roughgarden [89], Cohen-Addad and Kanade [51], and Balcan et al. [22] study are infinite, defined by real-valued parameters. The goal is to select parameters at each timestep while minimizing regret. These papers provide conditions under which it is possible to design algorithms achieving sublinear regret. These are conditions on the cost functions mapping the real-valued parameters to the algorithm's performance on any input (the "dual functions," as formalized in Chapter 7). In our setting, the choice of a pruning S can be viewed as a parameter, but this parameter is combinatorial, not real-valued, so the prior analyses do not apply.

Several works have studied how to take advantage of structure shared over a sequence of repeated computations for specific applications, including linear programming [30] and matching [63]. As in our work, these algorithms have full access to the problem instances they are attempting to solve. These approaches are quite different (e.g., using machine classifiers) and highly tailored to the application domain, whereas we provide a general algorithmic framework and instantiate it in several different settings.

Since our algorithm receives input instances in an online fashion and makes no distributional assumptions on these instances, our setting is reminiscent of online optimization. However, unlike the typical online setting, we observe each input x_i *before* choosing an output y_i. Thus, if runtime costs were not a concern, we could always return the best output for each input. We seek to trade off correctness for lower runtime costs. In contrast, in online optimization, one must commit to an output y_i before seeing each input x_i, in both the full information and bandit settings [see, e.g., 13, 105]. In such a setting, one cannot hope to return the best y_i for each x_i with

$f_S(z_i) = f(z_i)$ on some of the inputs z_i. In all of the applications we discuss, the total runtime required for these computations is upper bounded by the total time required to compute $f_{S_i}(z_i)$ for $i \in [T]$.

significant probability. Instead, the typical goal is that the performance over all inputs should compete with the performance of the best fixed output in hindsight.

11.2 Model

We start by defining our model of repeated computation. Let $\mathcal{Z}$ be an abstract set of problem instances and let Y be a set of possible solutions. We design an algorithm that operates over T rounds: on round i, it receives an instance $z_i \in \mathcal{Z}$ and returns some element of Y.

Definition 11.2.1 (Repeated algorithm)**.** Over T rounds, a repeated algorithm $\mathscr{A}$ encounters a sequence of inputs $z_1, z_2, \ldots, z_T \in \mathcal{Z}$. On round i, after receiving input z_i, it outputs $\mathcal{A}(z_{1:i}) \in Y$, where $z_{1:i}$ denotes the sequence $z_1, \ldots, z_i$. A repeated algorithm may maintain a state from period to period, and thus $\mathscr{A}(z_{1:i})$ may potentially depend on all of $z_1, ..., z_i$.

We assume each problem instance $z \in \mathcal{Z}$ has a unique correct solution (invoking tie-breaking assumptions as necessary; in Section 11.5, we discuss how to handle problems that admit multiple solutions). We denote the mapping from instances to correct solutions as $f : \mathcal{Z} \to Y$. For example, in the case of shortest paths, we fix a graph G and a pair (s, t) of source and terminal nodes. Each instance $z \in \mathcal{Z}$ represents a weighting of the graph's edges. The set Y consists of all paths from s to t in G. Then $f(z)$ returns the shortest path from s to t in G, given the edge weights z (breaking ties according to some canonical ordering of the elements of Y, as discussed in Section 11.5). To measure correctness, we use a *mistake bound model* [see, e.g., 131].

Definition 11.2.2 (Repeated algorithm mistake bound)**.** The mistake bound of the repeated algorithm $\mathcal{A}$ given inputs $z_1, \ldots, z_T$ is

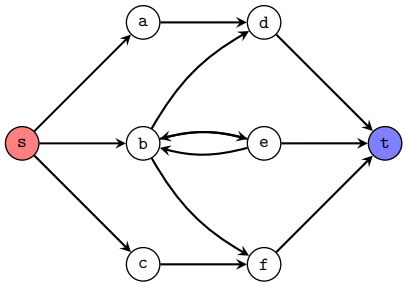
$$M_T(\mathcal{A}, z_{1:T}) = \mathbb{E}\left[\sum_{i=1}^{T} \mathbb{I}_{\{\mathcal{A}(z_{1:i}) \neq f(z_i)\}}\right],$$

where the expectation is over the algorithm's random choices.

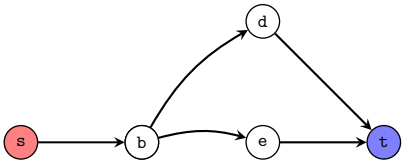
To minimize the number of mistakes, the naïve algorithm would simply compute the function $f(z_i)$ at every round i. However, in our applications, we will have the option of computing other functions mapping the set $\mathcal{Z}$ of inputs to the set Y of outputs that are faster to compute than f. Broadly speaking, these simpler functions are defined by subsets S of a universe $\mathscr{U}$, or "prunings" of $\mathscr{U}$. For example, in the shortest paths problem, given a fixed graph $G = (V, E)$ as well as source and terminal nodes $s, t \in V$, the universe is the set of edges, i.e., $\mathscr{U} = E$. Each input z is a set of edge weights and $f(z)$ computes the shortest s-t path in G under the input weights. The simpler function corresponding to a subset $S \subset E$ of edges also takes as input weights z, but it returns the shortest path from s to t using only edges from the set S (with $f_S(z) = \perp$ if no such path exists). Intuitively, the universe $\mathscr{U}$ contains all the information necessary to compute the correct solution $f(z)$ to any input z, whereas the function corresponding to a subset $S \subset \mathscr{U}$ can only compute a subproblem using information restricted to S.

Let $f_S : \mathcal{Z} \to Y$ denote the function corresponding to the set $S \subseteq \mathscr{U}$. We make two natural assumptions on these functions. First, we assume the function corresponding to the universe $\mathscr{U}$ is always correct. Second, we assume there is a unique smallest set $S^*(z) \subseteq \mathscr{U}$ that any pruning must contain in order to correctly compute $f(z)$. These assumptions are summarized below.

Assumption 11.2.3. *For all $x \in \mathcal{Z}$, $f_{\mathscr{U}}(z) = f(z)$. Also, there exists a unique smallest set $S^*(z) \subseteq \mathscr{U}$ such that $f_{\mathscr{U}}(z) = f_S(z)$ if and only if $S^*(z) \subseteq S$.*



(a) Original unweighted graph G.



(b) Optimal pruning.

Figure 11.2: Repeated shortest paths and optimal pruning of a graph G. If the shortest path was always s-b-e-t or s-b-d-t, it would be unnecessary to search the entire graph for each instance.

Given a sequence of inputs $z_1, \ldots, z_T$, our algorithm returns the value $f_{S_i}(z_i)$ on round i, where the choice of S_i depends on the first i inputs $z_1, \ldots, z_i$. In our applications, it is typically faster to compute f_S over $f_{S'}$ if $|S| < |S'|$. Thus, our goal is to minimize the number of mistakes the algorithm makes while simultaneously minimizing $\mathbb{E}\left[\sum |S_i|\right]$. Though we are agnostic to the specific runtime of computing each function f_{S_i}, minimizing $\mathbb{E}\left[\sum |S_i|\right]$ roughly amounts to minimizing the search space size and our algorithm's runtime in the applications we consider.

We now describe how this model can be instantiated in three classic settings: shortest-path routing, string search, and linear programming.

Shortest-path routing. In the repeated shortest paths problem, we are given a graph $G = (V, E)$ (with static structure) and a fixed pair $s, t \in V$ of source and terminal nodes. In period $i \in [T]$, the algorithm receives a nonnegative weight assignment $z_i : E \to \mathbb{R}_{\geq 0}$. Figure 11.2 illustrates the pruning model applied to the repeated shortest paths problem.

For this problem, the universe is the edge set (i.e., $\mathscr{U} = E$) and S is a subset of edges in the graph. The set $\mathcal{Z}$ consists of all possible weight assignments to edges in the graph G and $Y \subseteq 2^E \cup \{\perp\}$ is the set of all paths in the graph, with $\perp$ indicating that no path exists. The function $f(z)$ returns the shortest s-t path in G given edge weights z. For any $S \subseteq \mathscr{U}$, the function $f_S : \mathcal{Z} \to Y$ computes the shortest s-t path on the subgraph induced by the edges in S (breaking ties by a canonical edge ordering). If S does not include any s-t path, we define $f_S(z) = \perp$. Part 1 of Assumption 11.2.3 holds because $\mathscr{U} = E$, so $f_{\mathscr{U}}$ computes the shortest path on the entire graph. Part 2 of Assumption 11.2.3 also holds: since $f_S : \mathcal{Z} \to Y$ computes the shortest s-t path on the subgraph induced by the edges in S (breaking ties by some canonical edge ordering), we can see that $f_S(z_i) = S^*(z_i)$ if and only if $S^*(z_i) \subseteq S$. To "canonicalize" the algorithm so there is always a unique solution, we assume there is a given ordering on edges and that ties are broken lexicographically according to the path description. This is easily achieved by keeping the heap maintained by Dijkstra's algorithm sorted not only by distances but also lexicographically.

Linear programming. We consider computing $\operatorname{argmax}_{\boldsymbol{y} \in \mathbb{R}^n} \left\{\boldsymbol{z}^T \boldsymbol{y} : A\boldsymbol{y} \leq \boldsymbol{b}\right\}$, where we assume that $(A, \boldsymbol{b}) \in \mathbb{R}^{m \times n} \times \mathbb{R}^m$ is fixed across all times steps but the vector $\boldsymbol{z}_i \in \mathcal{Z} \subseteq \mathbb{R}^n$ defining the objective function $\boldsymbol{z}_i^T \boldsymbol{y}$ may differ for each $i \in [T]$. To instantiate our pruning model, the universe $\mathscr{U} = [m]$ is the set of all constraint indices and each $S \subseteq \mathscr{U}$ indicates a subset of those constraints. The set Y equals $\mathbb{R}^n \cup \{\perp\}$. For simplicity, we assume that the set $\mathcal{Z} \subseteq \mathbb{R}^n$ of objectives contains only directions $\boldsymbol{z}$ such that there is a unique solution $\boldsymbol{y} \in \mathbb{R}^n$ that is the intersection of exactly n

constraints in A. This avoids both dealing with solutions that are the intersection of more than n constraints and directions that are under-determined and have infinitely-many solutions forming a facet. See Section 11.5 for a discussion of this issue in general.

Given $\boldsymbol{z} \in \mathbb{R}^n$, the function f computes the linear program's optimal solution, i.e., $f(\boldsymbol{z}) = \text{argmax}_{\boldsymbol{y} \in \mathbb{R}^n} \left\{ \boldsymbol{z}^T \boldsymbol{y} : A\boldsymbol{y} \leq \boldsymbol{b} \right\}$. For a subset of constraints $S \subseteq \mathscr{U}$, the function f_S computes the optimal solution restricted to those constraints, i.e., $f_S(\boldsymbol{z}) = \text{argmax}_{\boldsymbol{y} \in \mathbb{R}^n} \{ \boldsymbol{z}^T \boldsymbol{y} : A_S \boldsymbol{y} \leq \boldsymbol{b}_S \}$, where $A_S \in \mathbb{R}^{|S| \times n}$ is the submatrix of A consisting of the rows indexed by elements of S and $\boldsymbol{b}_S \in \mathbb{R}^{|S|}$ is the vector $\boldsymbol{b}$ with indices restricted to elements of S. We further write $f_S(\boldsymbol{z}) = \perp$ if there is no unique solution to the linear program (which may happen for small sets S even if the whole LP does have a unique solution). Part 1 of Assumption 11.2.3 holds because $A_{\mathscr{U}} = A$ and $\boldsymbol{b}_{\mathscr{U}} = \boldsymbol{b}$, so it is indeed the case that $f_{\mathscr{U}} = f$. To see why part 2 of Assumption 11.2.3 also holds, suppose that $f_S(\boldsymbol{z}) = f(\boldsymbol{z})$. If $f(\boldsymbol{z}) \neq \perp$, the vector $f_S(\boldsymbol{z})$ must be the intersection of exactly n constraints in A_S, which by definition are indexed by elements of S. This means that $S^*(\boldsymbol{z}) \subseteq S$.

String search. In string search, the goal is to find the location of a short pattern in a long string. At timestep i, the algorithm receives a long string q_i of some fixed length n and a pattern p_i of some fixed length $m \leq n$. We denote the long string as $q_i = \left(q_i^{(1)}, \ldots, q_i^{(n)}\right)$ and the pattern as $p_i = \left(p_i^{(1)}, \ldots, p_i^{(m)}\right)$. The goal is to find an index $j \in [n - m + 1]$ such that $p_i = \left(q_i^{(j)}, q_i^{(j+1)}, \ldots, q_i^{(j+m-1)}\right)$. The function f returns the smallest such index j, or $\perp$ if there is no match. In this setting, the set $\mathscr{Z}$ of inputs consists of all string pairs of length n and m (e.g., $\{A, T, G, C\}^{n \times m}$ for DNA sequences) and the set $Y = [n - m + 1]$ is the set of all possible match indices. The universe $\mathscr{U} = [n - m + 1]$ also consists of all possible match indices. For any $S \subseteq \mathscr{U}$, the function $f_S(q_i, p_i)$ returns the smallest index $j \in S$ such that $p_i = \left(q_i^{(j)}, q_i^{(j+1)}, \ldots, q_i^{(j+m-1)}\right)$, which we denote j_i^*. It returns $\perp$ if there is no match. We can see that part 1 of Assumption 11.2.3 holds: $f_{\mathscr{U}}(q, p) = f(q, p)$ for all $(q, p) \in \mathscr{Z}$, since $f_{\mathscr{U}}$ checks every index in $[n - m + 1]$ for a match. Moreover, part 2 of Assumption 11.2.3 holds because $f_{\mathscr{U}}(z_i) = f_S(z_i)$ if and only if $S^*(z_i) = \{j_i^*\} \subseteq S$.

11.3 The algorithm

We now present an algorithm (Algorithm 6), denoted $\mathcal{A}^*$, that encounters a sequence of inputs $z_1, \ldots, z_T$ one-by-one. At timestep i, it computes the value $f_{S_i}(z_i)$, where the choice of $S_i \subseteq \mathscr{U}$ depends on the first i inputs $z_1, \ldots, z_i$. We prove that, in expectation, the number of mistakes it makes (i.e., rounds where $f_{S_i}(z_i) \neq f(z_i)$) is small, as is $\sum_{i=1}^{T} |S_i|$.

Our algorithm keeps track of a pruning of $\mathscr{U}$, which we call $\bar{S}_i$ at timestep i. In the first round, the pruned set is empty ($\bar{S}_1 = \emptyset$). On round i, with some probability p_i, the algorithm computes the function $f_{\mathscr{U}}(z_i)$ and then computes $S^*(z_i)$, the unique smallest set that any pruning must contain in order to correctly compute $f_{\mathscr{U}}(z_i)$. (As we discuss in Section 11.3.1, in all of the applications we consider, computing $S^*(z_i)$ amounts to evaluating $f_{\mathscr{U}}(z_i)$.) The algorithm unions $S^*(z_i)$ with $\bar{S}_i$ to create the set $\bar{S}_{i+1}$. Otherwise, with probability $1 - p_i$, it outputs $f_{\bar{S}_i}(z_i)$, and does not update the set $\bar{S}_i$ (i.e., $\bar{S}_{i+1} = \bar{S}_i$). It repeats in this fashion for all T rounds.

In the remainder of this section, we use the notation S^* to denote the smallest set such that $f_{S^*}(z_i) = f(z_i)$ for all $i \in [T]$. We now provide a mistake bound for Algorithm 6.

Algorithm 6 Our repeated algorithm $\mathcal{A}^*$

$\bar{S}_1 \leftarrow \emptyset$
for $i \in \{1, \ldots, T\}$ **do**
    Receive input $z_i \in \mathcal{Z}$.
    With probability p_i, output $f_{\mathscr{U}}(z_i)$. Compute $S^*(z_i)$ and set $\bar{S}_{i+1} \leftarrow \bar{S}_i \cup S^*(z_i)$.
    Otherwise (with probability $1 - p_i$), output $f_{\bar{S}_i}(z_i)$ and set $\bar{S}_{i+1} \leftarrow \bar{S}_i$.

Theorem 11.3.1. *For any $p \in (0, 1]$ such that $p_i \geq p$ for all $i \in [T]$ and any inputs $z_1, \ldots, z_T$, Algorithm 6 has a mistake bound of*

$$M_T(\mathcal{A}^*, z_{1:T}) \leq \frac{|S^*|(1-p)(1-(1-p)^T)}{p} \leq \frac{|S^*|}{p}.$$

Corollary 11.3.2. *Algorithm 6 with $p_i = \frac{1}{\sqrt{i}}$ has a mistake bound of $M_T(\mathcal{A}^*, z_{1:T}) \leq |S^*|\sqrt{T}$.*

In the following theorem, we prove that the mistake bound in Theorem 11.3.1 is nearly tight. In particular, we show that for any $k \in \{1, \ldots, T\}$ there exists a random sequence of inputs $z_1, \ldots, z_T$ such that $\mathbb{E}[|S^*|] \approx k$ and $M_T(\mathcal{A}^*, z_{1:T}) = \frac{k(1-p)(1-(1-p/k)^T)}{p}$. This nearly matches the upper bound from Theorem 11.3.1 of

$$\frac{|S^*|(1-p)(1-(1-p)^T)}{p}.$$

Theorem 11.3.3. *For any $p \in (0, 1]$, any time horizon T, and any $k \in \{1, \ldots, T\}$, there is a random sequence of inputs to Algorithm 6 such that*

$$\mathbb{E}[|S^*|] = k\left(1 - \left(1 - \frac{1}{k}\right)^T\right)$$

and its expected mistake bound with $p_i = p$ for all $i \in [T]$ is

$$\mathbb{E}[M_T(\mathcal{A}^*, \boldsymbol{z}_{1:T})] = \frac{k(1-p)(1-(1-p/k)^T)}{p}.$$

The expectation is over the sequence of inputs.

In Theorem 11.3.1, we bounded the expected number of mistakes Algorithm 6 makes. Next, we bound $\mathbb{E}\left[\frac{1}{T}\sum|S_i|\right]$, where S_i is the set such that Algorithm 6 outputs $f_{S_i}(z_i)$ in round i (so either $S_i = \bar{S}_i$ or $S_i = \mathscr{U}$, depending on the algorithm's random choice). In our applications, minimizing $\mathbb{E}\left[\frac{1}{T}\sum|S_i|\right]$ means minimizing the search space size, which roughly amounts to minimizing the average expected runtime of Algorithm 6.

Theorem 11.3.4. *For any inputs $z_1, \ldots, z_T$, let $S_1, \ldots, S_T$ be the sets such that on round i, Algorithm 6 computes the function f_{S_i}. Then*

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}|S_i|\right] \leq |S^*| + \frac{1}{T}\sum_{i=1}^{T} p_i(|\mathscr{U}| - |S^*|),$$

where the randomness is over the coin tosses of Algorithm 6.

If we set $p_i = 1/\sqrt{i}$ for all i, we have the following corollary, since $\sum_{i=1}^{T} p_i \leq 2\sqrt{T}$.

Corollary 11.3.5. *Given a set of inputs $z_1, \ldots, z_T$, let $S_1, ..., S_T$ be the sets such that on round i, Algorithm 6 computes the function f_{S_i}. If $p_i = \frac{1}{\sqrt{i}}$ for all $i \in [T]$, then*

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}|S_i|\right] \leq |S^*| + \frac{2(|\mathscr{U}| - |S^*|)}{\sqrt{T}},$$

where the expectation is over the random choices of Algorithm 6.

11.3.1 Instantiations of Algorithm 6

We now revisit and discuss instantiations of Algorithm 6 for the three applications outlined in Section 11.2: shortest-path routing, linear programming, and string search. For each problem, we describe how one might compute the sets $S^*(z_i)$ for all $i \in [T]$.

Shortest-path routing. In this setting, the algorithm computes the true shortest path $f(z)$ using, say, Dijkstra's shortest-path algorithm, and the set $S^*(z)$ is simply the union of edges in that path. Since $S^* = \cup_{i=1}^{T} S^*(z_i)$, the mistake bound of $|S^*|\sqrt{T}$ given by Corollary 11.3.2 is particularly strong when the shortest path does not vary much from day to day. Corollary 11.3.5 guarantees that the average edge set size run through Dijkstra's algorithm is at most $|S^*| + \frac{2(|E|-|S^*|)}{\sqrt{T}}$. Since the worst-case running time of Dijkstra's algorithm on a graph $G' = (V', E')$ is $\tilde{O}(|V'| + |E'|)$, minimizing the average edge set size is a good proxy for minimizing runtime.

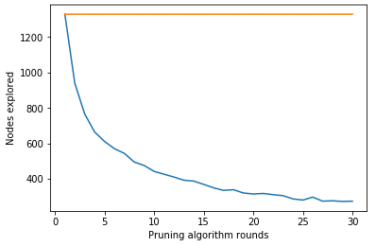
Linear programming. In the context of linear programming, computing the set $S^*(\boldsymbol{z}_i)$ is equivalent to computing $f(\boldsymbol{z})$ and returning the set of tight constraints. Since $S^* = \cup_{i=1}^{T} S^*(\boldsymbol{z}_i)$, the mistake bound of $|S^*|\sqrt{T}$ given by Corollary 11.3.2 is strongest when the same constraints are tight across most timesteps. Corollary 11.3.5 guarantees that the average constraint set size considered in each round is at most $|S^*| + \frac{2(m-|S^*|)}{\sqrt{T}}$, where m is the total number of constraints. Since many well-known solvers take time polynomial in $|S_i|$ to compute f_{S_i}, minimizing $\mathbb{E}\left[\sum |S_i|\right]$ is a close proxy for minimizing runtime.

String search. In this setting, the set $S^*(q_i, p_i)$ consists of the smallest index j such that $p_i = \left(q_i^{(j)}, q_i^{(j+1)}, \ldots, q_i^{(j+m-1)}\right)$, which we denote j_i^*. This means that computing $S^*(q_i, p_i)$ is equivalent to computing $f(q_i, p_i)$. The mistake bound of $|S^*|\sqrt{T} = \left|\cup_{i=1}^{T} \{j_i^*\}\right| \sqrt{T}$ given by Corollary 11.3.2 is particularly strong when the matching indices are similar across string pairs. Corollary 11.3.5 guarantees that the average size of the searched index set in each round is at most $|S^*| + \frac{2(n-|S^*|)}{\sqrt{T}}$. Since the expected average running time of our algorithm using the naïve string-matching algorithm to compute f_{S_i} is $\mathbb{E}\left[\frac{m}{T}\sum_{i=1}^{T}|S_i|\right]$, minimizing $\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}|S_i|\right]$ amounts to minimizing runtime.

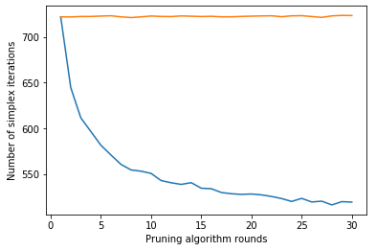
11.4 Experiments

In this section, we present experimental results for shortest-path routing and linear programming.

(a) Grey nodes: the nodes visited by Dijkstra's algorithm. Black nodes: the nodes in our algorithm's pruned subgraph.



(b) Top line: average number of nodes Dijkstra's algorithm explores. Bottom line: average number of nodes Algorithm 6 explores.



(c) Top line: average number of simplex iterations the simplex algorithm makes. Bottom line: average number of simplex iterations Algorithm 6 makes.

Figure 11.3: Empirical evaluation of Algorithm 6 applied to shortest-path routing in Pittsburgh (Figures 11.3a and 11.3b) and linear programming (Figure 11.3c).

Shortest-path routing. We test Algorithm 6's performance on real-world street maps, which we access via Python's OSMnx package [35]. Each street is an edge in the graph and each intersection is a node. The edge's weight is the street's distance. We run our algorithm for 30 rounds (i.e., $T = 30$) with $p_i = 1/\sqrt{i}$ for all $i \in [T]$. On each round, we randomly perturb each edge's weight via the following procedure. Let $G = (V, E)$ be the original graph we access via Python's OSMnx package. Let $\boldsymbol{z} \in \mathbb{R}^{|E|}$ be a vector representing all edges' weights. On the i^{th} round, we select a vector $\boldsymbol{r}_i \in \mathbb{R}^{|E|}$ such that each component is drawn i.i.d. from the normal distribution with a mean of 0 and a standard deviation of 1. We then define a new edge-weight vector $\boldsymbol{z}_i$ such that $z_i[j] = \mathbb{1}_{\{z[j]+r_i[j]>0\}} \left(z[j] + r_i[j]\right)$.

In Figures 11.3a and 11.3b, we illustrate our algorithm's performance in Pittsburgh. Figure 11.3a illustrates the nodes explored by our algorithm over $T = 30$ rounds. The goal is to get from the upper to the lower star. The nodes colored grey are the nodes Dijkstra's algorithm would have visited if we had run Dijkstra's algorithm on all T rounds. The nodes colored black are the nodes in the pruned subgraph after the T rounds. Figure 11.3b illustrates the results of running our algorithm a total of 5000 times ($T = 30$ rounds each run). The top (orange) line shows the number of nodes Dijkstra's algorithm explored averaged over all 5000 runs. The bottom (blue) line shows the average number of nodes our algorithm explored. Our algorithm returned the incorrect path on a 0.068 fraction of the $5000 \cdot T = 150,000$ rounds.

Linear programming. We generate linear programming instances representing the linear relaxation of the combinatorial auction winner determination problem. Auction Test Suite (CATS) [124] to generate these instances. This test suite is meant to generate instances that are realistic and economically well-motivated. We use the CATS generator to create an initial instance with an objective function defined by a vector $\boldsymbol{z}$ and constraints defined by a matrix A and a vector $\boldsymbol{b}$. On each round, we perturb the objective vector.

From the CATS "Arbitrary" generator, we create an instance with 204 bids and 538 goods which has 204 variables and 946 constraints. We run Algorithm 6 for 30 rounds ($T = 30$) with $p_i = 1/\sqrt{i}$ for all $i \in [T]$, and we repeat this 5000 times. In Figure 11.3c, the top (orange) line shows the number of simplex iterations the full simplex algorithm makes averaged over all 5000 runs. The bottom (blue) line shows the number of simplex iterations our algorithm makes averaged over all 5000 runs. We solve the linear program on each round using the SciPy default

linear programming solver [103], which implements the simplex algorithm [61]. Our algorithm returned the incorrect solution on a 0.018 fraction of the $5000 \cdot T = 150,000$ rounds.

11.5 Multiple solutions and approximations

In this work, we have assumed that each problem has a unique solution, which we can enforce by defining a canonical ordering on solutions. For string matching, this could be the first match in a string as opposed to any match. For shortest-path routing, it is not difficult to modify shortest-path algorithms to find, among the shortest paths, the one with lexicographically "smallest" description given some ordering of edges. Alternatively, one might simply assume that there is exactly one solution, e.g., no ties in a shortest-path problem with real-valued edge weights. This latter solution is what we have chosen for the linear programming model, for simplicity.

It would be natural to try to extend our work to problems that have multiple solutions, or even to approximate solutions. However, addressing multiple solutions in repeated computation rapidly raises NP-hard challenges. To see this, consider a graph with two nodes, s and t, connected by m parallel edges. Suppose the goal is to find any shortest path and suppose that in each period, the edge weights are all 0 or 1, with at least one edge having weight 0. If Z_i is the set of edges with 0 weight on period i, finding the smallest pruning which includes a shortest path on each period is trivially equivalent to set cover on the sets Z_i. Hence, any repeated algorithm handling problems with multiple solutions must address this computational hardness.

Chapter 12

Proposed research and future directions

Our results in this proposal suggest many exciting directions for future research in both algorithm configuration and mechanism design. In this chapter, I describe a few of these open questions, and conclude with a proposed schedule for the remainder of my time as a PhD student.

12.1 Algorithm configuration

12.1.1 Application-independent directions

Mapping problem instances to high-performing configurations. Throughout this proposal, we adopted the classic batch learning model for algorithm configuration: the configuration procedure uses a training set of problem instances to learn a *single* parameter setting that will have high performance on instances outside of the training set. A more flexible approach could learn a mapping from problem instances to configurations. This would entail using features of a problem instance to predict a configuration that is well-suited for that instance. This is the idea behind portfolio-based algorithm selectors such as SATzilla [191], but we are not aware of any papers that provide a theoretical analysis of this problem. This approach was also used to help field large-scale combinatorial auctions [171]. This direction has the potential for both algorithmic and statistical innovations.

Faster configuration algorithms for infinite parameter spaces. One of the major challenges in implementing the configuration algorithm from Chapter 10 (Algorithm 5) is its sample complexity. There are a few ways we could address this issue. For example, it may be beneficial to use a multiplicative uniform convergence bound in terms of Rademacher complexity, since our definition of approximate optimality (Definition 10.1.2) is a multiplicative notion. In contrast, we currently rely on an additive error bound (Theorem 2.2.2). Data-dependent multiplicative uniform convergence bounds may be of independent interest beyond the context of algorithm configuration.

Learning within an instance. As an algorithm solves a problem instance, it may be useful for the algorithm to adapt. For example, it may be advantageous to adjust the variable selection policy branch-and-bound uses while building the search tree; a policy used at the root of the tree may no longer be optimal near the leaves. This type of learning within an instance has been explored with the aid of reinforcement learning [60], but to our knowledge, we do not yet have a theoretical understanding of this problem.

	test set	dfs	bfs	bfs/plunge	estimate	estim/plunge	hybrid
time	MIPLIB	+22	+28	+3	+14	−3	+8
	CORAL	+49	+38	+2	+19	−3	−5
	MILP	+12	+20	+1	+8	+5	+3
	ENLIGHT	+53	+74	+53	0	−11	+25
	ALU	−40	+101	+25	+113	+25	+3
	FCTP	+59	+25	+9	+22	−2	+7
	ACC	+102	−7	−7	−7	+8	+7
	FC	−4	+14	+2	+6	+1	+5
	ARCSET	+152	+53	+11	+42	+5	+13
	MIK	−7	+44	+3	+39	−2	+6
	total	**+28**	**+30**	**+4**	**+16**	**0**	**+3**
nodes	MIPLIB	+64	−18	+3	−15	−3	+7
	CORAL	+143	−28	0	−9	+3	−9
	MILP	+41	−12	+2	−9	+9	+8
	ENLIGHT	+109	+16	+36	+2	+7	+13
	ALU	−43	+50	+29	+87	+17	+6
	FCTP	+74	−6	+2	−5	−3	+2
	ACC	+425	−30	−7	−29	+25	+27
	FC	−9	−31	+9	−32	+9	+12
	ARCSET	+249	+6	0	+19	+7	+3
	MIK	+19	+3	−4	+6	+2	0
	total	**+78**	**−17**	**+3**	**−9**	**+4**	**+2**

Table 6.1. Performance effect of different node selection strategies for solving MIP instances. The values denote the percental changes in the shifted geometric mean of the runtime (top) and number of branching nodes (bottom) compared to the default *interleaved best estimate/best first search* strategy. Positive values represent a deterioration, negative values an improvement.

Figure 12.1: Experiments by Achterberg [1]. These experiments are run on SCIP, the leading open-source MIP solver.

12.1.2 Integer programming directions

Integer programming is an especially rich, high-impact area for future research, given the widespread use of the highly-customizable branch-and-bound algorithm.

Learning node selection policies. In Chapter 3, we provided guarantees for learning high-performing variable selection policies (VSPs). Node selection policies (NSPs) are another important facet of branch-and-bound. An NSP determines, at each round of the algorithm, which leaf of the search tree should be branched on next. A common choice is best-first search, but Achterberg [1] proposes seven different NSPs and suggests that one could combine multiple strategies by, for example, "applying the selection rules in an interleaving fashion or by using weighted combinations of the individual node selection score values." Achterberg [1] compares several NSPs in Figure 12.1, and we can see that no one strategy is universally optimal. For example, depth-first search (DFS) generally performs poorly, except on the ALU dataset, which consists of infeasible MIPs, where DFS outperforms all other strategies. In future research, we could learn high-performing NSPs, and combine VSP learning with NSP learning. This would involve algorithmic innovations, since our learning algorithm in Chapter 3 only tunes one VSP parameter.

Non-linear combinations of scoring rules. For the journal version of the paper described in Chapter 3 [21], we plan to learn non-linear combinations of scoring rules, such as $\text{score}_1^{\rho}\text{score}_2^{1-\rho}$, with $\rho \in [0,1]$. When $\text{score}_1(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^-}, 10^{-6}\right\}$, $\text{score}_2(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, 10^{-6}\right\}$, and $\rho = \frac{1}{2}$, this scoring rule recovers the product scoring rule from Section 3.2, which is known to perform well in practice [1]. We plan to prove sample complexity bounds for these non-linear combinations of scoring rules, and find examples where learning the right choice of parameters leads to strong improvements over a default parameter setting, such as the product scoring rule.

Learning cutting planes. Cutting planes are a crucial aspect of branch-and-bound's ability to quickly solve integer programs. A cutting plane is a linear inequality that is used to refine the integer program's feasible region. There are a variety of different cutting planes that a branch-and-bound algorithm might employ, and some have their own tunable parameters. How should we choose which cuts to employ during branch-and-bound, and when should we employ them? Can we use a machine learning approach to decide?

12.2 Mechanism design

New advances for tightening pseudo-dimension bounds, or proving lower bounds. In Section 6.1, we provide a few pseudo-dimension lower bounds (Theorem 6.1.22), but intuition suggests that it might be possible to tighten some of the bounds. For example, the best-known pseudo-dimension bound for item-pricing mechanisms with a single combinatorial buyer is $O(m^2)$ [24, 146], where m is the number of items for sale. In essence, the proof of this pseudo-dimension bound identifies, for any fixed set of buyers' values, 2^m hyperplanes splitting the set of all m-dimensional price vectors into 2^{m^2} regions where the buyer's most-preferred bundle is fixed. However, there are only 2^m different bundles, and intuition suggests that the region of prices where the buyer prefers a particular bundle is a convex region. This suggests that our pseudo-dimension proof may suffer from overcounting.

Richness of neutral affine maximizers. In Section 6.2, we studied neutral affine maximizers (NAMs). Let n be the number of agents. There is a very simple subset of NAMs of size n defined by parameter vectors $\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_n$ where each vector $\boldsymbol{\rho}_i$ has a 0 in the i^{th} component and a 1 in all other components. In other words, agent i is the sink agent and all other agents have equal say in the NAM's outcome. This class has a pseudo-dimension of $O(\log n)$, whereas the class of all NAMs has a pseudo-dimension of $\Omega(n)$ (Theorem 6.2.3), so learning over the class of all NAMs requires exponentially more samples. In this research direction, we aim to better understand how much more social welfare we obtain by optimizing over the class of all NAMs. In other words, what is the difference in the social welfare of the optimal NAM in this simple subset versus the optimal NAM defined by any parameter vector $\boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}$? Let $u_{\boldsymbol{\mu}}(\boldsymbol{v}) \in [0, H]$ be the social welfare of the NAM parameterized by $\boldsymbol{\mu}$ on the set of bids $\boldsymbol{v}$. One hypothesis is that for any distribution $\mathcal{D}$,

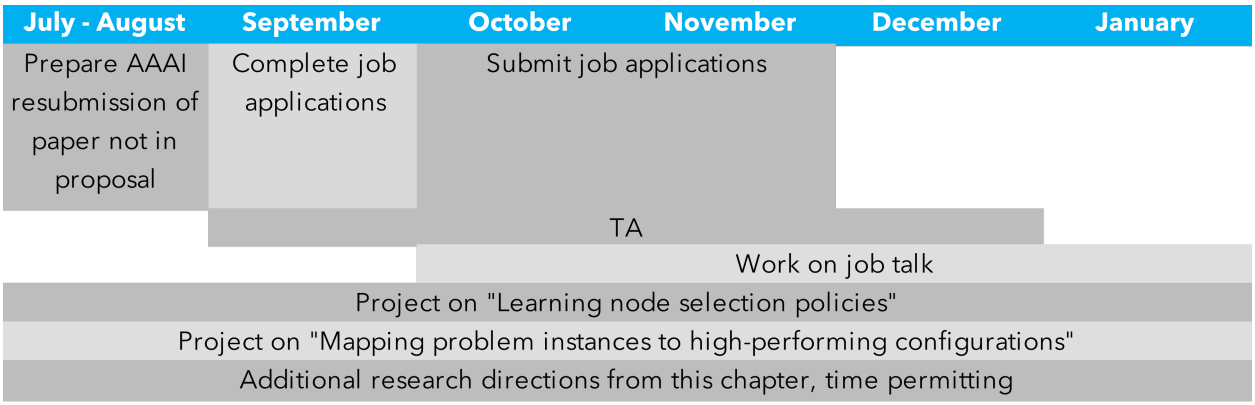
$$\max_{i \in [n]} \mathop{\mathbb{E}}_{\boldsymbol{v} \sim \mathcal{D}} \left[u_{\boldsymbol{\mu}_i}(\boldsymbol{v})\right] \geq \max_{\boldsymbol{\mu} \in \mathbb{R}^n} \mathop{\mathbb{E}}_{\boldsymbol{v} \sim \mathcal{D}} \left[u_{\boldsymbol{\mu}}(\boldsymbol{v})\right] - O\left(\frac{H}{n}\right).$$

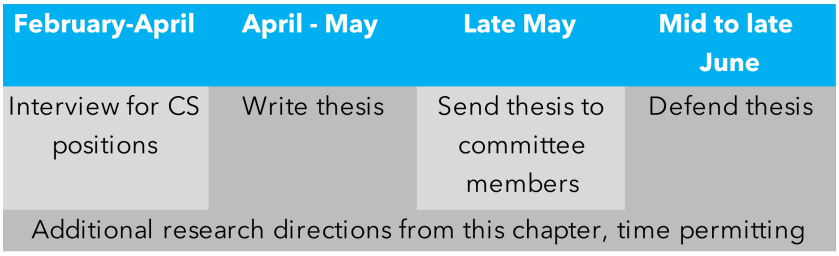
Intuitively, it seems like there is always some agent whose value can be ignored without decreasing the optimal social welfare by too much, where by optimal social welfare, we mean the highest social welfare of any alternative. Roughly speaking, this is because either all agents' values are equally consequential, in which case ignoring one agent's value will only decrease the optimal

social welfare by $O\left(\frac{H}{n}\right)$, or some agents' values are more consequential than others, in which case ignoring an inconsequential agent will not change the optimal social welfare by too much.

12.3 Proposed schedule

The figure below outlines how I plan to spend my time for the remainder of my PhD. I aim to defend by the end of my sixth year, June 2021. If time permits, I hope to look into other research directions included in this chapter but not in the schedule.





Bibliography

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

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

[3] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, January 2005.

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

[5] Ekaterina Alekseeva, Yury Kochetov, and Alexandr Plyasunov. An exact method for the discrete $(r|p)$-centroid problem. *Journal of Global Optimization*, 63(3):445–460, 2015.

[6] Noga Alon, Moshe Babaioff, Yannai A Gonczarowski, Yishay Mansour, Shay Moran, and Amir Yehudayoff. Submultiplicative Glivenko-Cantelli and uniform convergence of revenues. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[7] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

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

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

[10] Aaron Archer, Christos Papadimitriou, Kunal Talwar, and Éva Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *Internet Mathematics*, 1(2):129–150, 2004.

[11] Patrick Assouad. Densité et dimension. *Annales de l'Institut Fourier*, 33(3):233–282, 1983.

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

[13] Baruch Awerbuch and Robert Kleinberg. Online linear optimization and adaptive routing. *J. Comput. Syst. Sci.*, 74(1):97–114, 2008.

[14] Eduardo M Azevedo and Eric Budish. Strategy-proofness in the large. *The Review of Economic Studies*, 86(1):81–116, 2018.

[15] Moshe Babaioff, Nicole Immorlica, Brendan Lucier, and S. Matthew Weinberg. A simple and approximately optimal mechanism for an additive buyer. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2014.

[16] Moshe Babaioff, Yannai A Gonczarowski, and Noam Nisan. The menu-size complexity of revenue approximation. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2017.

[17] 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.

[18] Maria-Florina Balcan, Avrim Blum, Jason Hartline, and Yishay Mansour. Reducing mechanism design to algorithm design via machine learning. *Journal of Computer and System Sciences*, 74:78–89, December 2008.

[19] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of automated mechanism design. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

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

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

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

[23] 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*, 2018.

[24] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. A general theory of sample complexity for multi-item profit maximization. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2018.

[25] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? *arXiv preprint arXiv:1908.02894*, 2019.

[26] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Estimating approximate incentive compatibility. *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2019.

[27] Maria-Florina Balcan, Travis Dick, and Manuel Lang. Learning to link. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

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

[29] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Refined bounds for algorithm configuration: The knife-edge of dual class approximability. 2020.

[30] Ashis Gopal Banerjee and Nicholas Roy. Efficiently solving repeated integer linear programming problems by learning solutions of similar linear programming problems using boosting trees. Technical Report MIT-CSAIL-TR-2015-00, MIT Computer Science and Artificial Intelligence Laboratory, 2015.

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

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

[33] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, 1971.

[34] Christian Bessiere and Jean-Charles Régin. Mac and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 61–75. Springer, 1996.

[35] Geoff Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

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

[37] 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.

[38] Felix Brandt, Tuomas Sandholm, and Yoav Shoham. Spiteful bidding in sealed-bid auctions. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India, 2007. Early version in GTDT-05.

[39] Patrick Briest, Shuchi Chawla, Robert Kleinberg, and S Matthew Weinberg. Pricing randomized allocations. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

[40] Sébastien Bubeck, Nikhil R Devanur, Zhiyi Huang, and Rad Niazadeh. Online auctions and multi-scale online learning. *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2017.

[41] Yang Cai and Constantinos Daskalakis. Learning multi-item auctions with (or without) samples. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2017.

[42] Yang Cai, Nikhil R. Devanur, and S. Matthew Weinberg. A duality based unified approach to Bayesian mechanism design. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2016.

[43] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

[44] Nicolo Cesa-Bianchi, Claudio Gentile, and Yishay Mansour. Regret minimization for reserve prices in second-price auctions. *IEEE Transactions on Information Theory*, 61(1):549–564, 2015.

[45] Moses Charikar and Anthony Wirth. Maximizing quadratic programs: extending Grothendieck's inequality. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2004.

[46] Shuchi Chawla, Jason Hartline, and Robert Kleinberg. Algorithmic pricing via virtual valuations. In *Proceedings of the ACM Conference on Electronic Commerce (EC)*, 2007.

[47] Shuchi Chawla, David L Malec, and Balasubramanian Sivan. The power of randomness in bayesian optimal mechanism design. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, pages 149–158, 2010.

[48] Shuchi Chawla, Jason Hartline, and Denis Nekipelov. Mechanism design for data science. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2014.

[49] Shuchi Chawla, Jason Hartline, and Denis Nekipelov. A/B testing of auctions. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2016.

[50] Shuchi Chawla, Jason D. Hartline, and Denis Nekipelov. Mechanism redesign. *arXiv preprint arXiv:1708.04699*, 2017.

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

[52] Richard Cole and Tim Roughgarden. The sample complexity of revenue maximization. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2014.

[53] Wolfram Conen and Tuomas Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 256–259, Tampa, FL, 2001. More detailed description of algorithmic aspects in IJCAI-01 Workshop on Economic Agents, Models, and Mechanisms, pp. 71–80.

[54] Vincent Conitzer and Tuomas Sandholm. Complexity of mechanism design. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 103–110, Edmonton, Canada, 2002.

[55] Vincent Conitzer and Tuomas Sandholm. Applications of automated mechanism design. In *UAI-03 workshop on Bayesian Modeling Applications*, Acapulco, Mexico, 2003.

[56] Vincent Conitzer and Tuomas Sandholm. Computational criticisms of the revelation principle. In *The Conference on Logic and the Foundations of Game and Decision Theory (LOFT)*, Leipzig, Germany, 2004. Earlier versions: AMEC-03, EC-04.

[57] Vincent Conitzer and Tuomas Sandholm. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 132–141, New York, NY, 2004.

[58] Vincent Conitzer and Tuomas Sandholm. Incremental mechanism design. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)*, Hyderabad, India, 2007.

[59] Peter Cramton, Robert Gibbons, and Paul Klemperer. Dissolving a partnership efficiently. *Econometrica*, pages 615–632, 1987.

[60] Hanjun Dai, Elias Boutros Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[61] George Dantzig. *Linear programming and extensions*. Princeton University Press, 2016.

[62] Constantinos Daskalakis and Vasilis Syrgkanis. Learning in auctions: Regret is hard, envy is easy. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2016.

[63] Supratim Deb, Devavrat Shah, and et al. Fast matching algorithms for repetitive optimization: An application to switch scheduling. In *Proceedings of the Conference on Information Sciences and Systems (CISS)*, 2006.

[64] Ofer Dekel, Felix Fischer, and Ariel D Procaccia. Incentive compatible regression learning. *Journal of Computer and System Sciences*, 76(8):759–777, 2010.

[65] Nikhil R Devanur, Zhiyi Huang, and Christos-Alexandros Psomas. The sample complexity of auctions with side information. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2016.

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

[67] 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.

[68] Shahar Dobzinski and Shaddin Dughmi. On the power of randomization in algorithmic mechanism design. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2009.

[69] Shahar Dobzinski and Mukund Sundararajan. On characterizations of truthful mechanisms for combinatorial auctions and scheduling. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2008.

[70] Paul Dütting, Felix Fischer, Pichayut Jirapinyo, John K Lai, Benjamin Lubin, and David C Parkes. Payment rules through discriminant-based classifiers. *ACM Transactions on Economics and Computation (TEAC)*, 3(1):5, 2015.

[71] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David C Parkes, and Sai Srivatsa Ravindranath. Optimal auctions through deep learning. *International Conference on Machine Learning (ICML)*, 2019.

[72] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *The American Economic Review*, 97(1):242–259, March 2007. ISSN 0002-8282.

[73] Edith Elkind. Designing and learning optimal finite support auctions. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007.

[74] Uriel Feige and Michael Langberg. The RPR^2 rounding technique for semidefinite programs. *Journal of Algorithms*, 60(1):1–23, 2006.

[75] Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial auctions via posted prices. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.

[76] Martin S Feldstein. Equity and efficiency in public sector pricing: the optimal two-part tariff. *The Quarterly Journal of Economics*, pages 176–187, 1972.

[77] Zhe Feng, Harikrishna Narasimhan, and David C Parkes. Deep learning for revenue-optimal auctions with budgets. In *Autonomous Agents and Multi-Agent Systems*, 2018.

[78] David Fernández-Baca, Timo Seppäläinen, and Giora Slutzki. Parametric multiple sequence alignment and phylogeny construction. *Journal of Discrete Algorithms*, 2(2):271–287, 2004.

[79] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017.

[80] Matteo Fischetti and Michele Monaci. Branching on nonchimerical fractionalities. *Operations Research Letters*, 40(3):159–164, 2012.

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

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

[83] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[84] Boris Goldengorin, John Keane, Viktor Kuzmenko, and Michael Tso. Optimal supplier choice with discounting. *Journal of the Operational Research Society*, 62(4):690–699, 2011.

[85] Kira Goldner and Anna R Karlin. A prior-independent revenue-maximizing auction for multiple additive bidders. In *International Workshop On Internet And Network Economics (WINE)*, 2016.

[86] Noah Golowich, Harikrishna Narasimhan, and David C Parkes. Deep learning for multi-facility location mechanism design. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[87] Yannai A Gonczarowski and Noam Nisan. Efficient empirical revenue maximization in single-parameter auction environments. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 856–868, 2017.

[88] Yannai A Gonczarowski and S Matthew Weinberg. The sample complexity of up-to-ε multi-dimensional revenue maximization. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

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

[90] Dan Gusfield, Krishnan Balasubramanian, and Dalit Naor. Parametric optimization of sequence alignment. *Algorithmica*, 12(4-5):312–326, 1994.

[91] Matt Harada. The ad exchange's place in a first-price world. *Marketing Technology Insights*, 2018. URL https://martechseries.com/mts-insights/guest-authors/ad-exchanges-place-first-price-world/.

[92] Robert Haralick and Gordon Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

[93] Sergiu Hart and Noam Nisan. Approximate revenue maximization with multiple items. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2012.

[94] Jason Hartline and Samuel Taggart. Non-revelation mechanism design. *arXiv preprint arXiv:1608.01875*, 2016.

[95] 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.

[96] Robert W. Holley, Jean Apgar, George A. Everett, James T. Madison, Mark Marquisee, Susan H. Merrill, John Robert Penswick, and Ada Zamir. Structure of a ribonucleic acid. *Science*, 147(3664):1462–1465, 1965.

[97] Jorg Homberger and Hermann Gehring. A two-level parallel genetic algorithm for the uncapacitated warehouse location problem. In *Hawaii International Conference on System Sciences*, pages 67–67. IEEE, 2008.

[98] Eric Horvitz, Yongshao Ruan, Carla Gomez, Henry Kautz, Bart Selman, and Max Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2001.

[99] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[100] Zhiyi Huang, Yishay Mansour, and Tim Roughgarden. Making the most of your samples. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2015.

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

[102] Philippe Jehiel, Moritz Meyer-Ter-Vehn, and Benny Moldovanu. Mixed bundling auctions. *Journal of Economic Theory*, 134(1):494–512, 2007.

[103] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed January 2019].

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

[105] Adam Tauman Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005.

[106] Fatma Kılınç Karzan, George L Nemhauser, and Martin Savelsbergh. Information-based branching schemes for binary linear mixed integer problems. *Mathematical Programming Computation*, 1(4):249–293, 2009.

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

[108] 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.

[109] 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.

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

[111] Yuri Kochetov and Dmitry Ivanenko. Computationally difficult instances for the uncapacitated facility location problem. In *Metaheuristics: Progress as real problem solvers*, pages 351–367. Springer, 2005.

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

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

[114] Anshul Kothari, David Parkes, and Subhash Suri. Approximately-strategyproof and tractable multi-unit auctions. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 166–175, San Diego, CA, 2003.

[115] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 489–504, 2018.

[116] Vijay Krishna. *Auction Theory*. Academic Press, 2002.

[117] 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.

[118] Sébastien Lahaie, Andrés Munoz Medina, Balasubramanian Sivan, and Sergei Vassilvitskii. Testing incentive compatibility in display ad auctions. In *Proceedings of the International World Wide Web Conference (WWW)*, 2018.

[119] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, pages 497–520, 1960.

[120] Kate Larson and Tuomas Sandholm. Costly valuation computation in auctions. In *Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, pages 169–182, Siena, Italy, July 2001.

[121] Kate Larson and Tuomas Sandholm. Mechanism design and deliberative agents. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 650–656, Utrecht, The Netherlands, 2005.

[122] Ron Lavi, Ahuva Mu'Alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 574–583, 2003.

[123] Pierre Le Bodic and George Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, pages 1–37, 2017.

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

[125] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

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

[127] Erez Lieberman-Aiden, Nynke L. van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R. Lajoie, Peter J. Sabo, Michael O. Dorschner, Richard Sandstrom, Bradley Bernstein, M. A. Bender, Mark Groudine, Andreas Gnirke, John Stamatoyannopoulos, Leonid A. Mirny, Eric S. Lander, and Job Dekker. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 2009. ISSN 0036-8075. doi: 10.1126/science.1181369.

[128] Anton Likhodedov and Tuomas Sandholm. Methods for boosting revenue in combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 232–237, San Jose, CA, 2004.

[129] Anton Likhodedov and Tuomas Sandholm. Approximating revenue-maximizing combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, PA, 2005.

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

[131] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

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

[133] Benjamin Lubin and David C Parkes. Approximate strategyproofness. *Current Science*, pages 1021–1032, 2012.

[134] Darío G Lupiáñez, Malte Spielmann, and Stefan Mundlos. Breaking TADs: how alterations of chromatin domains result in disease. *Trends in Genetics*, 32(4):225–237, 2016.

[135] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning (ICML)*, 2018.

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

[137] R Preston McAfee. Amicable divorce: Dissolving a partnership with simple mechanisms. *Journal of Economic Theory*, 56(2):266–293, 1992.

[138] Andrés Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[139] Timo Mennle and Sven Seuken. An axiomatic approach to characterizing and relaxing strategyproofness of one-sided matching mechanisms. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2014.

[140] Debasis Mishra and Arunava Sen. Roberts' theorem with neutrality: A social welfare ordering approach. *Games and Economic Behavior*, 75(1):283–298, 2012.

[141] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 464–473, 2018.

[142] Mehryar Mohri and Andrés Muñoz Medina. Learning theory and algorithms for revenue optimization in second price auctions with reserve. In *International Conference on Machine Learning (ICML)*, 2014.

[143] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

[144] John Morgan, Kenneth Steiglitz, and George Reis. The spite motive and equilibrium behavior in auctions. *Contributions to Economic Analysis & Policy*, 2(1), 2003.

[145] Jamie Morgenstern and Tim Roughgarden. On the pseudo-dimension of nearly optimal auctions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[146] Jamie Morgenstern and Tim Roughgarden. Learning simple auctions. In *Conference on Learning Theory (COLT)*, 2016.

[147] Swaprava Nath and Tuomas Sandholm. Efficiency and budget balance in general quasi-linear domains. *Games and Economic Behavior*, 113:673 – 693, 2019.

[148] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.

[149] Tri-Dung Nguyen and Tuomas Sandholm. Multi-option descending clock auction. Technical report, 2015. Mimeo.

[150] Ruth Nussinov and Ann B Jacobson. Fast algorithm for predicting the secondary structure of single-stranded rna. *Proceedings of the National Academy of Sciences*, 77(11):6309–6313, 1980.

[151] Walter Y Oi. A Disneyland dilemma: Two-part tariffs for a Mickey Mouse monopoly. *The Quarterly Journal of Economics*, 85(1):77–96, 1971.

[152] Abraham Othman and Tuomas Sandholm. Better with byzantine: Manipulation-optimal mechanisms. In *International Symposium on Algorithmic Game Theory (SAGT)*, Cyprus, 2009.

[153] Lior Pachter and Bernd Sturmfels. Parametric inference for biological sequence analysis. *Proceedings of the National Academy of Sciences*, 101(46):16138–16143, 2004. doi: 10.1073/pnas.0406011101.

[154] Lior Pachter and Bernd Sturmfels. Tropical geometry of statistical models. *Proceedings of the National Academy of Sciences*, 101(46):16132–16137, 2004. doi: 10.1073/pnas.0406010101.

[155] David Parkes. Optimal auction design for agents with hard valuation problems. In *Agent-Mediated Electronic Commerce Workshop at the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[156] Rachel Parkin. A year in first-price. *Ad Exchanger*, 2018. URL https://adexchanger.com/the-sell-sider/a-year-in-first-price/.

[157] Parag A Pathak and Tayfun Sönmez. School admissions reform in Chicago and England: Comparing mechanisms by their vulnerability to manipulation. *American Economic Review*, 103(1):80–106, 2013.

[158] László Patthy. Detecting homology of distantly related proteins with consensus sequences. *Journal of molecular biology*, 198(4):567–577, 1987.

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

[160] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9661–9670, 2018.

[161] Kevin Roberts. The characterization of implementable social choice rules. In J-J Laffont, editor, *Aggregation and Revelation of Preferences*. North-Holland Publishing Company, 1979.

[162] Michael Rothkopf, Thomas Teisberg, and Edward Kahn. Why are Vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109, 1990.

[163] Tim Roughgarden and Okke Schrijvers. Ironing in the dark. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2016.

[164] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

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

[166] 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.

[167] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 256–262, Washington, D.C., July 1993.

[168] Tuomas Sandholm. Issues in computational Vickrey auctions. *International Journal of Electronic Commerce*, 4(3):107–129, 2000. Early version in ICMAS-96.

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

[170] Tuomas Sandholm. Automated mechanism design: A new application area for search algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 19–36, Cork, Ireland, 2003.

[171] 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.

[172] Tuomas Sandholm and Craig Boutilier. Preference elicitation in combinatorial auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, pages 233–263. MIT Press, 2006. Chapter 10.

[173] Tuomas Sandholm, Anton Likhodedov, and Andrew Gilpin. Automated design of revenue-maximizing combinatorial auctions. *Operations Research*, 63(5):1000–1025, 2015. Special issue on Computational Economics. Subsumes and extends over a AAAI-05 paper and a AAAI-04 paper.

[174] David Sankoff and Robert J Cedergren. Simultaneous comparison of three or more sequences related by a tree. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison/edited by David Sankoff and Joseph B. Krustal*, 1983.

[175] J. Michael Sauder, Jonathan W. Arthur, and Roland L. Dunbrack Jr. Large-scale comparison of protein sequence alignment algorithms with structure alignments. *Proteins: Structure, Function, and Bioinformatics*, 40(1):6–22, 2000.

[176] Tzur Sayag, Shai Fine, and Yishay Mansour. Combining multiple heuristics. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 242–253. Springer, 2006.

[177] Ankit Sharma and Tuomas Sandholm. Asymmetric spite in auctions. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[178] Vasilis Syrgkanis. A sample complexity measure with applications to learning optimal auctions. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[179] Pingzhong Tang and Tuomas Sandholm. Mixed-bundling auctions with reserve prices. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012.

[180] Pingzhong Tang and Tuomas Sandholm. Optimal auctions for spiteful bidders. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[181] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[182] Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2): 264–280, 1971.

[183] Hal R. Varian. Position auctions. *International Journal of Industrial Organization*, pages 1163–1178, 2007.

[184] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

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

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

[187] 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.

[188] H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.

[189] Robert B Wilson. *Nonlinear pricing*. Oxford University Press on Demand, 1993.

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

[191] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Satzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008.

[192] Andrew Chi-Chih Yao. An n-to-1 bidder reduction for multi-item auctions and its applications. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.