# Automated algorithm and mechanism configuration

Ellen Vitercik

CMU-CS-21-125
August 2021

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Maria-Florina Balcan (co-chair)
Tuomas Sandholm (co-chair)
Ameet Talwalkar
Eric Horvitz (Microsoft)
Kevin Leyton-Brown (University of British Columbia)

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



This research was sponsored by the National Science Foundation under a Graduate Research Fellowship (1252522 and 1745016) and grant numbers CCF-1422910, CCF-1535967, IIS-1618714, IIS-1718457, and IIS-1901403; Microsoft; a Microsoft Research Women's Fellowship; an IBM Ph.D. Fellowship; and a fellowship from Carnegie Mellon University's Center for Machine Learning and Health. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

**Keywords:** automated algorithm design, data-driven algorithm design, automated algorithm configuration, sample complexity, machine learning theory, mechanism design

*For my parents, Carol and Greg*

## Abstract

Algorithms from diverse domains often have tunable parameters that have a significant impact on 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 tools for solving combinatorial and non-convex 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 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.

## Acknowledgments

First and foremost, thank you to Nina and Tuomas for an exhilarating PhD experience. You have an incredible synergy as co-advisors and your guidance has covered every single aspect of research: internalizing prior research; formulating new and exciting research problems; solving those problems; collaborating with others; writing papers, grants, and reviews; incorporating feedback; giving talks; and mentoring students. You have a gift for formulating research problems that are not only fascinating, but have solutions that are plausibly within reach (given a concerted effort!). I recognize that this is one of the most important qualities of an advisor, but it cannot be taken for granted. Whenever I work on research without the two of you, I ask myself, "What would Nina and Tuomas do?"

I've also had the opportunity to collaborate with other incredible researchers, including Daniel Alabi, Dan DeBlasio, Nika Haghtalab, Bernhard Haeupler, Carl Kingsford, Katrina Ligett, Cam Musco, Vaishnavh Nagarajan, Amir Shahrasbi, Umar Syed, and Christos Tzamos. Thank you to Christian Borgs, Jennifer Chayes, Adam Tauman Kalai, Andrés Muñoz Medina, and Sergei Vassilvitskii for being awesome internship hosts. Special thanks to Travis Dick and my other labmate collaborators Siddharth Prasad and Colin White for always making research fun. I'm also privileged to have a tremendous thesis committee: Eric Horvitz, Kevin Leyton-Brown, and Ameet Talwalkar. Thank you for the time you've taken out of your exceedingly busy schedules to mentor and advise me. Special thanks to Adam, Jennifer, and Kevin for being exceptional advocates.

I've loved school since the beginning, all thanks to some outstanding teachers. The Lincoln Community School (and the Lincoln community more broadly) is a gem. Thank you to Ellen Fenn, Anna Howell, Bill Jesdale, Alice Leeds, and Tory Riley for always making sure I was academically challenged. Majoring in math and learning about theoretical computer science at Columbia was a truly awesome experience, thanks to my inordinately talented professors—especially Al Aho, Allison Bishop, Adam Cannon, Anand Deopurkar, and Mike Woodbury.

Pittsburgh has been the ideal place to do a PhD not just for the academics but also for the exceptional community. Thank you to Naama Ben-David, Angela Jiang, Nic Resch, and David Wajc—I'm so glad we all started together[1]. Thank you to Zan Gilani for miraculously ending up in Pittsburgh and staying for almost as long as I did. Thank you to Nika Haghtalab for being the ultimate role model. Thank you to John Wright for teaching me how to give talks and always staying in touch. Thank you to Priya Donti for being an inspirational officemate. Thank you also to Noam Brown, Gabriele Farina, Bailey Flannigan, Ellis Hershkowitz, Anson Kahng, Ryan Kavanagh, Roie Levin, Colin White, and so many others for the dinners, coffee breaks, kayaking trips, orchestra concerts, hikes, and practice talks. I'm also grateful for my everlasting Vermont friendships: Bridgette Bartlett, Grace Blewer, Yuki Davis, Adrienne Lueders-Dumont, and Liz Saslaw. Thank you for reminding me that there's life beyond academia!

Most importantly: my parents and Rohan. Thank you to my dad, for raising me in a household where music was always playing and for wholeheartedly helping

[1]David started a year before, but close enough!

me devote myself to music growing up, which taught me focus and determination. And thank you to my mom, for being—by far—the coolest person I'll ever meet (sorry Dad and Rohan, but I know you'll agree!) and for modeling an unparalleled level of independent-mindedness. Thank you both for lots of other things too! And of course, thank you to Rohan, my best friend for 10 years going on 100, "even still"!

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

An important property of those algorithms that are typically used in practice is *broad applicability*—the ability to solve problems across diverse domains. For example, integer programming solvers, which are the most widely-used tools for solving combinatorial problems, have a huge array of applications, including routing, manufacturing, scheduling, planning, and many others. However, broadly applicable algorithms can have unsatisfactory default, out-of-the-box performance: they can have slow runtime or return poor-quality solutions, among other pitfalls.

One key challenge is that these broadly-applicable algorithms often come with many tunable parameters. This is true, for example, of integer programming solvers like CPLEX and Gurobi: CPLEX comes with a 170-page manual describing 172 tunable parameters [IBM ILOG Inc, 2017], which can have a significant impact on the time it takes CPLEX to find an optimal solution. Tuning these parameters by hand is a notoriously time-consuming, tedious, and error-prone process. This raises the question: how should one find the best configuration of these parameters for the specific application domain at hand? Whichever configuration is best for solving the routing problems a shipping company must solve day after day is likely not well-suited for the scheduling problems an airline must solve.

In practice, we often have ample data about the specific application domain in which we will use the algorithm, data we could potentially harness in the process of algorithm configuration. For example, a shipping company has access to all of the routing problems it had to solve over the course of a year, and an airline has access to all of the scheduling problems it has to solve day after day. If we could use this data together with machine learning to automate the process of algorithm configuration, then we could shift the burden of parameter tuning from human to machine, which would allow us to solve more problems faster, and in conjunction, these would provide significant boons in both profit and welfare.

Our goal is to provide practical procedures that come with rigorous theoretical guarantees for integrating machine learning and optimization into the process of algorithm design, and in particular, algorithm configuration. Since the early 2000s, researchers have proposed myriad ways in which machine learning can aid algorithm design, leading to breakthroughs in constraint satisfaction programming and integer programming [e.g., Horvitz et al., 2001, Hutter et al., 2009, 2011, Kadioglu et al., 2010, Sandholm, 2013, Xu et al., 2008, 2011], among many other fields. Starting in the mid-2010s, there has been a surge of interest in this topic from a theoretical perspective, with research on automated algorithm configuration and selection [Balcan, 2020, Balcan et al., 2017, 2018a,b,c, 2020a,b,c,e, 2021a,c, Blum et al., 2021, Cohen-Addad and

Kanade, 2017, Garg and Kalai, 2018, Gupta and Roughgarden, 2017, Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019]—the focus of this thesis—as well as learning-augmented algorithms, where worst-case algorithms are endowed with machine-learned hints about the input distribution [Hsu et al., 2019, Kraska et al., 2018, Lykouris and Vassilvitskii, 2018, Mitzenmacher, 2018, Purohit et al., 2018].

This thesis covers the use of machine learning in the context of algorithm design in a variety of different settings, including integer programming algorithms, algorithms for economic contexts (or *mechanisms*), and computational biology algorithms. It also exposes overarching structure linking these seemingly disparate domains which has allowed us to provide very general procedures for using machine learning in the context of algorithm design, together with unifying theoretical guarantees.

## 1.1 A machine learning approach to automated configuration

The majority of the results in this thesis apply to a well-studied approach to automated algorithm configuration based on a classic machine learning paradigm called *batch learning*. First, we fix an algorithm with parameters we aim to optimize, such as an integer programming solver. The configuration procedure receives as input a batch of typical problem instances from the specific application domain at hand, referred to as the *training set*. This set could consist of, for example, all of the routing integer programs that a shipping company in Pittsburgh had to solve over the course of a year. Demand changes every day, so the shipping company will have to solve new routing integer programs daily. Though different, these routing integer programs share similar underlying structure—for example, the underlying road network remains the same. Therefore, we can hope that if we find a configuration that works well on data from the past—our training set—it will also have strong performance on problems from the same application that we will encounter in the future. Performance can be measured, for example, in terms of the algorithm's runtime, its solution quality, or its memory usage, among other metrics.

There are several important questions we need to answer about this approach to algorithm configuration:

1. *How* should we find a configuration with strong average empirical performance over the training set? An extensive line of research has studied this question from an applied perspective [e.g., Hutter et al., 2009, 2011, Kadioglu et al., 2010, Sandholm, 2013, Xu et al., 2008, 2011], but only recently has it been investigated from a theoretical perspective. In Part II of this thesis, we provide a procedure that *provably* finds nearly-optimal configurations.

2. *No matter what procedure we use to optimize the parameters*—automated or manual, optimal or suboptimal—can we guarantee that any configuration's average empirical performance over the training set is indicative of its future performance on problems from the same application domain but which are not already in our training set? Learning theory warns us that if the parameterized algorithm is highly complex and the training set is too small, a parameter setting may lead to strong average empirical performance over the training set but poor future performance on unseen instances. In other words, the configuration procedure may *overfit* to the training set. This raises the question: how large should the training set be to ensure that for any parameter setting, its average empirical performance

over the training set is close to its expected, future performance? This type of bound is known as a *sample complexity guarantee*. In Part I of this thesis, we thoroughly investigate this question of sample complexity.

One of the key challenges we face in providing these types of provable guarantees for algorithm configuration is that in these combinatorial domains, an algorithm's performance—measured, for example, in terms of its runtime or solution quality—is a notoriously volatile function of its parameters, with many jump discontinuities. Intuitively, this is because an infinitesimal change in the algorithm's parameters can cause a cascade of changes in its behavior, triggering large jumps in its performance. This is unlike functions we understand well from a theoretical perspective in machine learning, where there is typically a close connection between a function's parameters and its value on any given input. Since we don't have this predictable behavior in combinatorial algorithm configuration, we must understand: what structure *is* there which will allow us to provide provable guarantees? In this thesis, we uncover this structure for a diverse array of algorithm configuration problems, with the chapters organized as follows.

**Chapter 2: Background and related research.** We frame the algorithm configuration problem formally, using a learning-theoretic model of algorithm configuration studied first by Gupta and Roughgarden [2017] and by many subsequent papers [Balcan, 2020, Balcan et al., 2017, 2018a,b,c, 2020a,e, 2021a,c, Blum et al., 2021, Garg and Kalai, 2018]. We then introduce classic tools from theoretical machine learning that we will use throughout this thesis, including *pseudo-dimension* and *Rademacher complexity*. These tools allow us to quantify the *intrinsic complexity* of a parameterized algorithm, which then allows us to derive sample complexity guarantees. To help put the results in this thesis into a broader context, we describe related research on the use of machine learning in the context of algorithm design, focusing on theoretical research.

**Chapter 3: Integer quadratic programming algorithms.** We begin by analyzing 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 4: Integer linear programming algorithms.** We next analyze 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 a machine learning approach can significantly improve tree size.

**Chapter 5: 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 impact, for example, the mechanism's revenue. We provide sample complexity bounds for tuning these mechanism parameters.

**Chapter 6: Computational 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 7: Pseudo-dimension bounds for piecewise-structured performance functions.** In our analysis of the diverse algorithm and mechanism 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, the algorithm's performance—for example, its runtime or solution quality—is a piecewise-structured function of its parameters (for example, the function is piecewise-constant or -linear). This structure has also been observed in contexts beyond those studied in this thesis, including clustering and greedy algorithm configuration [Balcan et al., 2017, 2018c, 2020a, Gupta and Roughgarden, 2017]. In this chapter, we provide a broadly-applicable theorem that recovers these prior sample complexity bounds. We also describe the relationship to general structure used for no-regret online algorithm configuration [Balcan et al., 2018b, 2020b,c] in Sections 2.2 and 7.1.

**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 4.

**Chapter 9: Portfolio-based algorithm selection.** Chapters 3-8 focused on the problem of learning a *single* parameter configuration with strong future, expected performance. A more flexible approach is to learn a *portfolio* of multiple parameter configurations and then at runtime, use the configuration with the strongest predicted performance. This approach has seen enormous success in practice. In this chapter, we provide end-to-end, provable guarantees for learning a portfolio in conjunction with an *algorithm selector*, which dictates which configuration in the portfolio should be used at runtime.

**Chapter 10: 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 5). 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.

Part I of this thesis studied statistical challenges that arise when using machine learning in the context of algorithm design. Next, Part II studies algorithmic challenges. We analyze *how* to find provably high-performing configurations over a training set of typical problem instances and other means of integrating machine learning into algorithm design.

**Chapter 11: Frugal training with generalization guarantees.** A recent line of research provides algorithms that return nearly-optimal parameter settings from within a finite set [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019, 2020]. 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 12: Learning to prune: Speeding up repeated computations.** In the final chapter of this thesis, 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.

# Part I

# Sample complexity guarantees

# Chapter 2

# Background and related research

## 2.1 Notation and learning theory background

In Part I of this thesis, we provide sample complexity guarantees for algorithm configuration. Here we summarize the general problem statement together with notation and learning-theoretic tools that we will use throughout. We use a learning-theoretic model of algorithm configuration studied first by Gupta and Roughgarden [2017] and by many subsequent papers [Balcan, 2020, Balcan et al., 2017, 2018a,b,c, 2020a,e, 2021a,c, Blum et al., 2021, Garg and Kalai, 2018]. Throughout the thesis, we adopt notation used in a paper by Balcan et al. [2018b]. Each chapter will also include its own domain-dependent background.

Throughout the thesis, 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 consist 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. Depending on the context, we may wish to minimize this performance metric (for example, when it measures runtime) or maximize it (for example, when it measures solution quality). 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 $z_1, \ldots, z_N \sim \mathcal{D}$, 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 $\hat{\rho} = \text{argmax}_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{z \in \mathcal{S}} u_{\boldsymbol{\rho}}(z)$. With probability at least $1 - \delta$ over the draw of $N \geq N_{\mathcal{U}}(\epsilon, \delta)$ samples, $\max_{\boldsymbol{\rho} \in \mathcal{P}} \mathbb{E}_{z \sim \mathcal{D}} \left[ u_{\boldsymbol{\rho}}(z) \right] - \mathbb{E}_{z \sim \mathcal{D}} \left[ u_{\hat{\rho}}(z) \right] \leq 2\epsilon$. A symmetric argument holds if our goal is to find a parameter vector $\boldsymbol{\rho}$ that minimizes $u_{\boldsymbol{\rho}}(z)$ as opposed to maximizes it.

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.

### 2.1.1 Pseudo-dimension

Pseudo-dimension [Pollard, 1984] 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 [Pollard, 1984])**.** 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 $\text{Pdim}(\mathcal{U}) = |\mathcal{S}|$.

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 $\text{VCdim}(\mathcal{U})$ [Vapnik and Chervonenkis, 1971].

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

**Theorem 2.1.3** (Pollard [1984])**.** *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 $\epsilon$: $\left| \frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(z_i) - \mathbb{E}_{z \sim \mathcal{D}} \left[ u_{\boldsymbol{\rho}}(z) \right] \right| \leq \epsilon$.*

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.

We will often use the following lemma to derive pseudo-dimension bounds.

**Lemma 2.1.4** (Shalev-Shwartz and Ben-David [2014]). *Let $a \geq 1$ and $b > 0$. Then $x < a \log x + b$ implies that $x < 4a \log(2a) + 2b$.*

### 2.1.2 Rademacher complexity

*Rademacher complexity* [Bartlett and Mendelson, 2002, Koltchinskii, 2001] 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 2.1.5** (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

$$\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{N} \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \{-1,1\}^N} \left[ \sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] u_{\boldsymbol{\rho}}(z_i) \right],$$

where each $\sigma[i]$ equals $-1$ or $1$ with equal probability.

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

**Theorem 2.1.6** (e.g., Mohri et al. [2012]). *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 $\boldsymbol{\rho} \in \mathcal{P}$, $\left| \frac{1}{N} \sum_{i=1}^{N} u_{\boldsymbol{\rho}}(z_i) - \mathbb{E}_{z \sim \mathcal{D}} \left[ u_{\boldsymbol{\rho}}(z) \right] \right| = O\left( \widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{U}) + H \sqrt{\frac{1}{N} \ln \frac{1}{\delta}} \right)$.*

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

**Theorem 2.1.7** (Pollard [1984]). *Let $\mathcal{U}$ be a set of functions mapping $\mathcal{Z}$ to $[-H, H]$. For any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathcal{R}}_{\mathcal{S}}(\mathcal{U}) = O\left( H \sqrt{\frac{\operatorname{Pdim}(\mathcal{U})}{N}} \right)$.*

Massart [2000] proved the following result which we use several times in this thesis.

**Lemma 2.1.8** (Massart [2000]). *Let $A \subseteq [0, 1]^N$ be a finite set of vectors. Then*

$$\frac{1}{N} \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \{-1,1\}^N} \left[ \sup_{\boldsymbol{a} \in A} \sum_{i=1}^{N} \sigma[i] a[i] \right] \leq \sqrt{\frac{2 \ln |A|}{N}}.$$

## 2.2 Related research

To help put the results in this thesis into a broader context, we describe related research on the use of machine learning in the context of algorithm design, focusing on theoretical research. In each chapter, we describe additional research related to the chapter's content.

**Online algorithm configuration.** A related line of research studies theoretical guarantees for algorithm configuration in the *online learning* model [Balcan et al., 2018b, 2020b,c, Cohen-Addad and Kanade, 2017, Gupta and Roughgarden, 2017], in contrast to the batch learning model described in Sections 1.1 and 2.1. As in the batch learning model, there is a fixed algorithm whose parameters we aim to configure. In the online learning model, problem instances (for example, integer programs) arrive one-by-one over a series of timesteps. These problem instances need not be drawn i.i.d. from any distribution—they may even be adversarially selected. At each time step, the learner chooses a configuration of the algorithm, receives a problem instance, and receives a reward (or penalty) which equals the performance of the algorithm (for example, its runtime or solution quality) with the learner's chosen configuration given that timestep's problem instance as input. The goal is to minimize *regret*, which is the difference between the learner's cumulative reward (or penalty) and that of the optimal configuration in hindsight.

Regret minimization is impossible in the worst case [Gupta and Roughgarden, 2017], but prior research shows that under some conditions, it is possible. For several algorithm configuration problems defined by a single tunable parameter, Gupta and Roughgarden [2017] and Cohen-Addad and Kanade [2017] prove positive results (small constant per-round regret or even no regret) in cases where instances are "smoothed," *a la* smoothed analysis [Spielman and Teng, 2004]. Balcan et al. [2018b, 2020b,c] show that no-regret online algorithm configuration is possible when the algorithm's performance is a piecewise-Lipschitz function of its parameters and the boundaries between pieces do not concentrate. These papers prove that across many parameterized algorithms, the algorithm's performance is indeed piecewise-Lipschitz, and under mild assumptions, the boundaries do not concentrate. Balcan, Dick, and Vitercik [2018b] also provide sample complexity bounds for the batch learning setting under the same assumptions. This piecewise-Lipschitz property is even more general than the structure we formalize and exploit in Chapter 7 to provide batch learning guarantees—namely, that the algorithm's performance is a "piecewise-structured" function of its parameters. However, the online and batch learning guarantees from prior research [Balcan et al., 2018b, 2020b,c] require the additional structure that the boundaries between pieces are dispersed—an assumption that is not needed for the batch learning sample complexity bounds in Chapter 7. Chapter 7, however, only applies to batch learning, not online learning. We expand on this distinction in Section 7.1.

**Clustering algorithm configuration.** Several papers provide configuration procedures and sample complexity guarantees for clustering algorithms, an area not covered in this thesis. Balcan et al. [2017] provide generalization bounds for parameterized agglomerative clustering algorithms, where the parameters define the merge function used to build the agglomerative clustering hierarchy. Balcan et al. [2020a] study how to learn *both* a distance metric and a merge function. They provide an efficient configuration algorithm that learns near-optimal distance and merge functions from within several infinite classes, with extensive experiments. Balcan et al. [2018c] study a parameterized clustering algorithm that generalizes Lloyd's method, and includes the k-means++ algorithm as a special case. Finally, Garg and Kalai [2018] provide batch learning sample complexity guarantees for clustering algorithm configuration, using bit-representation arguments to bound the sample complexity of optimizing the parameters of several clustering algorithms.

**Mechanism configuration.** A long line of research studies how machine learning can be used to design and analyze mechanisms, beginning with research by Likhodedov and Sandholm [2004, 2005] and Balcan et al. [2005, 2008]. We provide many more references and a detailed comparison of our research with prior research in Section 5.1.

**Algorithm scheduling.** A related theoretical direction studies a learning-theoretic model for algorithm scheduling [Sayag et al., 2006, Streeter and Golovin, 2009, Streeter et al., 2007], which is a similar but distinct problem from algorithm configuration. In their setup, there are multiple algorithms capable of computing a correct solution to a given 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, these papers show how to learn a *schedule* with nearly minimum expected cost.

**Algorithm configuration procedures with guarantees on frugality of training.** A line of research by Kleinberg et al. [2017, 2019] and Weisz et al. [2018, 2019] provides learning-based algorithm configuration procedures with provable guarantees. Their algorithms return nearly-optimal parameter settings from within a finite set, where the parameterized algorithm's performance is measured in terms of runtime. In that finite setting, the primary challenge is to find a nearly-optimal configuration as quickly as possible. Those algorithms can also be used when the parameter space is infinite by first sampling $\tilde{\Omega}(1/\gamma)$ configurations for some $\gamma \in (0,1)$ and then running the algorithm over this finite set. The authors guarantee that the output configuration will be within the top $\gamma$-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, as we prove in Chapter 4. In Chapter 11, we work towards marrying the results presented in Part I on sample complexity guarantees for infinite parameter spaces with this line of research by Kleinberg et al. [2017, 2019] and Weisz et al. [2018, 2019]. Subsequent research by Weisz et al. [2020] extends the configuration procedure by Weisz et al. [2019], enabling it to quickly discard suboptimal configurations without running the risk of discarding nearly-optimal configurations. It can thus operate with a better choice of $\gamma$ compared to prior research [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019].

**Algorithms aided by machine learned advice.** A related line of research designs algorithms that are aided by "machine learned advice" [e.g., Hsu et al., 2019, Lykouris and Vassilvitskii, 2018, Mitzenmacher, 2018, Purohit et al., 2018]. 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.

# Chapter 3

# Integer quadratic programming algorithms

In this chapter, we study algorithm configuration for a class of integer quadratic programming approximation algorithms. The results in this chapter are joint work with Nina Balcan, Vaishnavh Nagarajan, and Colin White and they appeared in COLT 2017 [Balcan et al., 2017]. Many NP-hard problems, such as max-cut, max-2SAT, and correlation clustering, can be represented as an integer quadratic program (IQP). IQPs appear frequently in machine learning applications, such as MAP inference [Frostig et al., 2014, Huang et al., 2014, Zhong et al., 2014] and image segmentation and correspondence problems in computer vision [Brendel and Todorovic, 2010, Cour et al., 2006]. Max-cut is an important problem that can be formulated as an IQP and its applications in machine learning include community detection [Bandeira et al., 2016], variational methods for graphical models [Risteski and Li, 2016], and graph-based semi-supervised learning [Wang et al., 2013]. The seminal Goemans-Williamson max-cut algorithm is now a textbook example of semidefinite programming [Goemans and Williamson, 1995, Vazirani, 2013, Williamson and Shmoys, 2011]. Max-cut also arises in many other scientific domains, such as circuit design [Yoshimura et al., 2015] and computational biology [Snir and Rao, 2006].

We focus on IQPs 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 Charikar and Wirth [2004].

Most algorithms with the best approximation guarantees use a *semi-definite programming (SDP)* relaxation. The class of algorithms that we study consists of SDP rounding algorithms and is a generalization of the seminal Goemans-Williamson (GW) max-cut algorithm Goemans and Williamson [1995]. The SDP relaxation has the form

$$\text{maximize} \sum_{i,j\in[n]} a_{ij}\langle \boldsymbol{w}_i, \boldsymbol{w}_j\rangle \qquad \text{subject to } \boldsymbol{w}_i \in S^{n-1}. \tag{3.1}$$

Given the set of vectors $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\}$, 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 $\mathcal{N}$. 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

**Algorithm 1** SDP rounding algorithm with rounding function $r$

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

1: Draw a random vector $\boldsymbol{Z}$ from $\mathcal{N}$, the $n$-dimensional Gaussian distribution.
2: Solve the SDP (3.1) for the optimal embedding $U = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\}$.
3: Compute set of fractional assignments $r(\langle \boldsymbol{Z}, \boldsymbol{w}_1 \rangle), \ldots, r(\langle \boldsymbol{Z}, \boldsymbol{w}_n \rangle)$.
4: For all $i \in [n]$, set $x_i$ to 1 with probability $\frac{1}{2} + \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle\right)$ and $-1$ with probability $\frac{1}{2} - \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle\right)$.

**Output:** $x_1, \ldots, x_n$.

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 $\frac{1}{2} + \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle\right)$ and $-1$ with probability $\frac{1}{2} - \frac{1}{2} \cdot r\left(\langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle\right)$. See Algorithm 1 for the pseudocode. Algorithm 1 is known as a *Random Projection, Randomized Rounding* (RPR$^2$) algorithm, so named by the seminal work of Feige and Langberg [2006].

We focus on the class of $s$-linear rounding functions in this section. For the max-cut problem, Feige and Langberg [2006] 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 $\phi_s$ is defined as follows:

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

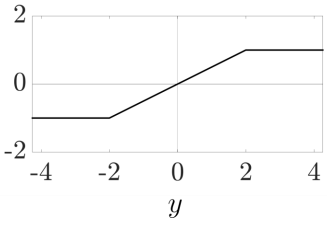



Figure 3.1: A graph of the 2-linear function $\phi_2$.

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 $\sum_{i,j \in [n]} a_{ij} x_i x_j$ 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 $A$, $\boldsymbol{Z}$, and the choice of a parameter $s$. This expected value is thus over distributions that are both external and internal to Algorithm 1: 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 1. 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 $\mathcal{S} = \left\{ \left(A^{(1)}, \boldsymbol{Z}^{(1)}\right), \ldots, \left(A^{(N)}, \boldsymbol{Z}^{(N)}\right) \right\} \sim (\mathcal{D} \times \mathcal{N})^N$. 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 1 given $A$ as input when it uses the hyperplane $\boldsymbol{Z}$ and the $s$-linear rounding function $\phi_s$ 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 1 given $A$ as input when it uses the $s$-linear rounding function $\phi_s$ 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 1, 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 $\mathbb{E}_{A,\boldsymbol{Z}\sim\mathcal{D}\times\mathcal{N}}\left[\mathbb{E}_{x_i\sim p_{(i,\boldsymbol{Z},A,s)}}\left[\sum_{i,j} a_{ij}x_i x_j\right]\right]$.[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.

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 1 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 $\mathbb{E}_{A,\boldsymbol{Z}\sim\mathcal{D}\times\mathcal{N}}\left[u_s(A, \boldsymbol{Z})\right]$.

We now define the empirical utility of a parameter $s$ as follows. Given a set of samples $\left(A^{(1)}, \boldsymbol{Z}^{(1)}\right), \ldots, \left(A^{(N)}, \boldsymbol{Z}^{(N)}\right) \sim \mathcal{D} \times \mathcal{N}$, 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 $\mathbb{E}_{A,\boldsymbol{Z}\sim\mathcal{D}\times\mathcal{N}}\left[u_s(A, \boldsymbol{Z})\right]$. 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 3.0.4, we provide a sample efficient and computationally efficient algorithm for finding $\hat{s}$.

[1] We use the abbreviated notation

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

[2] Since pseudo-dimension bounds imply uniform convergence guarantees for worst-case distributions, the distribution $\mathcal{N}$ over vectors need not be Gaussian, although this is the classic distribution of choice in the works by Goemans and Williamson [1995] and Feige and Langberg [2006]. Indeed, our results hold when $\mathcal{N}$ is any arbitrary distribution over $\mathbb{R}^n$.

We begin by characterizing the analytic form of $u_s$, which allows us to bound the pseudo-dimension of $\mathcal{U}$.

**Lemma 3.0.1.** *Given a matrix $A$ and a vector $\boldsymbol{Z}$, let $u_s(A, \boldsymbol{Z})$ denote the expected value of the solution returned by Algorithm 1 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. Then*

$$u_s(A, \boldsymbol{Z}) = \sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \phi_s(\langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle) \cdot \phi_s(\langle \boldsymbol{Z}, \boldsymbol{w}_j \rangle).$$

*Proof.* The expected value of the solution returned by Algorithm 1 given $A$ as input when it uses the hyperplane $\boldsymbol{Z}$ and the rounding function $r = \phi_s$ in Step 3 is

$$\begin{aligned} \mathop{\mathbb{E}}_{x_i \sim p_{(i,\boldsymbol{Z},A,s)}} \left[ \sum_{i,j \in [n]} a_{ij} x_i x_j \right] &= \sum_{i,j \in [n]} \mathop{\mathbb{E}}_{x_i, x_j} \left[ a_{ij} x_i x_j \right] \\ &= \sum_{i=1}^{n} a_{ii}^2 \mathop{\mathbb{E}}_{x_i} [x_i^2] + \sum_{i \neq j} a_{ij} \mathop{\mathbb{E}}_{x_i, x_j} \left[ x_i x_j \right]. \end{aligned}$$

Since the support of $p_{(i,\boldsymbol{Z},A,s)}$ is $\{-1, 1\}$, we know that

$$\sum_{i=1}^{n} a_{ii}^2 \mathop{\mathbb{E}}_{x_i} [x_i^2] + \sum_{i \neq j} a_{ij} \mathop{\mathbb{E}}_{x_i, x_j} \left[ x_i x_j \right] = \sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \mathop{\mathbb{E}}_{x_i, x_j} \left[ x_i x_j \right].$$

The draw $x_i \sim p_{(i,\boldsymbol{Z},A,s)}$ is independent from the draw $x_j \sim p_{(j,\boldsymbol{Z},A,s)}$, so

$$\sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \mathop{\mathbb{E}}_{x_i, x_j} \left[ x_i x_j \right] = \sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \mathop{\mathbb{E}}_{x_i \sim p_{(i,\boldsymbol{Z},A,s)}} [x_i] \mathop{\mathbb{E}}_{x_j \sim p_{(j,\boldsymbol{Z},A,s)}} \left[ x_j \right].$$

Since

$$\mathop{\mathbb{E}}_{x_i \sim p_{(i,\boldsymbol{Z},A,s)}} [x_i] = \frac{1}{2} + \frac{1}{2} \cdot \phi_s \left( \langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle \right) - \left( \frac{1}{2} - \frac{1}{2} \cdot \phi_s \left( \langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle \right) \right) = \phi_s \left( \langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle \right),$$

this means that

$$u_s(A, \boldsymbol{Z}) = \sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \mathop{\mathbb{E}}_{x_i} [x_i] \mathop{\mathbb{E}}_{x_j} \left[ x_j \right] = \sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \phi_s(\langle \boldsymbol{Z}, \boldsymbol{w}_i \rangle) \cdot \phi_s(\langle \boldsymbol{Z}, \boldsymbol{w}_j \rangle).$$

Putting all of these equalities together, the lemma statement holds. □

Using this lemma, 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 3.0.2.** *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 $s = |\langle \boldsymbol{w}_i, \boldsymbol{Z} \rangle|$ for some $\boldsymbol{w}_i$ in the optimal SDP embedding of $A$.*

**Algorithm 2** An algorithm for finding an empirical value maximizing $s$-linear rounding function

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

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

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 $\left|\left\langle \boldsymbol{Z}^{(i)}, \boldsymbol{w}_j^{(i)} \right\rangle\right| = s$.

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:** $\hat{s}$

*Proof.* Let $X = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\} \subset S^{n-1}$ be the optimal embedding of $A$. We may write $u_{A,\boldsymbol{Z}}(s) = \sum_{i=1}^{n} a_{ii}^2 + \sum_{i \neq j} a_{ij} \phi_s(v_i) \cdot \phi_s(v_j)$, where $v_i = \langle \boldsymbol{w}_i, \boldsymbol{Z} \rangle$ and $v_j = \langle \boldsymbol{w}_j, \boldsymbol{Z} \rangle$. For any $i \in [n]$, the specific form of $\phi_s(v_i)$ depends solely on whether $|v_i| \leq s$ or $|v_i| > s$ (recall that $s > 0$, by definition). So long as $|v_i| > s$, we know $\phi_s(v_i) = \pm 1$, where the sign depends on the sign of $v_i$. Otherwise, when $|v_i| < s$, $\phi_s(v_i) = v_i/s$. Therefore, if we order the set of real values $\{|v_i|, \ldots, |v_n|\}$, then so long as $s$ falls between two consecutive elements of this ordering, the form of $u_{A,\boldsymbol{Z}}(s)$ is fixed. In particular, each summand is either a constant, a constant multiplied by $\frac{1}{s}$, or a constant multiplied by $\frac{1}{s^2}$. This means that we may partition the positive real line into $n+1$ intervals where the form of $u_{A,\boldsymbol{Z}}(s)$ is a fixed quadratic function, as claimed. □

This fact implies the following pseudo-dimension bound.

**Theorem 3.0.3.** *The pseudo-dimension of $\mathcal{U} = \{u_s : s > 0\}$ is $O(\ln n)$.*

*Proof.* We prove this upper bound by showing that if a set $\mathcal{S}$ of size $N$ is shatterable, then $N = O(\log n)$. This means that the largest shatterable set must be of size $O(\log n)$, so the pseudo-dimension of $\mathcal{U}$ is $O(\log n)$. We arrive at this bound by fixing a tuple $\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right) \in \mathcal{S}$ and analyzing $u_{A,\boldsymbol{Z}}(s)$. In particular, we make use of Lemma 3.0.2, from which we know that $u_{A,\boldsymbol{Z}}(s)$ is composed of $n+1$ piecewise quadratic components. Therefore, if $t_i$ is the target corresponding to the element $\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$, we can partition the positive real line into at most $3(n+1)$ intervals where $u_{A,\boldsymbol{Z}}(s)$ is always either less than its target $t_i$ or greater than $t_i$ as $s$ varies over one fixed interval. The constant 3 term comes from the fact that for a single, continuous quadratic component of $u_{A,\boldsymbol{Z}}(s)$, the function may equal $t_i$ at most twice, so there are at most three subintervals where the function is less than or greater than $t_i$.

The set $\mathcal{S}$ consists of $N$ tuples $\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$, each of which corresponds to its own partition of the positive real line. If we merge these partitions, we are left with at most $(3n+2)N + 1$ intervals such that for all $i \in [N]$, $u_{A^{(i)}, \boldsymbol{Z}^{(i)}}(s)$ is always either less than its target $t_i$ or greater than $t_i$ as $s$ varies over one fixed interval. In other words, in one interval, the binary labeling of $\mathcal{S}$, defined by whether each sample is less than or greater than its target, is fixed. This means that if $\mathcal{S}$ is shatterable, the $2^N$ values of $s$ which induce all $2^N$ binary labelings of $\mathcal{S}$ must come from distinct intervals. Therefore $2^N \leq (3n+2)N + 1$, so $N = O(\log n)$. □

Lemma 3.0.2 also suggests a learning algorithm, Algorithm 2, that is computationally and sample efficient.

**Theorem 3.0.4.** *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 $\mathcal{D} \times \mathcal{N}$, let $\hat{s}$ be the output of Algorithm 2. With probability at least $1 - \delta$, the true utility of $\hat{s}$ is $\epsilon$-close optimal:*

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

*Proof.* Let $\mathcal{S} = \left\{\left(A^{(1)}, \boldsymbol{Z}^{(1)}\right), \ldots, \left(A^{(N)}, \boldsymbol{Z}^{(N)}\right)\right\}$ be a sample of size $N$. First, we prove that Algorithm 2 on input $\mathcal{S}$ returns the value $\hat{s}$ which maximizes $\frac{1}{N}\sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$ in polynomial time. In Lemma 3.0.2, we prove that each function $u_{A^{(i)},\boldsymbol{Z}^{(i)}}(s)$ is made up of at most $n+1$ piecewise components of the form $\frac{a}{s^2} + \frac{b}{s} + c$ for some $a, b, c \in \mathbb{R}$. Therefore, $\frac{1}{N}\sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$ is made up of at most $Nn+1$ piecewise components of the form $\frac{a}{s^2} + \frac{b}{s} + c$ as well. Moreover, by Lemma 3.0.2, if the border between two components falls at some $s \in \mathbb{R}_{>0}$, then it must be that $\left|\left\langle \boldsymbol{Z}^{(i)}, \boldsymbol{w}_j^{(i)} \right\rangle\right| = s$ for some $\boldsymbol{w}_j^{(i)}$ in the optimal max-cut SDP embedding of $A^{(i)}$. These are the thresholds which are computed in Step 2 of Algorithm 2. Therefore, as we increase $s$ starting at 0, $s$ will be a fixed inverse-quadratic function between the thresholds, so it is simple to find the optimal value of $s$ between any pair of consecutive thresholds (Step 3), and then the value maximizing $\frac{1}{N}\sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right)$ (Step 4), which is the global optimum.

Next, from Theorem 3.0.3, we have that with $N = O\left(\left(\frac{H}{\epsilon}\right)^2 \left(\log n + \log \frac{1}{\delta}\right)\right)$ samples, with probability at least $1 - \delta$, for all $s > 0$,

$$\left| \frac{1}{N} \sum_{i=1}^{N} u_s\left(A^{(i)}, \boldsymbol{Z}^{(i)}\right) - \mathop{\mathbb{E}}_{(A,\boldsymbol{Z})\sim\mathcal{D}\times\mathcal{N}} [u_s(A, \boldsymbol{Z})] \right| < \frac{\epsilon}{2}.$$

Since this is true for the parameter $\hat{s}$ returned by Algorithm 2 and for the optimal parameter $s^* = \text{argmax}_{s>0}\, \mathbb{E}_{A\sim\mathcal{D},\boldsymbol{Z}\sim\mathcal{N}} [u_s(A, \boldsymbol{Z})]$, we know that with probability at least $1 - \delta$,

$$\mathop{\mathbb{E}}_{A\sim\mathcal{D},\boldsymbol{Z}\sim\mathcal{N}} [u_{s^*}(A, \boldsymbol{Z})] - \mathop{\mathbb{E}}_{A\sim\mathcal{D},\boldsymbol{Z}\sim\mathcal{N}} [u_{\hat{s}}(A, \boldsymbol{Z})] \leq \epsilon,$$

as claimed. □

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

# Chapter 4

# 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., [Russell and Norvig, 2010, Williams, 2013]). For example, branch-and-bound (B&B) algorithms [Land and Doig, 1960] 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 [Liberatore, 2000].

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 [2009] 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, and the majority appeared in ICML 2018 [Balcan et al., 2018a].

## 4.1 Related research

Algorithm selection and configuration have been studied for decades, dating back to a seminal paper by Rice [1976]. In this section, we highlight related research in the context of algorithm configuration for tree search.

### 4.1.1 Industrial uses

Automated algorithm configuration has had success in industry already. Sandholm [2013] used it to configure integer programming approaches for winner determination in $60 billion of combinatorial auctions from 2001 to 2010. He used automated configuration approaches both to select among modeling choices and choices within the integer programming algorithms themselves. Since then, all the major commercial integer programming solvers have started to ship with algorithm configuration tools that allow customers to automatically tune the solver's parameters to their specific problems. For example, CPLEX introduced an automated parameter tuning tool in 2007 [IBM ILOG Inc, 2007].

### 4.1.2 Algorithm configuration procedures for tree search

A high-level distinction among automated algorithm configuration approaches is whether one is trying to find one configuration that performs well for the entire application-specific distribution over problem instances, or one is trying to select different configurations for different problems instances based on problem instance features. In this chapter we focus on the former

problem. The latter approach has also been used Kadioglu et al. [2010], Leyton-Brown et al. [2009], Sandholm [2013], Xu et al. [2008, 2011]. That work, too, has been empirical and has not offered generalization guarantees.

**General algorithm configuration procedures.** Hutter et al. [2009] provide an algorithm called ParamILS which uses iterated local search to find high-performing parameter settings, employing a careful adaptive capping procedure to make sure time is not wasted evaluating bad configurations. Ansótegui et al. [2009] propose an algorithm called GGA which relies on genetic algorithms to find high-performing configurations. ISAC [Kadioglu et al., 2010] (short for "instance specific algorithm configuration") uses GGA to learn a mapping from problem instances to high-performing parameter settings, in contrast with ParamILS and GGA, which learn a single parameter setting with strong expected performance.

Whereas ParamILS and GGA are *model-free* approaches, the next generation of algorithm configuration procedures, such as SMAC [Hutter et al., 2011], are *model-based* approaches. SMAC iterates between fitting models that predict algorithmic performance and using those models to select promising parameter settings to further investigate.

**Configuring variable-selection policies.** As in the present chapter, Khalil et al. [2016] 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. [2017] study a similar problem, although in their paper, the feature vectors in the training set describe nodes from multiple MILP instances. Neither of these papers come with theoretical guarantees, unlike the present chapter.

Several other papers also study data-driven variable selection from a purely experimental perspective. Di Liberto et al. [2016] devise an algorithm that learns how to dynamically switch between different branching heuristics in the tree. Karzan et al. [2009] propose techniques for choosing instance-specific branching rules based on first running a small partial search on the instance. In the context of CSP and SAT solving, several papers use a multi-armed bandit [Liang et al., 2016, Xia and Yap, 2018] or reinforcement learning [Lagoudakis and Littman, 2001] approach to optimize variable-selection policies.

From a theoretical perspective, Le Bodic and Nemhauser [2017] present a variable-selection model based on an abstraction to a simpler setting in which it is possible to analytically evaluate the dual bound improvement of choosing a given variable. Based on that model, they present a new variable selection policy which has strong performance on many MIPLIB instances.

**Configuring additional aspects of tree search.** Researchers have explored the use of machine learning techniques in the context of other aspects of B&B beyond variable selection, such as Sandholm [2013], who studies, for example, how to learn which MIP modeling techniques and cutting planes to use, Sabharwal et al. [2012] and He et al. [2014a], who study how to learn node selection policies, Kruber et al. [2017], who study how to detect decomposable model structure, and Khalil et al. [2017], who study how to determine when to run primal heuristics. Machine learning has also been used in a variety of ways to automate CSP solver configuration. The vast majority of the papers in that literature are empirical, whereas we provide theoretical guarantees also. More information about many of those papers can be found in the recent survey by Lodi and Zarpellon [2017].

**Beyond algorithm configuration.** Researchers have also studied many problems closely related to algorithm configuration, such as runtime prediction [Fischetti et al., 2019, Horvitz et al., 2001, Hutter et al., 2014], as well as the compilation and utilization of algorithm portfolios [Xu et al., 2008, 2010].

## 4.2 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 4.6, we provide another example of tree search for constraint satisfaction problems. 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 4.2.1** (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.

**Algorithm 3** 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** $\tilde{z} \in \left\{z_i^+, z_i^-\right\}$ **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 $\tilde{z}$.

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 4.2) 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 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 3 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.

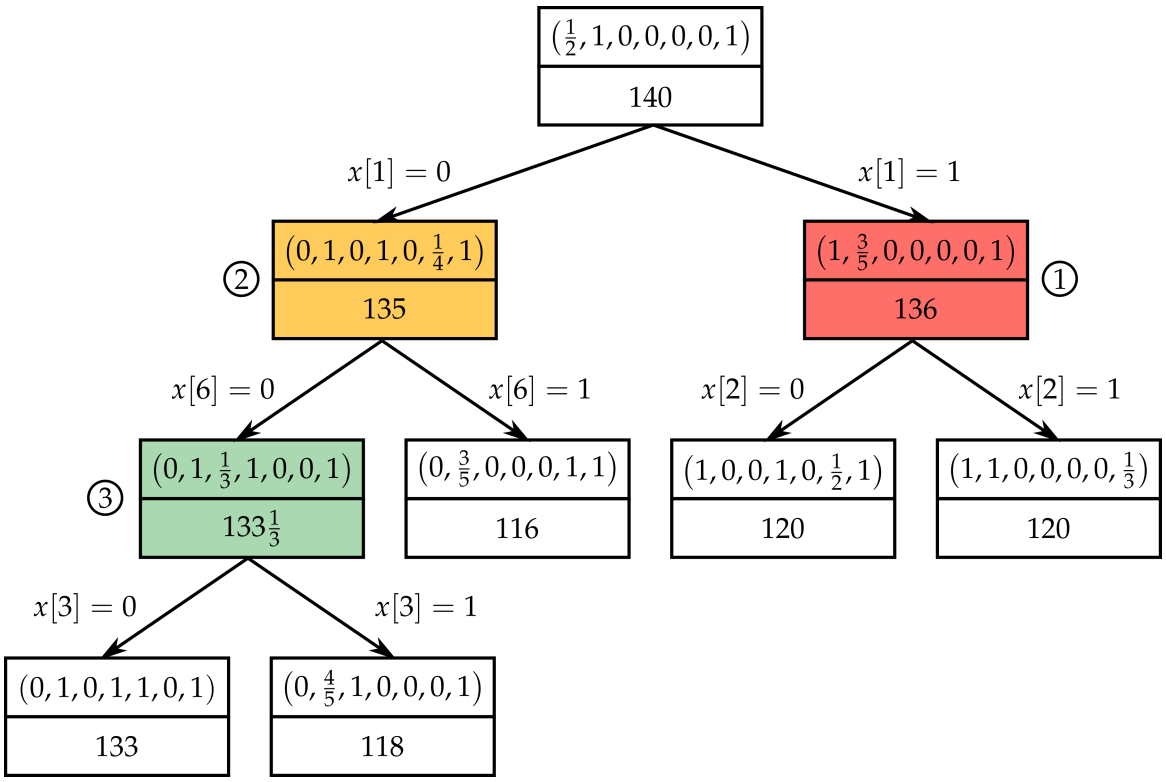



Figure 4.1: Illustration of Example 4.2.2.

**Example 4.2.2.** In Figure 4.1, we show the search tree built by B&B given as input the following MILP [Kolesar, 1967]:

$$\begin{array}{ll} \text{maximize} & 40x[1] + 60x[2] + 10x[3] + 10x[4] + 3x[5] + 20x[6] + 60x[7] \\ \text{subject to} & 40x[1] + 50x[2] + 30x[3] + 10x[4] + 10x[5] + 40x[6] + 30x[7] \leq 100 \\ & x[1], \ldots, x[7] \in \{0, 1\}. \end{array} \tag{4.1}$$

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 (4.1) 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 4.1, 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 (4.1). 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 4.2.3** (Score-based variable selection policy)**.** Let $\text{score}$ 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 ($\text{score}(\mathcal{T}, z, i) \in \mathbb{R}$). 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 $\text{argmax}_{x[j] \in N_{\mathcal{T},z}} \{\text{score}(\mathcal{T}, z, j)\}$ to branch on at the node $z$.

We list several common definitions of the function $\text{score}$ 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, $\text{score}(\mathcal{T}, z, i) = \min\{1 - \check{x}_z[i], \check{x}_z[i]\}$. The variable that maximizes $\text{score}(\mathcal{T}, z, i)$ 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 [Linderoth and Savelsbergh, 1999].** In this case, $\text{score}(\mathcal{T}, z, i) = (1-\rho) \cdot \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\} + \rho \cdot \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$ 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 $\rho = 1$, 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 $\rho = 0$.

**Product scoring rule [Achterberg, 2009].** In this case, $\text{score}(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^-}, \gamma\right\} \cdot \max\left\{\check{c}_z - \check{c}_{z_i^+}, \gamma\right\}$ where $\gamma = 10^{-6}$. Comparing $\check{c}_z - \check{c}_{z_i^-}$ and $\check{c}_z - \check{c}_{z_i^+}$ to $\gamma$ 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 $\gamma$. 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 [Gilpin and Sandholm, 2011].** 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 $\text{score}(\mathcal{T}, z, i) = -\sum_{j=1}^{n} (1 - \check{x}_z[i]) \cdot e\left(\check{x}_{z_i^-}[j]\right) + \check{x}_z[i] \cdot e\left(\check{x}_{z_i^+}[j]\right)$.

**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 $\check{x}_z$ 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 $\text{score}(\mathcal{T}, z, i) = (1-\rho) \cdot \min\left\{\check{c}_z - \tilde{c}_{z_i^+}, \check{c}_z - \tilde{c}_{z_i^-}\right\} + \rho \cdot \max\left\{\check{c}_z - \tilde{c}_{z_i^+}, \check{c}_z - \tilde{c}_{z_i^-}\right\}$ 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* [Bénichou et al., 1971, Gauthier and Ribière, 1977, Linderoth and Savelsbergh, 1999] and *reliability branching* [Achterberg et al., 2005].

**Pseudo-cost branching.** 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. Pseudo-cost branching is alternative to computing these values [Bénichou et al., 1971, Gauthier and Ribière, 1977, Linderoth and Savelsbergh, 1999]. To introduce pseudo-cost branching, we first define some notation, as presented in Achterberg's thesis [Achterberg, 2007]. At a node $z$, let $f_{z,i}^- = \check{x}_z[i] - \lfloor \check{x}_z[i] \rfloor$ and $f_{z,i}^+ = \lceil \check{x}_z[i] \rceil - \check{x}_z[i]$ denote how far the $i^{th}$ component of the LP relaxation's solution is from being integral. Let $\varsigma_{z,i}^-$ and $\varsigma_{z,i}^+$ be the objective gains per unit change in variable $x[i]$ at node $z$ after branching in each direction. More formally,

$$\varsigma_{z,i}^- = \frac{\check{c}_z - \check{c}_{z_i^-}}{f_{z,i}^-} \quad \text{and} \quad \varsigma_{z,i}^+ = \frac{\check{c}_z - \check{c}_{z_i^+}}{f_{z,i}^+}.$$

Let $\sigma_i^-$ be the sum of $\varsigma_{z,i}^-$ over all nodes $z$ where $x[i]$ was chosen as the branching variable and the LP relaxation of the node $z_i^-$ has already been solved. Let $\eta_i^-$ be the number of such nodes. Let $\sigma_i^+$ and $\eta_i^+$ be the corresponding values for the upwards branches. The pseudo-costs of variable $x[i]$ are defined as

$$\Psi_i^- = \frac{\sigma_i^-}{\eta_i^-} \quad \text{and} \quad \Psi_i^+ = \frac{\sigma_i^+}{\eta_i^+}.$$

We initialize the pseudo-costs using strong branching: if $\eta_i^- = 0$ when we are choosing which variable to branch on at a node $z$, we set

$$\Psi_i^- = \frac{\check{c}_z - \check{c}_{z_i^-}}{f_{z,i}^-} \tag{4.2}$$

and similarly for $\Psi_i^+$.

In pseudo-cost branching, we estimate $\check{c}_z - \check{c}_{z_i^-}$ using the value $\Psi_i^- f_{z,i}^-$ and we estimate $\check{c}_z - \check{c}_{z_i^+}$ using the value $\Psi_i^+ f_{z,i}^+$. For example, the linear scoring rule with parameter $\mu \in [0,1]$ using pseudo-cost branching is defined as

$$\text{score}(\mathcal{T}, z, i) = (1-\mu)\min\left\{\Psi_i^- f_{z,i}^-, \Psi_i^+ f_{z,i}^+\right\} + \mu \max\left\{\Psi_i^- f_{z,i}^-, \Psi_i^+ f_{z,i}^+\right\}.$$

**Reliability branching.** Reliability branching [Achterberg et al., 2005] as a variation on pseudo-cost branching. Variable $i$'s pseudo-costs are said to be *unreliable* if $\min\left\{\eta_i^-, \eta_i^+\right\} < \eta_{\text{rel}}$, where $\eta_{\text{rel}} \in \mathbb{Z}$ is a tunable parameter. We set the pseudo-costs of unreliable variables as in Equation (4.2). We refer the reader to Achterberg's thesis [Achterberg, 2007] for guidance about how to tune the parameter $\eta_{\text{rel}}$.

## 4.3 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.

### 4.3.1 Problem statement

Let $\mathcal{D}$ be a distribution over MILPs $z$. 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 set of variable-selection scoring rules, such as those in Section 4.2. Our high-level goal is to provide sample complexity guarantees for learning nearly-optimal convex combinations $\sum_{i=1}^d \rho[i]\text{score}_i$ of the scoring rules. 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 the size of the tree B&B builds. In the special case where we tune the tradeoff between two scoring rules $\rho\text{score}_1 + (1-\rho)\text{score}_2$, we use the notation $u_\rho(z)$. Our goal is to bound the pseudo-dimension of $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in [0,1]^d\right\}$. We also generalize our analysis to cover non-linear combinations of the form

$$\prod_{i=1}^d \text{score}_i^{\rho[i]} \tag{4.3}$$

.

Following prior work [e.g., Hutter et al., 2009, Kleinberg et al., 2017], we assume that there is some cap $\kappa$ on the range of the cost function $\text{cost}$. For example, if $\text{cost}$ is the size of the search tree, we may choose to terminate the algorithm when the tree size grows beyond some bound $\kappa$. We also assume that the problem instances in the support of $\mathcal{D}$ are over $n$ variables for some $n \in \mathbb{N}$.

Our results hold for utility functions that are *tree-constant*, which means that for any problem instance $z$, so long as the parameter vectors $\boldsymbol{\rho}$ and $\boldsymbol{\rho}'$ 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.

**Prior research on combining scoring rules.** For decades, convex combinations of scoring rules $\sum_{i=1}^{d} \rho[i]\text{score}_i$ have been used in B&B. For example, suppose

$$\text{score}_1(\mathcal{T}, z, i) = \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\} \text{ and } \text{score}_2(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\},$$

so by employing the scoring rule $(1-\rho)\text{score}_1 + \rho\text{score}_2$, we balance an "optimistic" and "pessimistic" approach to branching. Over time, researchers have proposed many different candidates for the parameter setting $\rho$. Gauthier and Ribière [1977] proposed setting $\rho = 1/2$. Bénichou et al. [1971] and Beale [1979] suggested setting $\rho = 1$. Linderoth and Savelsbergh [1999] found that $\rho = 2/3$ performs well. Achterberg [2009] found that experimentally, $\rho = 1/6$ performed best when comparing among $\rho \in \{0, 1/6, 1/3, 1/2, 1\}$. In our experiments, we find that the best choice of a parameter setting depends on the specific application domain at hand; no one parameter setting is optimal overall. In Section 4.3.2, we also prove that no one parameter setting is always optimal, and in fact we prove something significantly stronger: searching over any data-independent discretization of parameter values can lead to extremely bad algorithm configurations.

Non-linear combinations of scoring rules are also popular in B&B. SCIP [Achterberg, 2009], the best open-source solver, uses the product scoring rule $\text{score} = \text{score}_1\text{score}_2$ where

$$\text{score}_1(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^-}, 10^{-6}\right\}$$

and $\text{score}_2(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, 10^{-6}\right\}$. This is equivalent to Equation (4.3) with $\rho[1] = \rho[2] = \frac{1}{2}$. In our experiments, we show that alternative parameter choices can outperform this baseline.

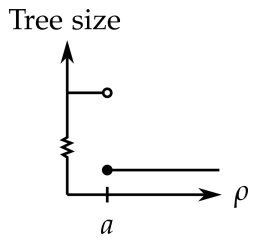
### 4.3.2 Impossibility results for data-independent approaches

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 [1977] proposed setting $\rho = 1/2$, Bénichou et al. [1971] and Beale [1979] suggested setting $\rho = 1$, and Linderoth and Savelsbergh [1999] found that $\rho = 2/3$ performs well. Achterberg [2009] found that experimentally, $\rho = 5/6$ performed best when comparing among $\rho \in \{0, 1/2, 2/3, 5/6, 1\}$.

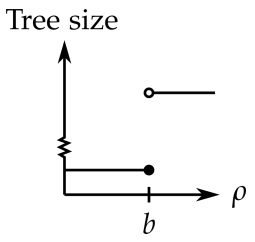
We show that for *any* discretization of the parameter space $[0, 1]$, there exists an infinite family of distributions over MILP problem instances such that for any parameter in the discretization, the expected tree size is exponential in $n$. Yet, there exists an infinite number of parameters such that the tree size is just a constant (with probability 1).

**Theorem 4.3.1.** *Let*

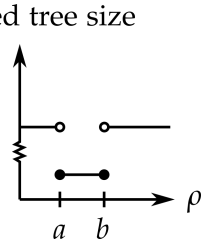
$$\text{score}_1(\mathcal{T}, z, i) = \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}, \text{score}_2(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\},$$




(a) The tree size plot for the instance $z_a$ as a function of $\rho$.




(b) The tree size plot for the instance $z_b$ as a function of $\rho$.




(c) The expected tree size plot under the distribution $\mathcal{D}$ as a function of $\rho$.

Figure 4.2: Illustrations of the proof of Theorem 4.3.1.

*and $u_\rho(z)$ be the size of the tree produced by B&B using the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$. 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.*

*Proof.* We populate the support of the distribution $\mathcal{D}$ by relying on two helpful theorems: Theorem 4.3.3 and 4.3.8. In Theorem 4.3.3, we prove that for all $\rho^* \in \left(\frac{1}{3}, \frac{2}{3}\right)$, there exists an infinite family $\mathcal{F}_{n,\rho^*}$ of MILP instances such that for any $z \in \mathcal{F}_{n,\rho^*}$, if $\rho \in [0,\rho^*)$, then the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ results in a B&B tree with $O(1)$ nodes and if $\rho \in (\rho^*, 1]$, the scoring rule results a tree with $2^{(n-4)/2}$ nodes. Conversely, in Theorem 4.3.8, we prove that there exists an infinite family $\mathcal{G}_{n,\rho^*}$ of MILP instances such that for any $z \in \mathcal{G}_{n,\rho^*}$, if $\rho \in [0,\rho^*)$, then the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ results in a B&B tree with $2^{(n-5)/4}$ nodes and if $\rho \in (\rho^*, 1]$, the scoring rule results a tree with $O(1)$ nodes.

Now, let $z_a$ be an arbitrary instance in $\mathcal{G}_{n,a}$ and let $z_b$ be an arbitrary instance in $\mathcal{F}_{n,b}$. The theorem follows by letting $\mathcal{D}$ be a distribution such that $\Pr_{z\sim\mathcal{D}}[z = z_a] = \Pr_{z\sim\mathcal{D}}[z = z_b] = 1/2$. See Figure 4.2 for an illustration. We know that if $\rho \in [0,1] \setminus (a,b)$, then the expected value of $u_\rho(z)$ is $\frac{1}{2}\left(O(1) + 2^{(n-5)/4}\right) \geq 2^{(n-9)/4}$. Meanwhile, if $\rho \in (a,b)$, then with probability 1,

$$u_\rho(z) = O(1).$$

Throughout the proof of this theorem, we assume the node-selection policy is depth-first search. We then prove that for any infeasible MILP, if NSP and NSP′ are two node-selection policies and $\text{score} = \rho\text{score}_1 + (1-\rho)\text{score}_2$ for any $\rho \in [0,1]$, then tree $\mathcal{T}$ B&B builds using NSP and score equals the tree $\mathcal{T}'$ it builds using NSP′ and score, as we prove in the following claim.

**Claim 4.3.2.** *Let $z$ be an infeasible MILP, let NSP and NSP′ be two node-selection policies, and let* score *be a path-wise scoring rule. The tree $\mathcal{T}$ B&B builds using NSP and* score *equals the tree $\mathcal{T}'$ it builds using NSP′ and* score*.*

*Proof of Claim 4.3.2.* For a contradiction, suppose $\mathcal{T} \neq \mathcal{T}'$. There must be a node $z_0$ in $\mathcal{T}$ where if $\mathcal{T}_{z_0}$ is the path from the root of $\mathcal{T}$ to $z_0$, then $\mathcal{T}_{z_0}$ is a rooted subtree of $\mathcal{T}'$, but either:

1. In $\mathcal{T}$, the node $z_0$ is fathomed but in $\mathcal{T}'$, $z_0$ is not fathomed, or
2. In $\mathcal{T}$, the node $z_0$ is not fathomed, but for all children ${z_0}'$ of $z_0$ in $\mathcal{T}$, if $\mathcal{T}_{{z_0}'}$ is the path from the root of $\mathcal{T}$ to ${z_0}'$, $\mathcal{T}_{{z_0}'}$ is not a rooted subtree of $\mathcal{T}'$.

We will show that neither case is possible, thus arriving at a contradiction. First, we know that since $z$ is infeasible, B&B will only fathom a node if it is infeasible. Therefore, the first case is impossible: if $z_0$ is fathomed in $\mathcal{T}$, it must be infeasible, so it will also be fathomed in $\mathcal{T}'$, and vice versa. Therefore, we know that B&B must branch on $z_0$ in both $\mathcal{T}$ and $\mathcal{T}'$. Let $\bar{\mathcal{T}}$ be the state of the tree B&B has built using NSP and score by the time it branches on $z_0$ and let $\bar{\mathcal{T}}'$ be the state of the tree B&B has built using NSP′ and score by the time it branches on $z_0$. Since $\mathcal{T}_{z_0}$ is a rooted subtree of both $\bar{\mathcal{T}}$ and $\bar{\mathcal{T}}'$, we know that for all variables $x[i]$, $\text{score}(\bar{\mathcal{T}}, z_0, i) = \text{score}(\mathcal{T}_z, z_0, i) = \text{score}(\bar{\mathcal{T}}', z_0, i)$. Therefore, B&B will branch on the same variable in both $\mathcal{T}$ and $\mathcal{T}'$, which is a contradiction, since this means that for all children ${z_0}'$ of $z_0$ in $\mathcal{T}$, if $\mathcal{T}_{{z_0}'}$ is the path from the root of $\mathcal{T}$ to ${z_0}'$, $\mathcal{T}_{{z_0}'}$ is a rooted subtree of $\mathcal{T}'$. □

Thus, the theorem holds for any node-selection policy. □

We now provide a proof sketch of Theorem 4.3.3, which helps us populate the support of the worst-case distributions in Theorem 4.3.1.

**Theorem 4.3.3.** *Let*

$$\text{score}_1(\mathcal{T}, z, i) = \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}, \text{score}_2(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\},$$

*and $u_\rho(z)$ be the size of the tree produced by B&B using the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$. For all even $n \geq 6$ and all $\rho^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, there exists an infinite family $\mathcal{F}_{n,\rho^*}$ of MILP instances such that for any $z \in \mathcal{F}_{n,\rho^*}$, if $\rho \in [0, \rho^*)$, then the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ results in a B&B tree with $O(1)$ nodes and if $\rho \in (\rho^*, 1]$, the scoring rule results a tree with $2^{(n-4)/2}$ nodes.*

For intuition, we begin with a proof sketch of Theorem 4.3.3, and then provide the full proof.

*Proof sketch.* The MILP instances in $\mathcal{F}_{n,\rho^*}$ are inspired by a worst-case B&B instance introduced by Jeroslow [1974]. He proved that for any odd $n'$, every B&B algorithm will build a tree with $2^{(n'-1)/2}$ nodes before it determines that for any $\boldsymbol{c} \in \mathbb{R}^{n'}$, the following MILP is infeasible:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} & 2\sum_{i=1}^{n'} x[i] = n' \\ & \boldsymbol{x} \in \{0,1\}^{n'}. \end{array}$$

We build off of this MILP to create the infinite family $\mathcal{F}_{n,\rho^*}$. Each MILP in $\mathcal{F}_{n,\rho^*}$ combines a hard version of Jeroslow's instance on $n-3$ variables $\{x[1], \ldots, x[n-3]\}$ and an easy version on 3 variables $\{x[n-2], x[n-1], x[n]\}$. Branch-and-bound only needs to determine that one of these problems is infeasible in order to terminate. The key idea of this proof is that if B&B branches on all variables in $\{x[n-2], x[n-1], x[n]\}$ first, it will terminate upon making a small tree. However, if B&B branches on all variables in $\{x[1], \ldots, x[n-3]\}$ first, it will create a tree with exponential size before it terminates. The challenge is to design an objective function that enforces the first behavior when $\rho < \rho^*$ and the second behavior when $\rho > \rho^*$. Proving this is the bulk of the work.

$$\begin{array}{ll}\text{maximize} & \left(1,2,3,4,5,0,\frac{3}{2},3-\frac{1}{2\mu^*}\right)\cdot \boldsymbol{x}\\ \text{subject to} & \begin{pmatrix}2&2&2&2&2&0&0&0\\0&0&0&0&0&2&2&2\end{pmatrix}\boldsymbol{x}=\begin{pmatrix}5\\3\end{pmatrix}\\ & \boldsymbol{x}\in\{0,1\}^8.\end{array}$$

(a) A big version of Jeroslow's instance on five variables.

$$\begin{array}{ll}\text{maximize} & \left(1,2,3,4,5,0,\frac{3}{2},3-\frac{1}{2\mu^*}\right)\cdot \boldsymbol{x}\\ \text{subject to} & \begin{pmatrix}2&2&2&2&2&0&0&0\\0&0&0&0&0&2&2&2\end{pmatrix}\boldsymbol{x}=\begin{pmatrix}5\\3\end{pmatrix}\\ & \boldsymbol{x}\in\{0,1\}^8.\end{array}$$

(b) A small version of Jeroslow's instance on three variables.

Figure 4.3: Illustrations of the construction from Theorem 4.3.3.

In a bit more detail, every instance in $\mathcal{F}_{n,\rho^*}$ is defined as follows. For any constant $\gamma \geq 1$, let $\boldsymbol{c}_1 = \gamma(1,2,\ldots,n-3)$ and let $\boldsymbol{c}_2 = \gamma\left(0,\frac{3}{2},3-\frac{1}{2\rho^*}\right)$. Let $\boldsymbol{c} = (\boldsymbol{c}_1,\boldsymbol{c}_2) \in \mathbb{R}^n$ be the concatenation of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Let $z_{\gamma,n}$ be the MILP

$$\begin{array}{ll}\text{maximize} & \boldsymbol{c}\cdot\boldsymbol{x}\\ \text{subject to} & 2\sum_{i=1}^{n-3}x[i]=n-3\\ & 2\left(x[n-2]+x[n-1]+x[n]\right)=3\\ & \boldsymbol{x}\in\{0,1\}^n.\end{array}$$

We define $\mathcal{F}_{n,\rho^*} = \{z_{n,\gamma} : \gamma \geq 1\}$.

For example, if $\gamma = 1$ and $n = 8$, then $z_{\gamma,n}$ is

$$\begin{array}{ll}\text{maximize} & \left(1,2,3,4,5,0,\frac{3}{2},3-\frac{1}{2\rho^*}\right)\cdot \boldsymbol{x}\\ \text{subject to} & \begin{pmatrix}2&2&2&2&2&0&0&0\\0&0&0&0&0&2&2&2\end{pmatrix}\boldsymbol{x}=\begin{pmatrix}5\\3\end{pmatrix}\\ & \boldsymbol{x}\in\{0,1\}^8.\end{array}$$

As is illustrated in Figure 4.3, we have essentially "glued together" two disjoint versions of Jeroslow's instance: the first five variables of $z_{1,8}$ correspond to a "big" version of Jeroslow's instance and the last three variables correspond to a small version. Since the goal is maximization, the solution to the LP relaxation of $z_{1,8}$ will try to obtain as much value from the first five variables $\{x[1],\ldots,x[5]\}$ as it can, but it is constrained to ensure that $2(x[1]+\cdots+x[5]) = 5$. Therefore, the first five variables will be set to $\left(0,0,\frac{1}{2},1,1\right)$. Similarly, the solution to the LP relaxation of $z_{1,8}$ will set $(x[6],x[7],x[8]) = \left(0,\frac{1}{2},1\right)$ because $\frac{3}{2} < 3-\frac{1}{2\rho^*}$ under our assumption that $\rho^* > \frac{1}{3}$. Thus, the solution to the LP relaxation of $z_{1,8}$ is $\left(0,0,\frac{1}{2},1,1,0,\frac{1}{2},1\right)$. There are only two fractional variables that B&B might branch on: $x[3]$ and $x[7]$. Straightforward calculations show that if $\mathcal{T}$ is the B&B tree so far, which just consists of the root node, $\rho\text{score}_1\left(\mathcal{T},z_{\gamma,n},3\right) + (1-\rho)\text{score}_2\left(\mathcal{T},z_{\gamma,n},3\right) = \frac{\gamma}{2}$ and $\rho\text{score}_1\left(\mathcal{T},z_{\gamma,n},7\right) + (1-\rho)\text{score}_2\left(\mathcal{T},z_{\gamma,n},7\right) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}$. This means that B&B will branch first on variable $x[7]$, which corresponds to the small version of Jeroslow's instance (see Figure 4.3b) if and only if $\frac{\gamma}{2} < \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}$, which occurs if and only if $\rho < \rho^*$. We show that this first branch sets off a cascade: if B&B branches first on variable $x[7]$, then it will proceed to branch on all variables in $\{x[6],x[7],x[8]\}$, thus terminating upon making a small tree. Meanwhile, if it branches on variable $x[3]$ first, it will then only branch on variables in $\{x[1],\ldots,x[5]\}$, creating a larger tree.

In the full proof, we generalize beyond eight variables to $n$, and expand the large version of Jeroslow's instance (as depicted in Figure 4.3a) from five variables to $n-3$. When $\rho < \rho^*$, we simply track B&B's progress to make sure it only branches on variables from the small

version of Jeroslow's instance $(x[n-2], x[n-1], x[n])$ before figuring out the MILP is infeasible. Therefore, the tree will have constant size. When $\rho > \rho^*$, we prove by induction that if B&B has only branched on variables from the big version of Jeroslow's instance $(x[1], \ldots, x[n-3])$, it will continue to only branch on those variables. We also prove it will branch on about half of these variables along each path of the B&B tree. The tree will thus have exponential size. □

Building off of this intuition, we now provide the full proof of Theorem 4.3.3, in which we will use the following notation. Given a MILP $z$, suppose that we branch on $x[i]$ and $x[j]$, setting $x[i] = 0$ and $x[j] = 1$. We use the notation $z_{i,j}^{-,+}$ to denote the resulting MILP. Similar, if we set $x[i] = 1$ and $x[j] = 0$, we denote the resulting MILP as $z_{i,j}^{+,-}$.

*Proof of Theorem 4.3.3.* For ease of notation in this proof, we will drop $\mathcal{T}$ from the input of the functions $\text{score}_1$ and $\text{score}_2$ since the scoring rules do not depend on $\mathcal{T}$, they only depend on the input MILP instance and variable.

For any constant $\gamma \geq 1$, let $\boldsymbol{c}_1 = \gamma(1, 2, \ldots, n-3)$ and let $\boldsymbol{c}_2 = \gamma\left(0, 1.5, 3 - \frac{1}{2\rho^*}\right)$. Let $\boldsymbol{c} = (\boldsymbol{c}_1, \boldsymbol{c}_2) \in \mathbb{R}^n$ be the concatenation of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Next, define the $n$-dimensional vectors $\boldsymbol{a}_1 = 2\sum_{i=1}^{n-3} \boldsymbol{e}_i$ and $\boldsymbol{a}_2 = 2\sum_{i=1}^{3} \boldsymbol{e}_{n-3+i}$, and let $A$ be a matrix whose first row is $\boldsymbol{a}_1$ and second row is $\boldsymbol{a}_2$. Let $z_{\gamma,n}$ be the MILP

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = (n-3, 3)^\top \\ & \boldsymbol{x} \in \{0,1\}^n. \end{array}$$

We define $\mathcal{F}_{n,\rho^*} = \{z_{n,\gamma} : \gamma \geq 1\}$.

**Example 4.3.4.** If $\gamma = 1$ and $n = 8$, then $z_{\gamma,n}$ is

$$\begin{array}{ll} \text{maximize} & \left(1, 2, 3, 4, 5, 0, 1.5, 3 - \frac{1}{2\rho^*}\right) \cdot \boldsymbol{x} \\ \text{subject to} & \begin{pmatrix} 2 & 2 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \end{pmatrix} \boldsymbol{x} = \begin{pmatrix} 5 \\ 3 \end{pmatrix} \\ & \boldsymbol{x} \in \{0,1\}^8. \end{array}$$

For every even $n \geq 6$, both of the constraints $\boldsymbol{a}_1 \cdot \boldsymbol{x} = 2\sum_{i=1}^{n-3} x[i] = n-3$ and $\boldsymbol{a}_2 \cdot \boldsymbol{x} = 2\sum_{i=1}^{3} x[n-3+i] = 3$ are infeasible for $\boldsymbol{x} \in \{0,1\}^n$ since $2\sum_{i=1}^{n-3} x[i]$ and $2\sum_{i=1}^{3} x[n-3+i]$ are even numbers but 3 and $n-3$ are odd numbers. The key idea of this proof is that if B&B branches on all variables in $\{x[n-2], x[n-1], x[n]\}$ first, it will terminate upon making a tree of size at most $2^3 = 8$, since at most three branches are necessary to determine that $\boldsymbol{a}_1 \cdot \boldsymbol{x} = 2\sum_{i=1}^{3} x[n-3+i] = 3$ is infeasible. However, if B&B branches on all variables in $\{x[1], \ldots, x[n-3]\}$ first, it will create a tree with exponential size before it terminates.

**Lemma 4.3.5.** *Suppose $\rho < \rho^*$. Then for any MILP $z_{\gamma,n} \in \mathcal{F}_{n,\rho^*}$, $\rho\text{score}_1 + (1-\rho)\text{score}_2$ branches on all variables in $\{x[n-2], x[n-1], x[n]\}$ before branching on variables in $\{x[1], \ldots, x[n-3]\}$.*

*Proof of Lemma 4.3.5.* For ease of notation, for the remainder of this proof, we drop the subscript $(\gamma, n)$ from $z_{\gamma,n}$ and denote this MILP as $z$. We first need to determine the form of $\check{\boldsymbol{x}}_z$, which is the optimal solution to the LP relaxation of $z$. It is easiest to see how the LP relaxation will set the variables $x[n-2]$, $x[n-1]$, and $x[n]$. The only constraints on these variables are that $2(x[n-2] + x[n-1] + x[n]) = 3$ and that $x[n-2], x[n-1], x[n] \in [0,1]$. Recall that

$\boldsymbol{c} = \left(1, 2, \ldots, n-3, 0, 1.5, 3 - \frac{1}{2\rho^*}\right)$ is the vector defining the objective value of $z$. Since $\rho^* > 1/3$, we know that $1.5 < 3 - \frac{1}{2\rho^*}$, which means that $c[n-2] < c[n-1] < c[n]$. Since the goal is to maximize $\boldsymbol{c} \cdot \boldsymbol{x}$, the LP relaxation will set $x[n-2] = 0$, $x[n-1] = \frac{1}{2}$, and $x[n] = 1$. The logic for the first $n-3$ variables is similar. In this case, $c[1] < c[2] < \cdots < c[n-3]$, so the LP relaxation's solution will put as much weight as possible on the variable $x[n-3]$, then as much weight as possible on the variable $x[n-4]$, and so on, putting as little weight as possible on the variable $x[1]$ since it has the smallest corresponding objective coefficient $c[1]$. Since the only constraints on these variables are that $2\sum_{i=1}^{n-3} x[i] = n-3$ and $x[1], \ldots, x[n-3] \in [0,1]$, the LP objective value can set $\left\lfloor \frac{n-3}{2} \right\rfloor$ of the variables to 1, it can set one variable to $\frac{1}{2}$, and it has to set the rest of the variables to 0. Letting $i = \left\lceil \frac{n-3}{2} \right\rceil$, this means the LP relaxation will set the first $i-1$ variables $x[1], \cdots x[i-1]$ to zero, it will set $x[i] = \frac{1}{2}$, and it will set $x[i+1], \ldots, x[n-3]$ to 1. In other words,

$$\check{\boldsymbol{x}}_z[j] = \begin{cases} 0 & \text{if } j \leq \lfloor (n-3)/2 \rfloor \text{ or } j = n-2 \\ \frac{1}{2} & \text{if } j = \lceil (n-3)/2 \rceil \text{ or } j = n-1 \\ 1 & \text{if } \lceil (n-3)/2 \rceil \leq j \leq n-3 \text{ or } j = n. \end{cases}$$

For example, if $n = 8$, then $\check{\boldsymbol{x}}_z = \left(0, 0, \frac{1}{2}, 1, 1, 0, \frac{1}{2}, 1\right)$. (See Figure 4.4a.) Therefore, the only candidate variables to branch on are $x[n-1]$ and $x[i]$ where again, $i = \lceil (n-3)/2 \rceil$.

To determine when variable B&B will branch on, we need to calculate $\check{\boldsymbol{x}}_{z_i^-}$ which is the solution to the LP relaxation of $z$ with the additional constraint that $x[i] = 0$, as well as $x_{z_i^+}$ which is the solution to the LP relaxation of $z$ with the additional constraint that $x[i] = 1$, and $x_{z_{n-1}^-}$ and $x_{z_{n-1}^+}$. First, we will determine the form of the vectors $\check{\boldsymbol{x}}_{z_{n-1}^-}$ and $\check{\boldsymbol{x}}_{z_{n-1}^+}$. Suppose we set $x[n-1] = 0$. We now need to ensure that $2(x[n-2] + 0 + x[n]) = 3$ and that $x[n-2], x[n] \in [0,1]$. Since $c[n-2] = 0 < 3 - \frac{1}{2\rho^*} = c[n]$, the solution to the LP relaxation of $z_{n-1}^-$ will set $x[n-2] = \frac{1}{2}$ and $x[n] = 1$. (See Figure 4.4b.) Similarly, if we set $x[n-1] = 1$, we now need to ensure that $2(x[n-2] + 1 + x[n]) = 3$ and that $x[n-2], x[n] \in [0,1]$. Therefore, the solution to the LP relaxation of $z_{n-1}^-$ will set $x[n-2] = 0$ and $x[n] = \frac{1}{2}$. (See Figure 4.4c.) In other words, $\check{\boldsymbol{x}}_{z_{n-1}^-} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{z_{n-1}^+} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$.

The argument for $\check{\boldsymbol{x}}_{z_i^-}$ and $\check{\boldsymbol{x}}_{z_i^-}$ is similar. Recall that in $\check{\boldsymbol{x}}_z$, the solution to the LP relaxation of the original MIP $z$, we have that $\check{x}_z[i] = \frac{1}{2}$ since it is the median of the variables $x[1], \ldots, x[n-3]$. For all $j > i$, we have that $\check{x}_z[j] = 1$ and for all $j < i$, we have that $\check{x}_z[j] = 0$. Suppose we set $x[i] = 0$. As before, the LP relaxation's solution will put as much weight as possible on the variable $x[n-3]$, then as much weight as possible on the variable $x[n-4]$, and so on, putting as little weight as possible on the variable $x[1]$ since it has the smallest corresponding objective coefficient $c[1]$. Since it cannot set $x[i] = \frac{1}{2}$, it will set the next-best variable to $\frac{1}{2}$, which is $x[i-1]$. (See Figure 4.4d.) In other words, $\check{\boldsymbol{x}}_{z_i^-} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_i + \frac{1}{2}\boldsymbol{e}_{i-1}$. If we set $x[i] = 1$, the LP relaxation's solution will have to take some weight away from the variables $x[i+1], \ldots, x[n-3]$ since it needs to ensure that $2\sum_{i=1}^{n-3} x[i] = n-3$. Therefore, it will set $x[i+1]$ to $\frac{1}{2}$ and $x[j]$ to 1 for all $j > i+1$. (See Figure 4.4e.) In other words, $\check{\boldsymbol{x}}_{z_i^+} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{i+1}$.

Therefore,

$$\check{c}_{z_i^-} = \check{\boldsymbol{c}}_z - \frac{1}{2}\left\lceil \frac{n-3}{2} \right\rceil + \frac{1}{2}\left\lfloor \frac{n-3}{2} \right\rfloor = \check{c}_z - \frac{\gamma}{2},$$

$$\check{\boldsymbol{c}}_{z_i^+} = \check{\boldsymbol{c}}_z + \frac{1}{2}\left\lceil \frac{n-3}{2} \right\rceil - \frac{1}{2}\left(\left\lceil \frac{n-3}{2} \right\rceil + 1\right) = \check{c}_z - \frac{\gamma}{2},$$

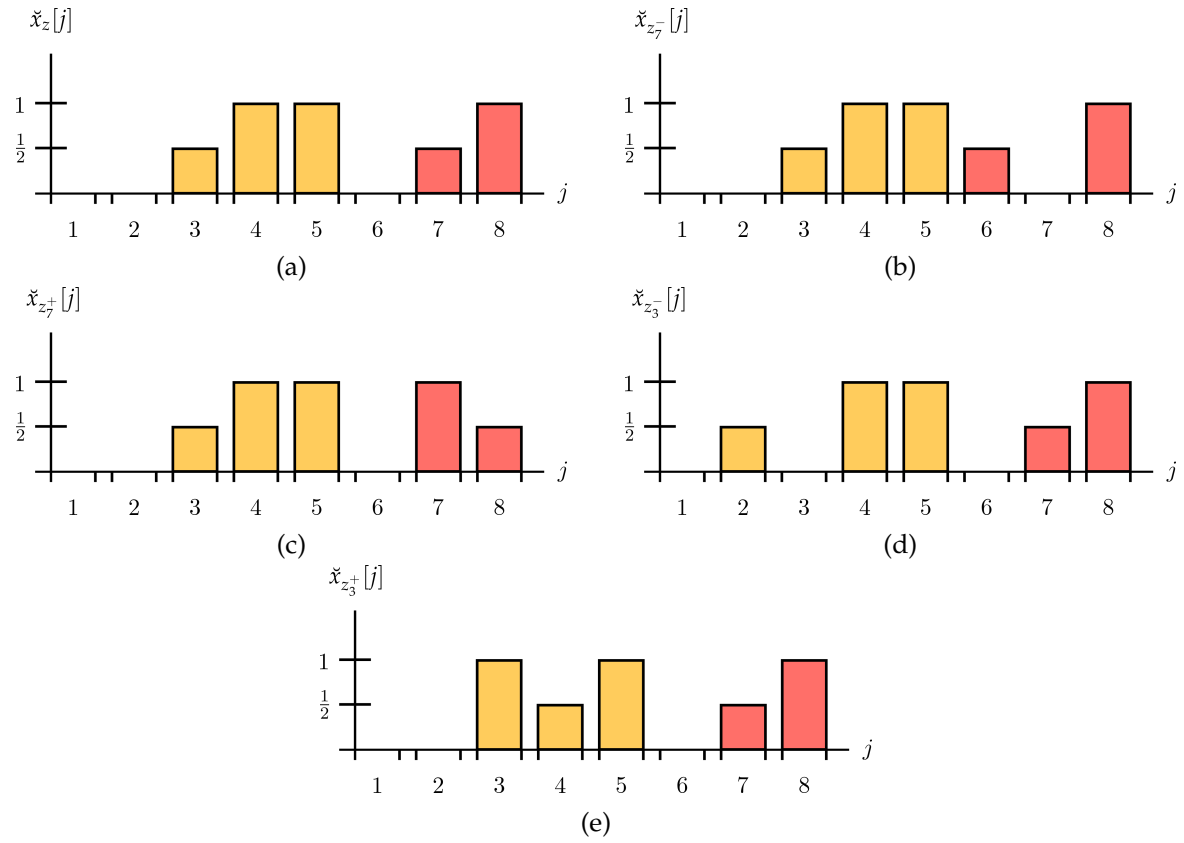



Figure 4.4: Illustrations to accompany the proof of Lemma 4.3.5 when $n = 8$. For each $j$ on the $x$-axis, the histogram gives the value of either $\check{x}_z[j]$ (Figure 4.4a), $\check{x}_{z_7^-}[j]$ (Figure 4.4b), $\check{x}_{z_7^+}[j]$ (Figure 4.4c), $\check{x}_{z_3^-}[j]$ (Figure 4.4d), or $\check{x}_{z_3^+}[j]$ (Figure 4.4e). In this case, $i = 3$.

$$\check{c}_{z_{n-1}^-} = \check{\boldsymbol{c}}_z - \frac{3\gamma}{4}, \text{ and}$$
$$\check{c}_{z_{n-1}^+} = \check{\boldsymbol{c}}_z + \frac{3}{4} - \frac{1}{2}\left(3 - \frac{1}{2\rho^*}\right) = \check{\boldsymbol{c}}_z - \frac{\gamma}{4}\left(3 - \frac{1}{\rho^*}\right).$$

This means that $\check{c}_z - \check{c}_{z_i^-} = \check{c}_z - \check{c}_{z_i^+} = \gamma/2$, $\check{c}_z - \check{c}_{z_{n-1}^-} = 3\gamma/4$, and $\check{c}_z - \check{c}_{z_{n-1}^+} = \frac{\gamma}{4}\left(3 - \frac{1}{\rho^*}\right)$. Therefore, $\rho\text{score}_1(z,i) + (1-\rho)\text{score}_2(z,i) = \gamma/2$ and $\rho\text{score}_1(z,n-1) + (1-\rho)\text{score}_2(z,n-1) = \frac{\rho\gamma}{4}\left(3 - \frac{1}{\rho^*}\right) + \frac{3\gamma(1-\rho)}{4} = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}$. This means that $\rho\text{score}_1(z,i) + (1-\rho)\text{score}_2(z,i) = \gamma/2 < \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*} = \rho\text{score}_1(z,n-1) + (1-\rho)\text{score}_2(z,n-1)$ so long as $\rho < \rho^*$.

The next node B&B will explore is $z_{n-1}^-$. The vector $\check{\boldsymbol{x}}_{z_{n-1}^-}$ has fractional values only in positions $i$ and $n-2$. Branching on $i$, we again have that $\check{c}_{z_{n-1}^-} - \check{c}_{z_{n-1,i}^{-,-}} = \check{c}_{z_{n-1}^-} - \check{c}_{z_{n-1,i}^{-,+}} = \gamma/2$. Branching on $n-2$, $z_{n-1,n-2}^{-,-}$ is infeasible, so $\check{c}_{z_{n-1}^-} - \check{c}_{z_{n-1,n-2}^{-,-}}$ equals some large number $B \geq ||\boldsymbol{c}||_1$. Next, $\check{\boldsymbol{x}}_{z_{n-1,n-2}^{-,+}} = \check{\boldsymbol{x}}_{z_{n-1}^-} + \frac{1}{2}\boldsymbol{e}_{n-2} - \frac{1}{2}\boldsymbol{e}_n$, so $\check{c}_{z_{n-1,n-2}^{-,+}} = \check{c}_{z_{n-1}^-} - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right)$. Therefore, $\rho\text{score}_1\left(z_{n-1}^-, i\right) + (1-\rho)\text{score}_2\left(z_{n-1}^-, i\right) = \gamma/2$ and

$$\rho\text{score}_1\left(z_{n-1}^-, n-2\right) + (1-\rho)\text{score}_2\left(z_{n-1}^-, n-2\right) = \frac{\rho\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) + (1-\rho)B$$

$$
\begin{aligned}
&= B + \rho\left(\frac{3\gamma}{2} - \frac{\gamma}{4\rho^*} - B\right) \\
&\geq B + \rho^*\left(\frac{3\gamma}{2} - \frac{\gamma}{4\rho^*} - B\right) \\
&= B - \frac{\gamma}{4} + \rho^*\left(\frac{3\gamma}{2} - B\right) \\
&> B - \frac{\gamma}{4} + \frac{3\gamma/4 - B}{3\gamma/2 - B}\left(\frac{3\gamma}{2} - B\right) \\
&= B - \frac{\gamma}{4} + \frac{3\gamma}{4} - B \\
&= \frac{\gamma}{2},
\end{aligned}
$$

where the final inequality holds because $\rho^* < 1 < \frac{B-3\gamma/4}{B-3\gamma/2}$. Therefore, $x[n-2]$ will be branched on next.

Since $z^{-,-}_{n-1,n-2}$ is infeasible, the next node B&B will explore is $z^{-,+}_{n-1,n-2}$. The vector $\check{\boldsymbol{x}}_{z^{-,+}_{n-1,n-2}}$ has fractional values only in positions $i$ and $n$. Both MILP instances $z^{-,+,-}_{n-1,n-2,n}$ and $z^{-,+,+}_{n-1,n-2,n}$ are infeasible, so $\rho\text{score}_1\left(z^{-,+}_{n-1,n-2}, n\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n-2}, n\right) = B$ whereas

$$
\rho\text{score}_1\left(z^{-,+}_{n-1,n-2}, i\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n-2}, i\right) = \frac{\gamma}{2},
$$

as before. Therefore, B&B will branch on $x[n]$ and fathom both children.

The next node B&B will explore is $z^{+}_{n-1}$. The vector $\check{\boldsymbol{x}}_{z^{+}_{n-1}}$ has fractional values only in positions $i$ and $n$. Branching on $i$, we again have that $\check{c}_{z^{+}_{n-1}} - \check{c}_{z^{+,-}_{n-1,i}} = \check{c}_{z^{+}_{n-1}} - \check{c}_{z^{+,+}_{n-1,i}} = \gamma/2$. Branching on $x[n]$, $\check{\boldsymbol{x}}_{z^{+,-}_{n-1,n}} = \check{\boldsymbol{x}}_{z^{+}_{n-1}} - \frac{1}{2}\boldsymbol{e}_n + \frac{1}{2}\boldsymbol{e}_{n-2}$, so $\check{c}_{z^{+,-}_{n-1,n}} = \check{c}_{z^{+}_{n-1}} - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right)$. Meanwhile, $z^{+,+}_{n-1,n}$ is infeasible, so $\check{c}_{z^{+}_{n-1}} - \check{c}_{z^{+,+}_{n-1,n-2}} = B$. Therefore, $\rho\text{score}_1\left(z^{+}_{n-1}, i\right) + (1-\rho)\text{score}_2\left(z^{+}_{n-1}, i\right) = \gamma/2$ and $\rho\text{score}_1\left(z^{+}_{n-1}, n\right) + (1-\rho)\text{score}_2\left(z^{+}_{n-1}, n\right) = \frac{\rho\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) + (1-\rho)B > \gamma/2$. Therefore, $x[n]$ will be branched on next.

The next node B&B will explore is $z^{+,-}_{n-1,n}$. The vector $\check{\boldsymbol{x}}_{z^{+,-}_{n-1,n}}$ has fractional values only in positions $i$ and $n-2$. Both MILP instances $z^{+,-,-}_{n-1,n,n-2}$ and $z^{+,-,-}_{n-1,n,n-2}$ are infeasible, so

$$
\rho\text{score}_1\left(z^{-,+}_{n-1,n}, n-2\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n}, n-2\right) = B
$$

whereas $\rho\text{score}_1\left(z^{-,+}_{n-1,n-2}, i\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n-2}, i\right) = \gamma/2$, as before. Therefore, B&B will branch on $x[n-2]$ and fathom both children.

At this point, all children have been fathomed, so B&B will terminate. □

**Lemma 4.3.6.** *Suppose $\rho > \rho^*$. Then for any MILP $z_{\gamma,n} \in \mathcal{F}_{n,\rho^*}$, B&B with the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ will create a tree of depth at least $2^{(n-5)/4}$.*

*Proof of Lemma 4.3.6.* To prove this lemma, we use induction to show that on any path from the root of the B&B tree to a node of depth $i = \lfloor (n-3)/2 \rfloor$, if $J$ are the set of indices

branched on along that path, then $J \subseteq \{x[1], \ldots, x[n-3]\}$. Even after branching on $i$ nodes from $\{x[1], \ldots, x[n-3]\}$, the MILP will still be feasible, so the branch will not yet have been fathomed (since the original MILP is infeasible, a node will be fathomed only when it is infeasible). Therefore, B&B will continue down every branch to depth $\lfloor (n-3)/2 \rfloor$, thus creating a tree with $2^{(n-4)/2}$ nodes.

**Claim 4.3.7.** *On any path from the root of the B&B tree to a node of depth $i = \lfloor (n-3)/2 \rfloor$, if $J$ are the set of indices branched on along that path, then $J \subseteq \{x[1], \ldots, x[n-3]\}$.*

*Proof of Claim 4.3.7.* We prove this claim by induction.

**Inductive hypothesis.** For $j \leq \lfloor (n-3)/2 \rfloor$, let $J$ be the set of indices branched on along an arbitrary path of the B&B tree from the root to a node of depth $j$. Then $J \subseteq \{x[1], \ldots, x[n-3]\}$.

**Base case ($j = 0$).** As we saw in the proof of Lemma 4.3.5, if $\rho > \rho^*$, then B&B will first branch on $x[i]$ where $i = \lceil (n-3)/2 \rceil$.

**Inductive step.** Let $j$ be an arbitrary index such that $0 \leq j \leq i-1$. Let $J$ be the set of indices branched on along an arbitrary path of the B&B tree from the root to a node of depth $j$. We know from the inductive hypothesis that $J \subseteq \{x[1], \ldots, x[n-3]\}$. Let $z'$ be the MILP at that node. Since $j \leq \lfloor (n-3)/2 \rfloor - 1$, we know that the LP relaxation of $z'$ is feasible. Let $z$ be the number of variables set to zero in $J$ and let $x[p_1], x[p_2], \ldots, x[p_t]$ be $\{x[1], \ldots, x[n-3]\} \setminus J$ ordered such that $p_k < p_{k'}$ for $k < k'$. We know that the solution to the LP relaxation of $z'$ will have the first $i' := \lfloor (n-3)/2 \rfloor - z$ variables $x[p_1], \ldots, x[p_{i'}]$ set to 0, it will set $x[p_{i'+1}]$ to $1/2$, and it will set the remaining variables in $\{x[1], \ldots, x[n-3]\} \setminus J$ to 1. Thus, the fractional variables are $x[p_{i'+1}]$ and $x[n-1]$. Note that since $z \leq |J| \leq \lfloor (n-3)/2 \rfloor - 1$, $i' = \lfloor (n-3)/2 \rfloor - z \geq 1$.

Suppose we branch on $x[p_{i'+1}]$. If we set $x[p_{i'+1}] = 0$, then the LP relaxation of $(z')^-_{p_{i'+1}}$ will set $x[p_{i'}]$ to be $1/2$ and otherwise the optimal solution will remain unchanged. Thus, $\check{c}_{z'} - \check{c}_{(z')^-_{p_{i'+1}}} = \check{c}_{z'} - (\check{c}_{z'} - \gamma p_{i'+1}/2 + \gamma p_{i'}/2) = \frac{\gamma(p_{i'+1} - p_{i'})}{2}$. Meanwhile, if we set $x[p_{i'+1}] = 1$, then the LP relaxation of $(z')^-_{p_{i'+1}}$ will set $x[p_{i'+2}]$ to be 0 and otherwise the optimal solution will remain unchanged. Thus, $\check{c}_{z'} - \check{c}_{(z')^+_{p_{i'+1}}} = \check{c}_{z'} - (\check{c}_{z'} + \gamma p_{i'+1}/2 - \gamma p_{i'+2}/2) = \frac{\gamma(p_{i'+2} - p_{i'+1})}{2}$. Suppose that $\check{c}_{z'} - \check{c}_{(z')^+_{p_{i'+1}}} > \check{c}_{z'} - \check{c}_{(z')^-_{p_{i'+1}}}$. Then

$$\begin{aligned}
\rho \text{score}_1\left(z', p_{i'+1}\right) + (1-\rho)\text{score}_2\left(z', p_{i'+1}\right) &= \frac{\gamma}{2}\left(\rho\left(p_{i'+1} - p_{i'}\right) + (1-\rho)\left(p_{i'+2} - p_{i'+1}\right)\right) \\
&\geq \frac{\gamma}{2}\left(\rho + (1-\rho)\right) \\
&= \frac{\gamma}{2}.
\end{aligned}$$

Meanwhile, suppose that $\check{c}_{z'} - \check{c}_{(z')^+_{p_{i'+1}}} \leq \check{c}_{z'} - \check{c}_{(z')^-_{p_{i'+1}}}$. Then

$$\begin{aligned}
\rho \text{score}_1\left(z', p_{i'+1}\right) + (1-\rho)\text{score}_2\left(z', p_{i'+1}\right) &= \frac{\gamma}{2}\left(\rho\left(p_{i'+2} - p_{i'+1}\right) + (1-\rho)\left(p_{i'+1} - p_{i'}\right)\right) \\
&\geq \frac{\gamma}{2}\left(\rho + (1-\rho)\right) \\
&= \frac{\gamma}{2}.
\end{aligned}$$

Meanwhile, as in the proof of Lemma 4.3.5, $\rho\text{score}_1(z', n-1) + (1-\rho)\text{score}_2(z', n-1) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*} < \frac{\gamma}{2}$ so long as $\rho > \rho^*$. Thus, B&B will branch next on $x[p_{i'}]$. □

□

□

**Theorem 4.3.8.** *Let*

$$\text{score}_1(z,i) = \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\} \text{ and } \text{score}_2(z,i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}.$$

*For all even $n \geq 6$ and all $\rho^* \in \left(\frac{1}{3}, \frac{2}{3}\right)$, there exists an infinite family $\mathcal{G}_{n,\rho^*}$ of MILP instances such that for any $z \in \mathcal{G}_{n,\rho^*}$, if $\rho \in [0, \rho^*)$, then the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ results in a B&B tree with $\Omega\left(2^{(n-5)/4}\right)$ nodes and if $\rho \in (\rho^*, 1]$, the scoring rule results a tree with $O(1)$ nodes.*

*Proof.* For any constant $\gamma \geq 1$, let $\boldsymbol{c}_1 \in \mathbb{R}^{n-3}$ be a vector such that

$$c_1[i] = \begin{cases} 0 & \text{if } i < (n-3)/2 \\ 1.5 & \text{if } i = \lceil (n-3)/2 \rceil \\ 3 - \frac{1}{2\rho^*} & \text{if } i > (n-3)/2 + 1 \end{cases}$$

and let $\boldsymbol{c}_2 = (1,2,3)$. Let $\boldsymbol{c} = \gamma\,(\boldsymbol{c}_1, \boldsymbol{c}_2) \in \mathbb{R}^n$ be the concatenation of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ multiplied with $\gamma$. For example, if $\gamma = 1$ and $n = 8$, then $\boldsymbol{c} = \left(0, 0, 1.5, 3 - \frac{1}{2\rho^*}, 3 - \frac{1}{2\rho^*}, 1, 2, 3\right)$ Next, define the $n$-dimensional vectors $\boldsymbol{a}_1 = 2\sum_{i=1}^{n-3} \boldsymbol{e}_i$ and $\boldsymbol{a}_2 = 2\sum_{i=1}^{3} \boldsymbol{e}_{n-3+i}$, and let $A$ be a matrix whose first row is $\boldsymbol{a}_1$ and second row is $\boldsymbol{a}_2$. Let $z_{\gamma,n}$ be the MILP

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c} \cdot \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = (n-3, 3)^\top \\ & \boldsymbol{x} \in \{0,1\}^n. \end{array}$$

We define $\mathcal{G}_{n,\rho^*} = \{z_{n,\gamma} : \gamma \geq 1\}$.

For every even $n \geq 6$, $z_{\gamma,n}$, both of the constraints $\boldsymbol{a}_1 \cdot \boldsymbol{x} = 2\sum_{i=1}^{n-3} x[i] = n-3$ and $\boldsymbol{a}_2 \cdot \boldsymbol{x} = 2\sum_{i=1}^{3} x[n-3+i] = 3$ are infeasible for $\boldsymbol{x} \in \{0,1\}^n$ since $2\sum_{i=1}^{n-3} x[i]$ and $2\sum_{i=1}^{3} x[n-3+i]$ are even numbers but 3 and $n-3$ are odd numbers. The key idea of this proof is that if B&B branches on all variables in $\{x[n-2], x[n-1], x[n]\}$ first, it will terminate upon making a tree of size at most $2^3 = 8$, since at most three branches are necessary to determine that $\boldsymbol{a}_1 \cdot \boldsymbol{x} = 2\sum_{i=1}^{3} x[n-3+i] = 3$ is infeasible. However, if B&B branches on all variables in $\{x[1], \ldots, x[n-3]\}$ first, it will create a tree with exponential size before it terminates.

**Lemma 4.3.9.** *Suppose $\rho > \rho^*$. Then for any MILP $z_{\gamma,n} \in \mathcal{G}_{n,\rho^*}$, $\rho\text{score}_1 + (1-\rho)\text{score}_2$ branches on all variables in $\{x[n-2], x[n-1], x[n]\}$ before branching on variables in $\{x[1], \ldots, x[n-3]\}$.*

*Proof of Lemma 4.3.9.* For ease of notation, for the remainder of this proof, we drop the subscript $(\gamma, n)$ from $z_{\gamma,n}$ and denote this MILP as $z$. The optimal solution to the LP relaxation of $z$ has the following form:

$$\check{\boldsymbol{x}}_z[j] = \begin{cases} 0 & \text{if } j \leq \lfloor (n-3)/2 \rfloor \text{ or } j = n-2 \\ \frac{1}{2} & \text{if } j = \lceil (n-3)/2 \rceil \text{ or } j = n-1 \\ 1 & \text{if } \lceil (n-3)/2 \rceil \leq j \leq n-3 \text{ or } j = n. \end{cases}$$

For example, if $n = 8$, then $\check{\boldsymbol{x}}_z = \left(0, 0, \frac{1}{2}, 1, 1, 0, \frac{1}{2}, 1\right)$. Therefore, the only candidate variables to branch on are $x[n-1]$ and $x[i]$ where $i = \lceil (n-3)/2 \rceil$. Branching on $x[i]$, we have $\check{\boldsymbol{x}}_{z_i^-} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_i + \frac{1}{2}\boldsymbol{e}_{i-1}$ and $\check{\boldsymbol{x}}_{z_i^+} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_i - \frac{1}{2}\boldsymbol{e}_{i+1}$. Branching on $x[n-1]$, we have $\check{\boldsymbol{x}}_{z_{n-1}^-} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{z_{n-1}^+} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{z_i^-} &= \check{c}_z - \frac{3\gamma}{4}, \\
\check{c}_{z_i^+} &= \check{c}_z + \frac{3\gamma}{4} - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) = \check{c}_z - \frac{3\gamma}{4}\left(1 - \frac{1}{\rho^*}\right), \\
\check{c}_{z_{n-1}^-} &= \check{c}_z - \gamma + \frac{\gamma}{2} = \check{c}_z - \frac{\gamma}{2}, \text{ and} \\
\check{c}_{z_{n-1}^+} &= \check{c}_z + \gamma - \frac{3\gamma}{2} = \check{c}_z - \frac{\gamma}{2}.
\end{aligned}$$

This means that $\check{c}_z - \check{c}_{z_i^-} = 3\gamma/4$, $\check{c}_z - \check{c}_{z_i^+} = \frac{\gamma}{4}\left(3 - \frac{1}{\rho^*}\right)$, and $\check{c}_z - \check{c}_{z_{n-1}^-} = \check{c}_z - \check{c}_{z_{n-1}^+} = \gamma/2$. Therefore, $\rho\text{score}_1(z,i) + (1-\rho)\text{score}_2(z,i) = \frac{\rho\gamma}{4}\left(3 - \frac{1}{\rho^*}\right) + \frac{3\gamma(1-\rho)}{4} = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}$ and $\rho\text{score}_1(z, n-1) + (1-\rho)\text{score}_2(z, n-1) = \gamma/2$. This means that $\rho\text{score}_1(z,i) + (1-\rho)\text{score}_2(z,i) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*} < \gamma/2 = \rho\text{score}_1(z, n-1) + (1-\rho)\text{score}_2(z, n-1)$ so long as $\rho > \rho^*$.

The next node B&B will explore is $z_{n-1}^-$. The vector $\check{\boldsymbol{x}}_{z_{n-1}^-}$ has fractional values only in positions $i$ and $n-2$. Branching on $i$, we again have that $\check{c}_{z_{n-1}^-} - \check{c}_{z_{n-1,i}^{-,-}} = 3\gamma/4$ and $\check{c}_{z_{n-1}^-} - \check{c}_{z_{n-1,i}^{-,+}} = \frac{\gamma}{4}\left(3 - \frac{1}{\rho^*}\right)$. Branching on $n-2$, $z_{n-1,n-2}^{-,-}$ is infeasible, so $\check{c}_{z_{n-1}^-} - \check{c}_{z_{n-1,n-2}^{-,-}}$ equals some large number $B \geq ||\boldsymbol{c}||_1$. Next, $\check{\boldsymbol{x}}_{z_{n-1,n-2}^{-,+}} = \check{\boldsymbol{x}}_{z_{n-1}^-} + \frac{1}{2}\boldsymbol{e}_{n-2} - \frac{1}{2}\boldsymbol{e}_n$, so $\check{c}_{z_{n-1,n-2}^{-,+}} = \check{c}_{z_{n-1}^-} - \gamma$. Therefore, $\rho\text{score}_1\left(z_{n-1}^-, i\right) + (1-\rho)\text{score}_2\left(z_{n-1}^-, i\right) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}$ and

$$\begin{aligned}
\rho\text{score}_1\left(z_{n-1}^-, n-2\right) + (1-\rho)\text{score}_2\left(z_{n-1}^-, n-2\right) &= \rho\gamma + (1-\rho)B \\
&= B + \rho\left(\gamma - B\right) \\
&= B - \frac{\rho\gamma}{4\rho^*} + \rho\left(\gamma + \frac{\gamma}{4\rho^*} - B\right) \\
&> B - \frac{\rho\gamma}{4\rho^*} + \frac{3\gamma/4 - B}{\gamma + \frac{\gamma}{4\rho^*} - B}\left(\gamma + \frac{\gamma}{4\rho^*} - B\right) \\
&= B - \frac{\rho\gamma}{4\rho^*} + 3\gamma/4 - B \\
&= \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*},
\end{aligned}$$

where the final inequality holds because $\rho < 1 < \frac{B - 3\gamma/4}{B - (\gamma + \gamma/(4\rho^*))}$. Therefore, $x[n-2]$ will be branched on next.

Since $z_{n-1,n-2}^{-,-}$ is infeasible, the next node B&B will explore is $z_{n-1,n-2}^{-,+}$. The vector $\check{\boldsymbol{x}}_{z_{n-1,n-2}^{-,+}}$ has fractional values only in positions $i$ and $n$. Since both MILP instances $z_{n-1,n-2,n}^{-,+,-}$ and $z_{n-1,n-2,n}^{-,+,+}$ are infeasible, so $\rho\text{score}_1\left(z_{n-1,n-2}^{-,+}, n\right) + (1-\rho)\text{score}_2\left(z_{n-1,n-2}^{-,+}, n\right) = B$ whereas

$\rho\text{score}_1\left(z^{-,+}_{n-1,n-2}, i\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n-2}, i\right) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*} < B$. Therefore, B&B will branch on $x[n]$ and fathom both children.

The next node B&B will explore is $z^+_{n-1}$. The vector $\check{\boldsymbol{x}}_{z^+_{n-1}}$ has fractional values only in positions $i$ and $n$. Branching on $i$, we again have that $\check{c}_{z^-_{n-1}} - \check{c}_{z^{-,-}_{n-1,i}} = 3\gamma/4$ and $\check{c}_{z^-_{n-1}} - \check{c}_{z^{-,+}_{n-1,i}} = \frac{\gamma}{4}\left(3 - \frac{1}{\rho^*}\right)$. Branching on $x[n]$, $\check{\boldsymbol{x}}_{z^{+,-}_{n-1,n}} = \check{\boldsymbol{x}}_{z^+_{n-1}} - \frac{1}{2}\boldsymbol{e}_n + \frac{1}{2}\boldsymbol{e}_{n-2}$, so $\check{c}_{z^{+,-}_{n-1,n}} = \check{c}_{z^+_{n-1}} - \gamma$. Meanwhile, $z^{+,+}_{n-1,n}$ is infeasible, so $\check{c}_{z^+_{n-1}} - \check{c}_{z^{+,+}_{n-1,n-2}}$ equals some large number $B \geq ||\boldsymbol{c}||_1$. Therefore, $\rho\text{score}_1\left(z^+_{n-1}, i\right) + (1-\rho)\text{score}_2\left(z^+_{n-1}, i\right) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}$ and

$$\rho\text{score}_1\left(z^+_{n-1}, n\right) + (1-\rho)\text{score}_2\left(z^+_{n-1}, n\right) = \rho\gamma + (1-\rho)B > \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*}.$$

Therefore, $x[n]$ will be branched on next.

The next node B&B will explore is $z^{+,-}_{n-1,n}$. The vector $\check{\boldsymbol{x}}_{z^{+,-}_{n-1,n}}$ has fractional values only in positions $i$ and $n-2$. Since both MILP instances $z^{+,-,-}_{n-1,n,n-2}$ and $z^{+,-,-}_{n-1,n,n-2}$ are infeasible, so $\rho\text{score}_1\left(z^{-,+}_{n-1,n}, n-2\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n}, n-2\right) = B$ whereas $\rho\text{score}_1\left(z^{-,+}_{n-1,n-2}, i\right) + (1-\rho)\text{score}_2\left(z^{-,+}_{n-1,n-2}, i\right) = \frac{3\gamma}{4} - \frac{\rho\gamma}{4\rho^*} < B$, as before. Therefore, B&B will branch on $x[n-2]$ and fathom both children.

At this point, all children have been fathomed, so B&B will terminate. □

**Lemma 4.3.10.** *Suppose $\rho < \rho^*$. Then for any MILP $z_{\gamma,n} \in \mathcal{G}_{n,\rho^*}$, B&B with the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ will create a tree of depth at least $2^{(n-5)/4}$.*

*Proof.* Let $i = \lceil (n-3)/2 \rceil$. We first prove two useful claims.

**Claim 4.3.11.** *Let $j$ be an even number such that $2 \leq j \leq i-2$ and let $J = \{x[i-j/2], \ldots, x[i+j/2-1]\}$. Suppose that B&B has branched on exactly the variables in $J$ and suppose that the number of variables set to 1 equals the number of variables set to 0. Then B&B will next branch on the variable $x[i+j/2]$. Similarly, suppose $J = \{x[i-j/2+1], x[i-j/2+2], \ldots, x[i+j/2]\}$. Suppose that B&B has branched on exactly the variables in $J$ and suppose that the number of variables set to 1 equals the number of variables set to 0. Then B&B will next branch on the variable $x[i-j/2]$.*

*Proof.* Let $z$ be the MILP contained in the node at the end of the path. This proof has two cases.

**Case 1:** $J = \{x[i-j/2], x[i-j/2+1], \ldots, x[i+j/2-1]\}$. In this case, there is a set

$$J_< = \{x[1], \ldots, x[i-j/2-1]\}$$

of $i - \frac{j}{2} - 1$ variables smaller than $x[i]$ that have not yet been branched on and there is a set $J_> = \{x[i+j/2], \ldots, x[n-3]\}$ of $n - 3 - \left(i + \frac{j}{2} - 1\right) = 2i - 1 - \left(i + \frac{j}{2} - 1\right) = i - \frac{j}{2}$ variables in $\{x[i+1], \ldots, x[n-3]\}$ that have not yet been branched on. Since the number of variables set to 1 in $J$ equals the number of variables set to 0, the LP relaxation will set the $i - \frac{j}{2} - 1$ variables in $J_<$ to 0, the $i - \frac{j}{2} - 1$ variables in $J_> \setminus \{x[i+j/2]\}$ to 1, and $x[i+j/2]$ to $\frac{1}{2}$. It will also set $x[n-2] = 0$, $x[n-1] = \frac{1}{2}$, and $x[n] = 1$. Therefore, the two fractional variables are $x[i+j/2]$ and $x[n-1]$. Branching on $x[i+j/2]$, we have $\check{\boldsymbol{x}}_{z^-_{i+j/2}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{i+j/2} + \frac{1}{2}\boldsymbol{e}_{i-j/2-1}$ and

$\check{\boldsymbol{x}}_{z^+_{i+j/2}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{i+j/2} - \frac{1}{2}\boldsymbol{e}_{i+j/2+1}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{z^-_{n-1}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{z^+_{n-1}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{z^-_{i+j/2}} &= \check{c}_z - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
\check{c}_{z^+_{i+j/2}} &= \check{c}_z \\
\check{c}_{z^-_{n-1}} &= \check{c}_z - 1 + \frac{1}{2} = \check{c}_z - \frac{\gamma}{2} \\
\check{c}_{z^+_{n-1}} &= \check{c}_z + 1 - \frac{3}{2} = \check{c}_z - \frac{\gamma}{2}
\end{aligned}$$

This means that $\check{c}_z - \check{c}_{z^-_{i+j/2}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right)$, $\check{c}_z - \check{c}_{z^+_{i+j/2}} = 0$, and $\check{c}_z - \check{c}_{z^-_{n-1}} = \check{c}_z - \check{c}_{z^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\rho\text{score}_1(z, i+j/2) + (1-\rho)\text{score}_2(z, i+j/2) = \frac{\gamma(1-\rho)}{2}\left(3 - \frac{1}{2\rho^*}\right)$ and $\rho\text{score}_1(z, n-1) + (1-\rho)\text{score}_2(z, n-1) = \frac{\gamma}{2}$. Since $\rho < \rho^*$ and $\rho^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, we have that

$$\begin{aligned}
\rho\text{score}_1(z, i+j/2) + (1-\rho)\text{score}_2(z, i+j/2) &= \frac{\gamma(1-\rho)}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
&\geq \frac{\gamma\left(1-\rho^*\right)}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
&\geq \frac{\gamma}{2}.
\end{aligned}$$

Therefore, $x[i+j/2]$ will be branched on next.

**Case 2:** $J = \{x[i-j/2+1], x[i-j/2+2], \ldots, x[i+j/2]\}$. In this case, there is a set

$$J_< = \{x[1], \ldots, x[i-j/2]\}$$

of $i - \frac{j}{2}$ variables smaller than $x[i]$ that have not yet been branched on and a set $J_> = \{x[i+j/2+1], \ldots, x[n-3]\}$ of $n - 3 - \left(i + \frac{j}{2}\right) = 2i - 1 - \left(i + \frac{j}{2}\right) = i - \frac{j}{2} - 1$ variables in $\{x[i+1], \ldots, x[n-3]\}$ that have not yet been branched on. Since the number of variables set to 1 in $J$ equals the number of variables set to 0, the LP relaxation will set the $i - \frac{j}{2} - 1$ variables in $J_< \setminus \{x[i-j/2]\}$ to 0, the $i - \frac{j}{2} - 1$ variables in $J_> \setminus \{x[i+j/2]\}$ to 1, and $x[i-j/2]$ to $\frac{1}{2}$. It will also set $x[n-2] = 0$, $x[n-1] = \frac{1}{2}$, and $x[n] = 1$. Therefore, the two fractional variables are $x[i-j/2]$ and $x[n-1]$. Branching on $x[i-j/2]$, we have $\check{\boldsymbol{x}}_{z^-_{i-j/2}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{i-j/2} + \frac{1}{2}\boldsymbol{e}_{i-j/2-1}$ and $\check{\boldsymbol{x}}_{z^+_{i-j/2}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{i-j/2} - \frac{1}{2}\boldsymbol{e}_{i+j/2+1}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{z^-_{n-1}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{z^+_{n-1}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{z^-_{i-j/2}} &= \check{c}_z \\
\check{c}_{z^+_{i-j/2}} &= \check{c}_z - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
\check{c}_{z^-_{n-1}} &= \check{c}_z - \gamma + \frac{\gamma}{2} = \check{c}_z - \frac{\gamma}{2}
\end{aligned}$$

$$\check{c}_{z^+_{n-1}} = \check{c}_z + \gamma - \frac{3\gamma}{2} = \check{c}_z - \frac{\gamma}{2}$$

This means that $\check{c}_z - \check{c}_{z^-_{i-j/2}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right)$, $\check{c}_z - \check{c}_{z^+_{i-j/2}} = 0$, and $\check{c}_z - \check{c}_{z^-_{n-1}} = \check{c}_z - \check{c}_{z^+_{n-1}} = \frac{\gamma}{2}$ as in the previous case, so $x[i - j/2]$ will be branched on next. □

**Claim 4.3.12.** *Suppose that $J$ is the set of variables branched on along a path of depth $2 \leq j \leq i-2$ where $j$ is odd and let $j' = \lfloor j/2 \rfloor$. Suppose that $J = \{x[i-j'], x[i-j'+1], \ldots, x[i+j']\}$. Moreover, suppose that the number of variables set to 1 in $J$ equals the number of variables set to 0, plus or minus 1. Then B&B will either branch on $x[i-j'-1]$ or $x[i+j'+1]$.*

*Proof.* Let $z$ be the MILP contained a the end of the path. There are $i - j' - 1$ variables $J_< = \{x[1], \ldots, x[i-j'-1]\}$ that are smaller than $x[i]$ that have not yet been branched on, and $n - 3 - (i+j') = 2i - 1 - (i+j') = i - j' - 1$ variables

$$J_> = \{x[i+j'+1], \ldots, x[n-3]\}$$

in $\{x[i+1], \ldots, x[n-3]\}$ that have not yet been branched on. Let $z$ be the number of variables in $J$ set to 0 and let $o$ be the number of variables set to 1. This proof has two cases:

**Case 1:** $z = o + 1$. Since $z + o = j$, we know that $z = j' + 1$ and $o = j'$. Therefore, the LP relaxation will set the variables in $J_< \setminus \{x[i-j'-1]\}$ to zero for a total of $|J_< \setminus \{x[i-j'-1]\}| + z = i - j' - 2 + j' + 1 = i - 1$ zeros, it will set the variables in $J_>$ to one for a total of $|J_>| + o = i - j' - 1 + j' = i - 1$ ones, and it will set $x[i-j'-1]$ to $\frac{1}{2}$. It will also set $x[n-2] = 0$, $x[n-1] = \frac{1}{2}$, and $x[n] = 1$. Therefore, the two fractional variables are $x[i-j'-1]$ and $x[n-1]$. Branching on $x[i-j'-1]$, we have $\check{\boldsymbol{x}}_{z^-_{i-j'-1}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{i-j'-1} + \frac{1}{2}\boldsymbol{e}_{i-j'-2}$ and $\check{\boldsymbol{x}}_{z^+_{i-j'-1}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{i-j'-1} - \frac{1}{2}\boldsymbol{e}_{i+j'+1}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{z^-_{n-1}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{z^+_{n-1}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{z^-_{i-j'-1}} &= \check{c}_z \\
\check{c}_{z^+_{i-j'-1}} &= \check{\boldsymbol{x}}_z - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
\check{c}_{z^-_{n-1}} &= \check{c}_z - \gamma + \frac{\gamma}{2} = \check{c}_z - \frac{\gamma}{2} \\
\check{c}_{z^+_{n-1}} &= \check{c}_z + \gamma - \frac{3\gamma}{2} = \check{c}_z - \frac{\gamma}{2}
\end{aligned}$$

This means that $\check{c}_z - \check{c}_{z^-_{i-j'-1}} = 0$, $\check{c}_z - \check{c}_{z^+_{i-j'-1}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right)$, and $\check{c}_z - \check{c}_{z^-_{n-1}} = \check{c}_z - \check{c}_{z^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\rho\text{score}_1(z, i-j'-1) + (1-\rho)\text{score}_2(z, i-j'-1) = \frac{\gamma(1-\rho)}{2}\left(3 - \frac{1}{2\rho^*}\right)$ and $\rho\text{score}_1(z, n-1) + (1-\rho)\text{score}_2(z, n-1) = \frac{\gamma}{2}$. Since $\rho < \rho^*$ and $\rho^* \in \left(\frac{1}{3}, \frac{1}{2}\right)$, we have that

$$\begin{aligned}
\rho\text{score}_1(z, i-j'-1) + (1-\rho)\text{score}_2(z, i-j'-1) &= \frac{\gamma(1-\rho)}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
&\geq \frac{\gamma\,(1-\rho^*)}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
&\geq \frac{\gamma}{2}.
\end{aligned}$$

Therefore, $x[i-j'-1]$ will be branched on next.

**Case 1:** $z = o - 1$. Since $z + o = j$, we know that $z = j'$ and $o = j' + 1$. Therefore, the LP relaxation will set the variables in $J_<$ to zero for a total of $|J_<| + z = i - j' - 1 + j' = i - 1$ zeros, it will set the variables in $J_> \setminus \{x[i + j' + 1]\}$ to one for a total of $|J_> \setminus \{x[i + j' + 1]\}| + o = i - j' - 2 + j' + 1 = i - 1$ ones, and it will set $x[i + j' + 1]$ to $\frac{1}{2}$. It will also set $x[n-2] = 0$, $x[n-1] = \frac{1}{2}$, and $x[n] = 1$. Therefore, the two fractional variables are $x[i + j' + 1]$ and $x[n-1]$. Branching on $x[i + j' + 1]$, we have $\check{\boldsymbol{x}}_{z^-_{i+j'+1}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{i+j'+1} + \frac{1}{2}\boldsymbol{e}_{i-j'-1}$ and $\check{\boldsymbol{x}}_{z^+_{i+j'+1}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{i+j'+1} - \frac{1}{2}\boldsymbol{e}_{i+j'+2}$. Branching on $x[n-1]$, we have that $\check{\boldsymbol{x}}_{z^-_{n-1}} = \check{\boldsymbol{x}}_z - \frac{1}{2}\boldsymbol{e}_{n-1} + \frac{1}{2}\boldsymbol{e}_{n-2}$ and $\check{\boldsymbol{x}}_{z^+_{n-1}} = \check{\boldsymbol{x}}_z + \frac{1}{2}\boldsymbol{e}_{n-1} - \frac{1}{2}\boldsymbol{e}_n$. Therefore,

$$\begin{aligned}
\check{c}_{z^-_{i+j'+1}} &= \check{c}_z - \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right) \\
\check{c}_{z^+_{i+j'+1}} &= \check{\boldsymbol{x}}_z \\
\check{c}_{z^-_{n-1}} &= \check{c}_z - \gamma + \frac{\gamma}{2} = \check{c}_z - \frac{\gamma}{2} \\
\check{c}_{z^+_{n-1}} &= \check{c}_z + \gamma - \frac{3\gamma}{2} = \check{c}_z - \frac{\gamma}{2}
\end{aligned}$$

This means that $\check{c}_z - \check{c}_{z^-_{i+j'+1}} = \frac{\gamma}{2}\left(3 - \frac{1}{2\rho^*}\right)$, $\check{c}_z - \check{c}_{z^+_{i+j'+1}} = 0$, and $\check{c}_z - \check{c}_{z^-_{n-1}} = \check{c}_z - \check{c}_{z^+_{n-1}} = \frac{\gamma}{2}$. Therefore, $\rho\text{score}_1(z, i + j' + 1) + (1-\rho)\text{score}_2(z, i + j' + 1) = \frac{\gamma(1-\rho)}{2}\left(3 - \frac{1}{2\rho^*}\right)$ and $\rho\text{score}_1(z, n-1) + (1-\rho)\text{score}_2(z, n-1) = \frac{\gamma}{2}$. As in the previous case, this means that $x[i + j' + 1]$ will be branched on next. □

We now prove by induction that there are $2^{(\lceil (n-3)/2 \rceil)-1)/2} \geq 2^{(n-5)/4}$ paths in the B&B tree of length at least $i - 2$. Therefore, the size of the tree is at least $2^{(n-5)/4}$.

**Inductive hypothesis.** Let $j$ be an arbitrary integer between 1 and $i - 2$. If $j$ is even, then there exist at least $2^{j/2}$ paths in the B&B tree from the root to nodes of depth $j$ such that if $J$ is the set indices branched on along a given path, then $J = \{x[i - j/2], x[i - j/2 + 1], \ldots, x[i + j/2 - 1]\}$ or $J = \{x[i - j/2 + 1], x[i - j/2 + 2], \ldots, x[i + j/2]\}$. Moreover, the number of variables set to 0 in $J$ equals the number of variables set to 1. Meanwhile, if $j$ is odd, let $j' = \lfloor j/2 \rfloor$. There exist at least $2^{(j+1)/2}$ paths in the B&B tree from the root to nodes of depth $j$ such that if $J$ is the set indices branched on along a given path, then $J = \{x[i - j'], x[i - j' + 1], \ldots, x[i + j']\}$. Moreover, the number of variables set to 0 in $J$ equals the number of variables set to 1, plus or minus 1.

**Base case.** To prove the base case, we need to show that B&B first branches on $x[i]$. We saw that this will be the case in Lemma 4.3.9 so long as $\rho < \rho^*$.

**Inductive step.** Let $j$ be an arbitrary integer between 1 and $i - 3$. There are two cases, one where $j$ is even and one where $j$ is odd. First, suppose $j$ is even. From the inductive hypothesis, we know that there exist at least $2^{j/2}$ paths in the B&B tree from the root to nodes of depth $j$ such that if $J$ is the set varibles branched on along a given path, then

$$J = \{x[i - j/2], x[i - j/2 + 1], \ldots, x[i + j/2 - 1]\}$$

or $J = \{x[i - j/2 + 1], x[i - j/2 + 2], \ldots, x[i + j/2]\}$. Moreover, the number of variables set to 0 in $J$ equals the number of variables set to 1. From Claim 4.3.11, we know that in the first case, $x[i + j/2]$ will be the next node B&B will branch on. This will create two new paths:

$$\{x[i - j/2], x[i - j/2 + 1], \ldots, x[i + j/2]\}$$

will be the set of variables branched along each path, and the number of variables set to 0 will equal the number of variables set to 1, plus or minus 1. Also from Claim 4.3.11, we know that in the second case, $x[i - j/2]$ will be the next node B&B will branch on. This will also create two new paths: $\{x[i - j/2], x[i - j/2 + 1], \ldots, x[i + j/2]\}$ will be the set of variables branched along each path, and the number of variables set to 0 will equal the number of variables set to 1, plus or minus 1. Since this is true for all $2^{j/2}$ paths, this leads to a total of $2^{j/2+1} = 2^{(j+2)/2}$ paths, meaning the inductive hypothesis holds.

Next, suppose $j$ is odd and let $j' = \lfloor j/2 \rfloor$. From the inductive hypothesis, we know that there exist at least $2^{(j+1)/2}$ paths in the B&B tree from the root to nodes of depth $j$ such that if $J$ is the set variables branched on along a given path, then $J = \{x[i - j'], x[i - j' + 1], \ldots, x[i + j']\}$. Moreover, the number of variables set to 0 in $J$ equals the number of variables set to 1, plus or minus 1. From Claim 4.3.11, we know that B&B will either branch on $x[i - j' - 1]$ or $x[i + j' + 1]$. Suppose the number of variables set to 0 in $J$ is 1 greater than the number of variables set to 1. If B&B branches on $x[i - j' - 1]$, we can follow the path where $x[i - j' - 1] = 1$, and this will give us a new path where

$$\begin{aligned} &\{x[i - j' - 1], x[i - j'], \ldots, x[i + j']\} \\ &= \{x[i - (j+1)/2], x[i - (j+1)/2 + 1], \ldots, x[i + (j+1)/2 - 1]\} \end{aligned}$$

are the variables branched on and the number of variables set to 0 equals the number of variables set to 1. If B&B branches on $x[i + j' + 1]$, we can follow the path where $x[i + j' + 1] = 1$, and this will give us a new path where

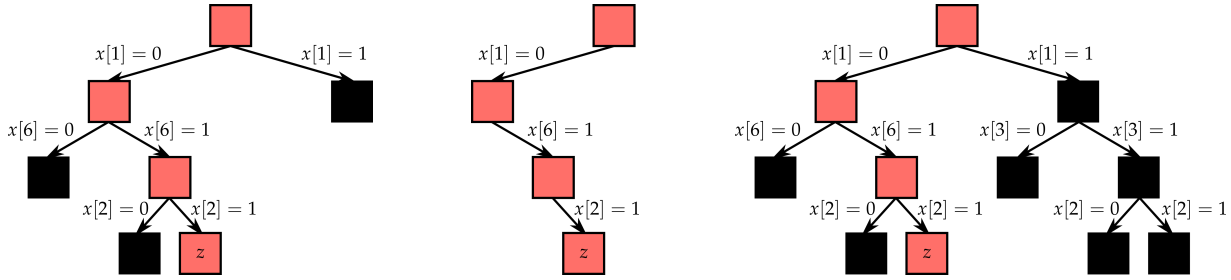
$$\begin{aligned} &\{x[i - j'], x[i - j' + 1], \ldots, x[i + j'], x[i + j' + 1]\} \\ &= \{x[i - (j+1)/2 + 1], x[i - (j+1)/2 + 1], \ldots, x[i + (j+1)/2]\} \end{aligned}$$

are the variables branched on and the number of variables set to 0 equals the number of variables set to 1. A symmetric argument holds if the number of variables set to 0 in $J$ is 1 less than the number of variables set to 1. Therefore, all $2^{(j+1)/2}$ paths can be extended by one edge, so the statement holds. □

□

### 4.3.3 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.




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

(b) The path $\mathcal{T}_z$ from the root of the tree $\mathcal{T}$ in Figure 4.5a to the node labeled $z$.

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

Figure 4.5: Illustrations to accompany the definition of a path-wise scoring rule (Definition 4.3.13). If the scoring rule score is path-wise, then for any variable $x_i$, $\text{score}(\mathcal{T}, z, i) = \text{score}(\mathcal{T}', z, i) = \text{score}(\mathcal{T}_z, z, i)$.

**Path-wise scoring rules**

In this section, we provide guarantees for MILPs where the non-continuous variables are constrained to be binary. Later, we provide guarantees for more general MILPs with arbitrary integer constraints. 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 4.5b illustrates the path $\mathcal{T}_z$ from the root of the tree $\mathcal{T}$ in Figure 4.5a to the node labeled $z$. We now state the definition of path-wise scoring rules.

**Definition 4.3.13** (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 $x_i$,

$$\text{score}(\mathcal{T}, z, i) = \text{score}(\mathcal{T}_z, z, i) \tag{4.4}$$

where $\mathcal{T}_z$ is the path from the root of $\mathcal{T}$ to $z$.[1] See Figure 4.5 for an illustration.

Definition 4.3.13 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 4.2. 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 4.2 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* [Bénichou et al., 1971, Gauthier and Ribière, 1977, Linderoth and Savelsbergh, 1999] and *reliability branching* [Achterberg et al., 2005], two

[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 (4.4) always holds.

widely-used branching strategies, are not path-wise, but our more general results later in this section do apply to those strategies.

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 4.3.14.** *Let* $\text{score}_1$ *and* $\text{score}_2$ *be two path-wise scoring rules, and let* $z$ *be an arbitrary MILP 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.*

*Proof.* We prove this lemma first by considering the actions of an alternative algorithm $A'$ which runs exactly like B&B, except it only fathoms nodes if they are feasible solutions to the original MILP or if they are infeasible. We then relate the behavior of $A'$ to the behavior of B&B to prove the lemma.

First, we prove the following bound on the number of search trees $A'$ will build on a given instance over the entire range of parameters. This bound matches that in the lemma statement.

**Claim 4.3.15.** *There are* $T \leq 2^{n(n-1)/2}n^n$ *intervals* $I_1, \ldots, I_T$ *partitioning* $[0,1]$ *where for any interval* $I_j$*, the search tree* $A'$ *builds using the scoring rule* $\rho\text{score}_1 + (1-\rho)\text{score}_2$ *is invariant across all* $\rho \in I_j$*.*[2]

*Proof of Claim 4.3.15.* We prove this claim by induction.

**Inductive hypothesis.** For $i \in \{1, \ldots, n\}$, there are $T \leq 2^{i(i-1)/2}n^i$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval $I_j$ and any two parameters $\rho, \rho' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees $A'$ builds using the scoring rules $\rho\text{score}_1 + (1-\rho)\text{score}_2$ and $\rho'\text{score}_1 + (1-\rho')\text{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$.

**Base case.** Before branching on any variables, the B&B tree $\mathcal{T}_0$ consists of a single root node $z$. Given a parameter $\rho$, $A'$ will branch on variable $x[k]$ so long as

$$k = \text{argmax}_{\ell \in [n]} \left\{\rho\text{score}_1(\mathcal{T}_0, z, \ell) + (1-\rho)\text{score}_2(\mathcal{T}_0, z, \ell)\right\}.$$

Since $\rho\text{score}_1(\mathcal{T}_0, z, \ell) + (1-\rho)\text{score}_2(\mathcal{T}_0, z, \ell)$ is a linear function of $\rho$ for each $\ell \in [n]$, we know that for any $k \in [n]$, there is at most one interval $I$ of the parameter space $[0,1]$ where $k = \text{argmax}_{\ell \in [n]} \{\rho\text{score}_1 + (1-\rho)\text{score}_2\}$. Thus, there are $T \leq n = 2^{1\cdot(1-1)/2}n^1$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval $I_j$, $A'$ branches on the same variable at the root node using the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ across all $\rho \in I_j$.

**Inductive step.** Let $i \in \{2, \ldots, n\}$ be arbitrary. From the inductive hypothesis, we know that there are $T \leq 2^{(i-2)(i-1)/2}n^{i-1}$ intervals $I_1, \ldots, I_T$ partitioning $[0,1]$ where for any interval $I_j$ and any two parameters $\rho, \rho' \in I_j$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees $A'$ builds using the scoring rules $\rho\text{score}_1 + (1-\rho)\text{score}_2$ and $\rho'\text{score}_1 + (1-\rho')\text{score}_2$, respectively, then $\mathcal{T}[i-1] = \mathcal{T}'[i-1]$. Consider an arbitrary node $z$ in $\mathcal{T}[i-1]$ (or equivalently, $\mathcal{T}'[i-1]$) at depth $i-1$. If

[2]This claim holds even when $\text{score}_1$ and $\text{score}_2$ are members of the more general class of *depth-wise* scoring rules, which we define as follows. For any search tree $\mathcal{T}$ of depth $\text{depth}(\mathcal{T})$ and any $j \in [n]$, let $\mathcal{T}[j]$ be the subtree of $\mathcal{T}$ consisting of all nodes in $\mathcal{T}$ of depth at most $j$. We say that score is a *depth-wise* scoring rule if for all search trees $\mathcal{T}$, all $j \in [\text{depth}(\mathcal{T})]$, all nodes $z$ of depth $j$, and all variables $x[i]$, $\text{score}(\mathcal{T}, z, i) = \text{score}(\mathcal{T}[j], z, i)$.

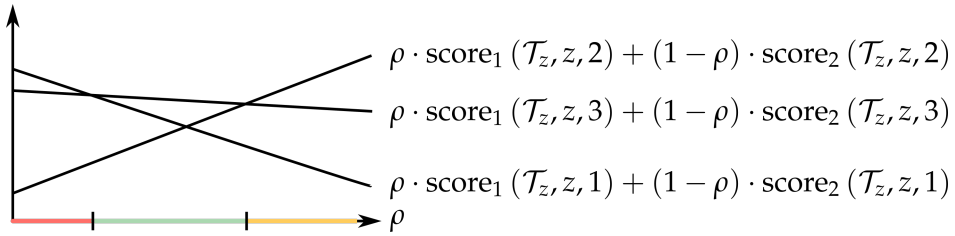



Figure 4.6: Illustrations of the proof of Claim 4.3.15 for a hypothetical MILP $z$ where the algorithm can either branch on $x[1]$, $x[2]$, or $x[3]$ next. In the left-most interval (colored pink), $x[1]$ will be branched on next, in the central interval (colored green), $x[3]$ will be branched on next, and in the right-most interval (colored orange), $x[2]$ next will be branched on next.

$z$ is integral or infeasible, then it will be fathomed no matter which parameter $\rho \in I_j$ the algorithm $A'$ uses. Otherwise, for all $\rho \in I_j$, let $\mathcal{T}_\rho$ be the state of the search tree $A'$ builds using the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$ at the point when it branches on $z$. By the inductive hypothesis, we know that across all $\rho \in I_j$, the path from the root to $z$ in $\mathcal{T}_\rho$ is invariant, and we refer to this path as $\mathcal{T}_z$. Given a parameter $\rho \in I_j$, the variable $x[k]$ will be branched on at node $z$ so long as $k = \text{argmax}_\ell \left\{\rho\text{score}_1(\mathcal{T}_\rho, z, \ell) + (1-\rho)\text{score}_2(\mathcal{T}_\rho, z, \ell)\right\}$, or equivalently, so long as $k = \text{argmax}_\ell \{\rho\text{score}_1(\mathcal{T}_z, z, \ell) + (1-\rho)\text{score}_2(\mathcal{T}_z, z, \ell)\}$. In other words, the decision of which variable to branch on is determined by a convex combination of the constant values $\text{score}_1(\mathcal{T}_z, z, \ell)$ and $\text{score}_2(\mathcal{T}_z, z, \ell)$ no matter which parameter $\rho \in I_j$ the algorithm $A'$ uses. Here, we critically use the fact that the scoring rule is path-wise.

Since $\rho\text{score}_1(\mathcal{T}_z, z, \ell) + (1-\rho)\text{score}_2(\mathcal{T}_z, z, \ell)$ is a linear function of $\rho$ for all $\ell$, there are at most $n$ intervals subdividing the interval $I_j$ such that the variable branched on at node $z$ is fixed. This is illustrated in Figure 4.6. Moreover, there are at most $2^{i-1}$ nodes at depth $i-1$, and each node similarly contributes a subpartition of $I_j$ of size $n$. If we merge all $2^{i-1}$ partitions, we have $T' \leq 2^{i-1}(n-1)+1$ intervals $I'_1, \ldots, I'_{T'}$ partitioning $I_j$ where for any interval $I'_p$ and any two parameters $\rho, \rho' \in I'_p$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees $A'$ builds using the scoring rules $\rho\text{score}_1 + (1-\rho)\text{score}_2$ and $\rho'\text{score}_1 + (1-\rho')\text{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. We can similarly subdivide each interval $I_1, \ldots, I_T$ for a total of

$$\bar{T} \leq 2^{(i-1)(i-2)/2} n^{i-1} \left(2^{i-1}(n-1)+1\right) \leq 2^{(i-1)(i-2)/2} n^{i-1} \left(2^{i-1} n\right) = 2^{i(i-1)/2} n^i$$

intervals $\bar{I}_1, \ldots, \bar{I}_{\bar{T}}$ partitioning $[0,1]$ such that for any interval $\bar{I}_t$, across all $\rho \in \bar{I}_t$ and any two parameters $\rho, \rho' \in \bar{I}_t$, if $\mathcal{T}$ and $\mathcal{T}'$ are the trees $A'$ builds using the scoring rules $\rho\text{score}_1 + (1-\rho)\text{score}_2$ and $\rho'\text{score}_1 + (1-\rho')\text{score}_2$, respectively, then $\mathcal{T}[i] = \mathcal{T}'[i]$. □

Next, we explicitly relate the behavior of B&B to $A'$, proving that the search tree B&B builds is a rooted subtree of the search tree $A'$ builds.

**Claim 4.3.16.** *Given a parameter $\rho \in [0,1]$, let $\mathcal{T}$ and $\mathcal{T}'$ be the trees B&B and $A'$ build respectively using the scoring rule $\rho\text{score}_1 + (1-\rho)\text{score}_2$. For any node $z$ of $\mathcal{T}$, let $\mathcal{T}_z$ be the path from the root of $\mathcal{T}$ to $z$. Then $\mathcal{T}_z$ is a rooted subtree of $\mathcal{T}'$.*

*Proof of Claim 4.3.16.* The path $\mathcal{T}_z$ can be labeled by a sequence of indices from $\{0,1\}$ and a sequence of variables from $\{x[1], \ldots, x[n]\}$ describing which variable is branched on and which value it takes on along the path $\mathcal{T}_z$. Let $((j_1, x[i_1]), \ldots, (j_t, x[i_t]))$ be this sequence of labels,

where $t$ is the number of edges in $\mathcal{T}_z$. We can similarly label every edge in $\mathcal{T}'$. We claim that there exists a path beginning at the root of $\mathcal{T}'$ with the labels $((j_1, x[i_1]), \ldots, (j_t, x[i_t]))$.

For a contradiction, suppose no such path exists. Let $(j_\tau, x[i_\tau])$ be the earliest label in the sequence $((j_1, x[i_1]), \ldots, (j_t, x[i_t]))$ where there is a path beginning at the root of $\mathcal{T}'$ with the labels $((j_1, x[i_1]), \ldots, (j_{\tau-1}, x[i_{\tau-1}]))$, but there is no way to continue the path using an edge labeled $(j_\tau, x[i_\tau])$. There are exactly two reasons why this could be the case:

1. The node at the end of the path with labels $((j_1, x[i_1]), \ldots, (j_{\tau-1}, x[i_{\tau-1}]))$ was fathomed by $A'$.

2. The algorithm $A'$ branched on a variable other than $x[i_\tau]$ at the end of the path labeled $((j_1, x[i_1]), \ldots, (j_{\tau-1}, x[i_{\tau-1}]))$.

In the first case, since $A'$ only fathoms a node if it is integral or infeasible, we know that B&B will also fathom the node at the end of the path with labels $((j_1, x[i_1]), \ldots, (j_{\tau-1}, x[i_{\tau-1}]))$. However, this is not the case since B&B next branches on the variable $x[i_\tau]$.

The second case is also not possible since the scoring rules are both path-wise. In a bit more detail, let $z'$ be the node at the end of the path with labels $((j_1, x[i_1]), \ldots, (j_{\tau-1}, x[i_{\tau-1}]))$. We refer to this path as $\mathcal{T}_{z'}$. Let $\bar{\mathcal{T}}$ (respectively, $\bar{\mathcal{T}}'$) be the state of the search tree B&B (respectively, $A'$) has built at the point it branches on $z'$. We know that $\mathcal{T}_{z'}$ is the path from the root to $z'$ in both of the trees $\bar{\mathcal{T}}$ and $\bar{\mathcal{T}}'$. Therefore, for all variables $x[k]$, $\rho\text{score}_1(\bar{\mathcal{T}}, z', k) + (1-\rho)\text{score}_2(\bar{\mathcal{T}}, z', k) = \rho\text{score}_1(\mathcal{T}_{z'}, z', k) + (1-\rho)\text{score}_2(\mathcal{T}_{z'}, z', k) = \rho\text{score}_1(\bar{\mathcal{T}}', z', k) + (1-\rho)\text{score}_2(\bar{\mathcal{T}}', z', k)$. This means that B&B and $A'$ will choose the same variable to branch on at the node $z'$.

Therefore, we have reached a contradiction, so the claim holds. □

Next, we use Claims 4.3.15 and 4.3.16 to prove Lemma 4.3.14. Let $I_1, \ldots, I_T$ be the intervals guaranteed to exist by Claim 4.3.15 and let $I_t$ be an arbitrary one of the intervals. Let $\rho'$ and $\rho''$ be two arbitrary parameters from $I_t$. We will prove that the scoring rules $\rho'\text{score}_1 + (1-\rho')\text{score}_2$ and $\rho''\text{score}_1 + (1-\rho'')\text{score}_2$ result in the same B&B search tree. For a contradiction, suppose that this is not the case. Consider the first iteration where B&B using the scoring rule $\rho'\text{score}_1 + (1-\rho')\text{score}_2$ differs from B&B using the scoring rule $\rho''\text{score}_1 + (1-\rho'')\text{score}_2$. By iteration, we mean lines 3 through 18 of Algorithm 3. Up until this iteration, B&B has built the same partial search tree $\mathcal{T}$. Since the node-selection policy does not depend on $\rho'$ or $\rho''$, B&B will choose the same leaf $z$ of the B&B search tree to branch on no matter which scoring rule it uses.

Suppose B&B chooses different variables to branch on in Step 5 of Algorithm 3 depending on whether it uses the scoring rule $\rho'\text{score}_1 + (1-\rho')\text{score}_2$ or $\rho''\text{score}_1 + (1-\rho'')\text{score}_2$. Let $\mathcal{T}_z$ be the path from the root of $\mathcal{T}$ to $z$. By Claim 4.3.15, we know that the algorithm $A'$ builds the same search tree using the two scoring rules. Let $\bar{\mathcal{T}}'$ (respectively, $\bar{\mathcal{T}}''$) be the state of the search tree $A'$ has built using the scoring rule $\rho'\text{score}_1 + (1-\rho')\text{score}_2$ (respectively, $\rho''\text{score}_1 + (1-\rho'')\text{score}_2$) by the time it branches on the node $z$. By Claims 4.3.15 and 4.3.16, we know that $\mathcal{T}_z$ is the path of from the root to $z$ of both $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$. By Claim 4.3.15, we know that $A'$ will branch on the same variable $x[i]$ at the node $z$ in both the trees $\bar{\mathcal{T}}'$ and $\bar{\mathcal{T}}''$, so $i = \text{argmax}_j \left\{\rho'\text{score}_1(\bar{\mathcal{T}}', z, j) + (1-\rho')\text{score}_2(\bar{\mathcal{T}}', z, j)\right\}$, or equivalently,

$$i = \text{argmax}_j \left\{\rho'\text{score}_1(\mathcal{T}_z, z, j) + (1-\rho')\text{score}_2(\mathcal{T}_z, z, j)\right\}, \tag{4.5}$$

and $i = \text{argmax}_j \left\{\rho''\text{score}_1(\bar{\mathcal{T}}'', z, j) + (1-\rho'')\text{score}_2(\bar{\mathcal{T}}'', z, j)\right\}$, or equivalently,

$$i = \text{argmax}_j \left\{\rho''\text{score}_1(\mathcal{T}_z, z, j) + (1-\rho'')\text{score}_2(\mathcal{T}_z, z, j)\right\}. \tag{4.6}$$

Returning to the search tree $\mathcal{T}$ that B&B is building, Equation (4.5) implies that

$$i = \text{argmax}_j \left\{\rho'\text{score}_1(\mathcal{T}, z, j) + (1-\rho')\text{score}_2(\mathcal{T}, z, j)\right\}$$

and Equation (4.6) implies that $i = \text{argmax}_j \left\{\rho''\text{score}_1(\mathcal{T}, z, j) + (1-\rho'')\text{score}_2(\mathcal{T}, z, j)\right\}$. Therefore, B&B will branch on $x[i]$ at the node $z$ no matter which scoring rule it uses.

Finally, since the trees B&B has built so far are identical, the choice of whether or not to fathom the children $z_i^+$ and $z_i^-$ does not depend on the scoring rule, so B&B will fathom the same nodes no matter whether it uses the scoring rule $\rho'\text{score}_1 + (1-\rho')\text{score}_2$ or $\rho''\text{score}_1 + (1-\rho'')\text{score}_2$. Therefore, we have reached a contradiction: the iterations were identical. We conclude that the lemma holds. □

This lemma implies the following pseudo-dimension bound. The proof is nearly identical to that of Theorem 3.0.3 from Chapter 3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 4.3.17.** *The pseudo-dimension of $\mathcal{U} = \left\{u_\rho : \rho \in [0,1]\right\}$ 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 $u_\rho$ over the samples and the expected value of $u_\rho$ is at most $\epsilon$: $\left|\frac{1}{N}\sum_{z\in\mathcal{S}} u_\rho(z) - \mathbb{E}_{z\sim\mathcal{D}}\left[u_\rho(z)\right]\right| \leq \epsilon$.

### General scoring rules

In this section, we provide sample complexity guarantees that apply to learning convex combinations of any set of scoring rules. These guarantees apply to MILPs with arbitrary integer constraints, whereas in the previous section, we required the non-continuous variables to be binary. Unlike the results in the previous section, these guarantees depend on the size of the search trees B&B is allowed to build before terminating, which we denote as $\kappa$. The following lemma corresponds to Lemma 4.3.14 for this setting.

**Lemma 4.3.18.** *Let $\text{score}_1, \ldots, \text{score}_d$ be $d$ arbitrary scoring rules and let $z$ be an arbitrary MILP over $n$ integer 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 $(\rho[1], \ldots, \rho[d]) \in R$.*

*Proof.* The proof has two steps. In Claim 4.3.19, we show that there are at most $n^\kappa$ different search trees that Algorithm 3 might produce for the MILP $z$ as we vary the mixing parameter vector $(\rho[1], \ldots, \rho[d])$. In Claim 4.3.20, for each of the possible search trees $\mathcal{T}$ that might be produced, we show that the set of parameter values $(\rho[1], \ldots, \rho[d])$ which give rise to that tree lie in the intersection of $\kappa n^2$ halfspaces. These facts together prove the lemma.

**Claim 4.3.19.** *There are only $n^\kappa$ different search trees that can be achieved by varying the parameter vector $(\rho[1], \ldots, \rho[d])$.*

*Proof of Claim 4.3.19.* Fix any $d$ mixing parameters $(\rho[1], \ldots, \rho[d])$ and let $v_1, \ldots, v_\kappa \in [n]$ be the sequence of branching variables chosen by B&B run with scoring rule $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$, ignoring which node of the tree each variable was chosen for. That is, $v_1$ is the variable branched on at the root, $v_2$ is the variable branched on at the next unfathomed node chosen by the node-selection policy, and so on. If Algorithm 3 with scoring rule $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ produces a tree of size $k < \kappa$, then define $v_t = 1$ for all $t \geq k$ (we are just padding the sequence $v_1, v_2, \ldots$ so that it has length $\kappa$). We will show that whenever two sets of mixing parameters $(\rho[1], \ldots, \rho[d])$ and $(\rho[1]', \ldots, \rho[d]')$ give rise to the same sequence of branching variable selections, they in fact produce identical search trees. This will imply that the number of distinct trees that can be produced by B&B with scoring rules of the form $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ is at most $n^\kappa$, since there are only $n^\kappa$ distinct sequences of $\kappa$ variables $v_1, \ldots, v_\kappa \in [n]$.

Let $(\rho[1], \ldots, \rho[d])$ and $(\tilde{\rho}[1], \ldots, \tilde{\rho}[d])$ be two sets of mixing parameters, and suppose running B&B with $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ and $\tilde{\rho}[1]\text{score}_1 + \cdots + \tilde{\rho}[d]\text{score}_d$ both results in the sequence of branching variable decisions being $v_1, \ldots, v_\kappa$. We prove that the resulting search trees are identical by induction on the iterations of the algorithm, where an iteration corresponds to Lines 4 through 18 of Algorithm 3. Our base case is before the first iteration when the two trees are trivially equal, since they both contain just the root node. Now suppose that up until the beginning of iteration $t$ the two trees were identical. Since the two trees are identical, the node-selection policy will choose the same node to branch on in both cases. In both trees, the algorithm will choose the same variable to branch on, since the sequence of branching variable choices $v_1, \ldots, v_\kappa$ is shared. Since the two trees are identical, we will branch using the same constraint $v_t \leq a$ for some integer $a \in \mathbb{Z}$ in both trees. Finally, if any of the children are fathomed, they will be fathomed in both trees, since the they are identical. It follows that all steps of B&B maintain equality between the two trees, and the claim follows. Also, whenever the sequence of branching variables differ, then the search tree produced will not be the same. In particular, on the first iteration where the two sequences disagree, the tree built so far will be identical up to that point, but the next variable branched on will be different, leading to different trees. □

Next, we argue that for any given B&B search tree $\mathcal{T}$, the set of mixing parameters $\boldsymbol{\rho} \in [0, 1]^d$ giving rise to $\mathcal{T}$ is defined by the intersection of $n^{\kappa+2}$ halfspaces.

**Claim 4.3.20.** *For a given search tree $\mathcal{T}$, there are at most $\kappa n^2$ halfspaces such that Algorithm 3 using the scoring rule $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ builds the tree $\mathcal{T}$ if and only if $(\rho[1], \ldots, \rho[d])$ lies in the intersection of those halfspaces.*

*Proof of Claim 4.3.20.* Let $v_1, \ldots, v_\kappa$ be the sequence of branching variable choices that gives rise to tree $\mathcal{T}$. We will prove the claim by induction on iterations completed by B&B. Let $\mathcal{T}_t$ be the state of B&B after $t$ iterations.

**Induction hypothesis.** For a given index $t \in [\kappa]$, there are at most $tn^2$ halfspaces such that B&B using the scoring rule $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ builds the partial tree $\mathcal{T}_t$ after $t$ iterations if and only if $(\rho[1], \ldots, \rho[d])$ lies in the intersection of those halfspaces.

**Base case.** In the base case, before the first iteration, the set of parameters that will produce the partial search tree consisting of just the root is the entire set of parameters, which vacuously is the intersection of zero hyperplanes.

**Inductive step.** For the inductive step, let $t < \kappa$ be an arbitrary tree size. By the inductive hypothesis, we know that there exists a set $\mathcal{B}$ of at most $tn^2$ halfspaces such that B&B using the scoring rule $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ builds the partial tree $\mathcal{T}_t$ after $t$ iterations if and only if $(\rho[1], \ldots, \rho[d])$ lies in the intersection of those halfspaces. Let $z'$ be the IP contained in the next node that B&B will branch on given $\mathcal{T}_t$. We know that B&B will choose to branch on variable $v_{t+1}$ at this node if and only if

$$\begin{aligned}&\rho[1]\text{score}_1(\mathcal{T}_t, z', v_{t+1}) + \cdots + \rho[d]\text{score}_d(\mathcal{T}_t, z', v_{t+1})\\ &> \max_{v' \neq v_{t+1}} \left\{\rho[1]\text{score}_1(\mathcal{T}_t, z', v') + \cdots + \rho[d]\text{score}_d(\mathcal{T}_t, z', v')\right\}.\end{aligned}$$

Since these functions are linear in $(\rho[1], \ldots, \rho[d])$, there are at most $n^2$ halfspaces defining the region where $v_{t+1} = \text{argmax}\{\rho[1]\text{score}_1(\mathcal{T}, z', v') + \cdots + \rho[d]\text{score}_d(\mathcal{T}, z', v')\}$. Let $\mathcal{B}'$ be this set of halfspaces. B&B using the scoring rule $\rho[1]\text{score}_1 + \cdots + \rho[d]\text{score}_d$ builds the partial tree $\mathcal{T}_{t+1}$ after $t+1$ iterations if and only if $(\rho[1], \ldots, \rho[d])$ lies in the intersection of the $(t+1)n^2$ halfspaces in the set $\mathcal{B} \cup \mathcal{B}'$. □

□

This piecewise structure implies the following pseudo-dimension guarantee.

**Theorem 4.3.21.** *The pseudo-dimension of* $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in [0,1]^d\right\}$ *is* $O\left(d\left(\kappa \log n + \log d\right)\right)$.

*Proof.* Suppose that $\text{Pdim}(\mathcal{U}) = N$ and let $\mathcal{S} = \{z_1, \ldots, z_N\}$ be a shatterable set of problem instances. We know there exists a set of targets $r_1, \ldots, r_N \in \mathbb{R}$ that witness the shattering of $\mathcal{S}$ by $\mathcal{U}$. This means that for every $\mathcal{S}' \subseteq \mathcal{S}$, there exists a parameter vector $(\rho_{1,\mathcal{S}'}, \ldots, \rho_{d,\mathcal{S}'})$ such that if $z_i \in \mathcal{S}'$, then $\text{cost}\left(z_i, \rho_{1,\mathcal{S}'}\text{score}_1 + \cdots + \rho_{d,\mathcal{S}'}\text{score}_d\right) \leq r_i$. Otherwise

$$\text{cost}\left(z_i, \rho_{1,\mathcal{S}'}\text{score}_1 + \cdots + \rho_{d,\mathcal{S}'}\text{score}_d\right) > r_i.$$

Let $M = \{(\rho_{1,\mathcal{S}'}, \ldots, \rho_{d,\mathcal{S}'}) : \mathcal{S}' \subseteq \mathcal{S}\}$. We will prove that $|M| = O\left(d\left(N\kappa n^{\kappa+2}\right)^d\right)$, and since $2^N = |M|$, this means that $\text{Pdim}(\mathcal{U}) = N = O\left(d\kappa \log n + d \log d\right)$ (see Lemma 2.1.4 in Section 2.1.1).

To prove our bound on $|M|$, we rely on Lemma 4.3.18, which tells us that for any problem instance $z$, there is a set $\mathcal{H}$ of at most $T \leq \kappa n^{\kappa+2}$ 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 $(\rho_1, \ldots, \rho_d) \in R$. If we merge all $T$ hyperplanes for all samples in $\mathcal{S}$, we are left with a set $\mathcal{H}'$ of $T' \leq m\kappa n^{\kappa+2}$ hyperplanes where for any connected component $R$ of $[0,1]^d \setminus \mathcal{H}'$ and any $z_i \in \mathcal{S}$, the search tree B&B builds using the scoring rule $\rho_1\text{score}_1 + \cdots + \rho_d\text{score}_d$ given as input $z_i$ is invariant across all $(\rho_1, \ldots, \rho_d) \in R$. Therefore, at most one element of $M$ can come from each connected component, of which there are $O(d|\mathcal{H}'|^d) = O\left(d\left(N\kappa n^{\kappa+2}\right)^d\right)$. Therefore, $|M| = O\left(d\left(N\kappa n^{\kappa+2}\right)^d\right)$, as claimed. □

**Algorithm 4** ERM Algorithm

**Input:** Problem instances $z_1, \ldots, z_m$, variable scoring rules $\text{score}_1, \text{score}_2$.

1: For each problem instance $z_i$, compute the piecewise constant $u_\rho$ as a function of the mixing parameter $\rho$.

2: Compute the point-wise average of the resulting piecewise constant functions.

**Output:** The $\rho^*$ in the interval minimizing the average cost.

In contrast to Theorem 4.3.17, this pseudo-dimension bound implies a sample complexity bound of $N_{\mathcal{U}}(\epsilon, \delta) := \Theta\left(\frac{H^2}{\epsilon^2}\left(d\left(\kappa \log n + \log d\right) + \ln \frac{1}{\delta}\right)\right)$, where $[-H, H]$ is the range of each utility function $u_\rho$. In some ways, this bound is stronger than that implied by Theorem 4.3.17 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 $\kappa$, whereas the bound implied by Theorem 4.3.17 only depends on the number of variables $n$.

### Learning algorithm

In this section we describe an empirical risk minimization algorithm capable of finding the best mixture of two variable-selection scoring rules for a given set of $m$ problem instances $z_1, \ldots, z_m$. We modify the tree search algorithm so that given a problem instance $z$ and a mixing paramter $\rho \in [0, 1]$, the search algorithm keeps track of the largest interval $I \subset [0, 1]$ such that the behavior of the algorithm is identical to the current run when run with any parameter $\rho' \in I$. With this, we can enumerate all possible behaviors of the algorithm for a single instance $z$ by running the algorithm with $\rho = 0$, followed by the smallest value of $\rho$ that will give a different outcome, and so on, until we have covered the entire interval $[0, 1]$. This procedure results in running the tree search algorithm on the instance $z$ exactly once for each possible behavior achievable across all values of the parameter $\rho \in [0, 1]$. By applying this algorithm to each problem $z_i$ for $i \in \{1, \ldots, m\}$, we discover how the tree search algorithm would perform on every instance for any value of the mixing parameter $\rho$. This allows us to divide the interval $[0, 1]$ into a finite number of intervals on which $\text{cost}$ is piecewise constant, and to compute the cost on each interval.

To see why this additional book keeping is possible, suppose we are choosing which variable to branch on in node $z$ of tree $\mathcal{T}$. We have two scoring rules $\text{score}_1$ and $\text{score}_2$ that each rank the candidate variables in $z$, and when we run the algorithm with parameter $\rho$, we combine these two scores as $(1 - \rho)\text{score}_1(\mathcal{T}, z, i) + \rho\text{score}_2(\mathcal{T}, z, i)$. For any parameter $\rho$ which results in the same variable having the highest score, the variable chosen for branching in this node will be identical. Let $i^*$ be the variable chosen by the algorithm when run with parameter $\rho$. The set of all $\rho'$ for which $i^*$ is the variable of the highest score is an interval (and its end points can be found by solving a linear equation to determine the value of $\rho'$ for which some other variable overtakes $i^*$ under the mixed score). Also, for every parameter $\rho'$ outside of this interval, the algorithm would indeed branch on a different node, resulting in a different outcome of the tree search algorithm. By taking the intersections of these intervals across all branching variable choices, we find the largest subset of $[0, 1]$ for which the algorithm would behave exactly the same, and this subset is an interval. The overhead of this book keeping is only linear in the number of candidate branch variables. Pseudo-code for the ERM algorithm is given in Algorithm 4.

**Generalization to non-linear combinations**

In this section, we show that our analysis from the previous sections also implies guarantees for learning non-linear combinations of the form $\prod_{i=1}^{d} \text{score}_i^{\rho[i]}$. In the following lemma, we prove that sample complexity bounds for learning convex combinations of scoring rules imply sample complexity bounds for learning non-linear combinations of scoring rules. It formally shows that if $m$ samples are sufficient to ensure that for any convex combination $\sum_{j=1}^{d} \rho[j]\text{score}_j$, average cost over the samples is $\epsilon$-close to expected cost, then $m$ samples are also sufficient to ensure the same holds for the non-convex combination $\prod_{i=1}^{d} \text{score}_i^{\rho[i]}$. Technically, this lemma holds for any $d$ scoring rules $\text{score}_1, \ldots, \text{score}_d$ from a set $\Psi$ that is *closed under logarithm*: for any function $\text{score} \in \Psi$, $\log \text{score}$ is also in $\Psi$. For example, $\Psi$ might be the set of all scoring rules or the set of path-wise scoring rules. These sets are both closed under logarithm; for example, if score is path-wise, then $\log \text{score}$ is also path-wise. In this section, we use the notation $u'_{\boldsymbol{\rho}}(z)$ to denote the utility of B&B (for example, its tree size) using the scoring rule $\prod_{i=1}^{d} \text{score}_i^{\rho[i]}$ (whereas $u_{\boldsymbol{\rho}}(z)$ denotes the utility of B&B using the scoring rule $\sum_{j=1}^{d} \rho[j]\text{score}_j$). We assume that $u'_{\boldsymbol{\rho}}$ is tree-constant, as is $u_{\boldsymbol{\rho}}$ (as defined in Section 4.3.1).

**Lemma 4.3.22.** *Let $\Psi$ be a set of scoring rules that is closed under logarithm. Suppose there exists a sample complexity function $m_\Psi : \mathbb{R}_{>0} \times (0,1) \to \mathbb{N}$ such that for any distribution $\mathcal{D}$ over MILPs, any $\text{score}_1, \ldots, \text{score}_d \in \Psi$, any $\epsilon > 0$, and any $\delta \in (0,1)$, with probability $1-\delta$ over the draw of $m \geq m_\Psi(\epsilon, \delta)$ MILPs $z_1, \ldots, z_m \sim \mathcal{D}$, for any $\rho[1], \ldots, \rho[d] \in [0,1]$,*

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

*Then for any $\text{score}_1, \ldots, \text{score}_d \in \Psi$, with probability at least $1-\delta$ over the draw of $m \geq m_\Psi(\epsilon, \delta)$ MILPs $z_1, \ldots, z_m \sim \mathcal{D}$, for any $\rho[1], \ldots, \rho[d] \in [0,1]$, $\left| \frac{1}{m} \sum_{i=1}^{m} u'_{\boldsymbol{\rho}}(z_i) - \mathbb{E}_{z \sim \mathcal{D}} \left[ u'_{\boldsymbol{\rho}}(z) \right] \right| \leq \epsilon$.*

*Proof.* We claim that Algorithm 3 builds the same tree using the scoring rule $\prod_{i=1}^{d} \text{score}_i^{\rho[i]}$ as it does using the scoring rule $\sum_{i=1}^{d} \rho[i] \log \text{score}_i$. Therefore, since $u_{\boldsymbol{\rho}}$ and $u'_{\boldsymbol{\rho}}$ are tree-constant, the lemma statement holds.

For a contradiction, suppose Algorithm 3 does not build the same tree using the scoring rule $\prod_{i=1}^{d} \text{score}_i^{\rho[i]}$ as it does using the scoring rule $\sum_{i=1}^{d} \rho[i] \log \text{score}_i$. Consider the first round of Algorithm 3 where the algorithm's behavior using the scoring rule $\prod_{i=1}^{d} \text{score}_i^{\rho[i]}$ differs from its behavior using the scoring rule $\sum_{i=1}^{d} \rho[i] \log \text{score}_i$. Let $\mathcal{T}'$ be the tree Algorithm 3 has built up until that round and let $z$ be the MILP represented by the next node that Algorithm 3 will branch on given $\mathcal{T}'$. It must be that at Step 5, the algorithm chose a different variable to branch on depending on the scoring rule. In other words,

$$\text{argmax}_j \left\{ \prod_{i=1}^{d} \text{score}_i(\mathcal{T}', z, j)^{\rho[i]} \right\} \neq \text{argmax}_j \left\{ \sum_{i=1}^{d} \rho[i] \log \text{score}_i(\mathcal{T}', z, j) \right\}.$$

However, if $j^* = \text{argmax}_j \left\{ \prod_{i=1}^d \text{score}_i(\mathcal{T}', z, j)^{\rho[i]} \right\}$, then[3]

$$j^* = \text{argmax}_j \left\{ \log \left( \prod_{i=1}^d \text{score}_i(\mathcal{T}', z, j)^{\rho[i]} \right) \right\},$$

which means that $j^* = \text{argmax}_j \left\{ \sum_{i=1}^d \rho[i] \log \text{score}_i(\mathcal{T}', z, j) \right\}$, which is a contradiction. Therefore, Algorithm 3 builds the same tree using the scoring rule $\prod_{i=1}^d \text{score}_i^{\rho[i]}$ as it does using the scoring rule $\sum_{i=1}^d \rho[i] \log \text{score}_i$, so the lemma statement holds. □

This implies the following corollary for learning non-linear combinations of path-wise scoring rules. It follows from Lemma 4.3.22, Theorem 4.3.17, and the fact that if score is path-wise, then $\log \text{score}$ is path-wise as well.

**Corollary 4.3.23.** *Let* $\text{score}_1$ *and* $\text{score}_2$ *be two path-wise scoring rules and let* $[-H, H]$ *be the range of each utility function* $u'_\rho$*. For any distribution* $\mathcal{D}$ *over MILPs with at most* $n$ *binary variables, any* $\epsilon > 0$*, and any* $\delta \in (0, 1)$*,* $m = O\left(\frac{H^2}{\epsilon^2}\left(n^2 + \ln \frac{1}{\delta}\right)\right)$ *samples are sufficient to ensure that with probability at least* $1 - \delta$ *over the draw* $z_1, \ldots, z_m \sim \mathcal{D}$*, for any* $\rho \in [0, 1]$*,* $\left| \mathbb{E}_{z \sim \mathcal{D}} \left[ u'_\rho(z) \right] - \frac{1}{m} \sum_{i=1}^m u'_\rho(z_i) \right| \leq \epsilon$.

Finally, we prove a similar corollary for non-linear combinations of general scoring rules, which follows from Lemma 4.3.22 and Theorem 4.3.21.

**Corollary 4.3.24.** *Let* $\text{score}_1, \ldots, \text{score}_d$ *be* $d$ *arbitrary scoring rules, let* $[-H, H]$ *be the range of each utility function* $u'_{\boldsymbol{\rho}}$*, and let* $\kappa$ *be a tree size cap. For any distribution* $\mathcal{D}$ *over MILPs* $z$ *with at most* $n$ *integer variables,* $m = O\left(\frac{H^2}{\epsilon^2}\left(d\kappa \log n + d \log d + \ln \frac{1}{\delta}\right)\right)$ *samples are sufficient to ensure that with probability at least* $1 - \delta$ *over the draw* $z_1, \ldots, z_m \sim \mathcal{D}$*, for any* $\boldsymbol{\rho} \in [0, 1]^d$*,*

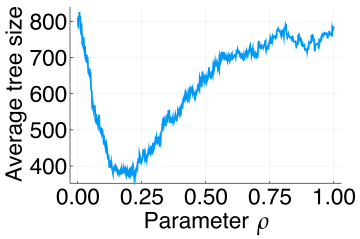
$$\left| \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ u'_{\boldsymbol{\rho}}(z) \right] - \frac{1}{m} \sum_{i=1}^m u'_{\boldsymbol{\rho}}(z_i) \right| \leq \epsilon.$$

## 4.4 Experiments

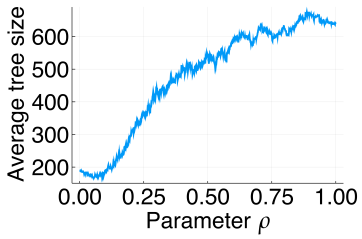
In this section, we show that the parameter of the variable-selection policy in B&B algorithms for MILP can have a dramatic effect on the average tree size generated for several domains, and no parameter value is optimal across these distributions.

**Experimental setup.** We use the C API of IBM ILOG CPLEX 12.8.0.0 to override the default variable-selection policy using a branch callback. Additionally, our callback performs extra book-keeping 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 4.3.2. We run CPLEX exactly once for each possible B&B tree on each instance. The CPLEX node-selection policy is set to "best bound" (aka. $A^*$ in AI), which is the most typical choice in MILP. All experiments were run on a 64-core machine with 512 GB of RAM.

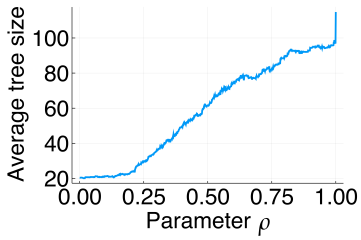
[3]For any set $X \subset \mathbb{R}_{>0}$, $\text{argmax}_{x \in X}\{x\} = \text{argmax}_{x \in X}\{\log x\}$. After all, if not, then there are $x_1, x_2 \in X$ such that $x_1 < x_2$ but $\log x_1 > \log x_2$, which is a contradiction.



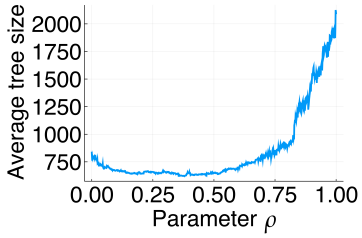

(a) CATS "arbitrary": 200 bids, 100 goods




(b) CATS "regions": 400 bids, 200 goods




(c) Clustering




(d) Linear separators

Figure 4.7: The average tree size produced by B&B when run with the linear scoring rule with parameter $\rho$ (Equation (4.7)).

For a variety of application domains detailed below, Figure 4.7 shows the average B&B tree size produced for each possible value of the $\rho$ parameter for the linear scoring rule, averaged over 100 independent samples from each distribution. Specifically, the scoring rule whose parameter we tune in Figure 4.7 has the form

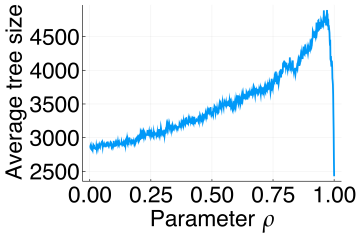
$$\text{score}(\mathcal{T}, z, i) = (1-\rho) \cdot \min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\} + \rho \cdot \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}, \tag{4.7}$$

where (as defined in Section 4.2) $\check{c}_z$, $\check{c}_{z_i^-}$, and $\check{c}_{z_i^-}$ are the objective values of the optimal solutions[4] to the LP relaxations of $z$, $z_i^+$, and $z_i^-$.

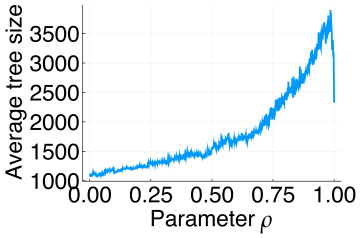
In Figure 4.8, we plot average tree size as a function of the $\rho$ parameter for the linear scoring rule, but we use pseudo-cost branching rather than strong branching (which we use in Figure 4.7). Using strong branching, B&B computes the changes in the LP relaxation objective values $\check{c}_z - \check{c}_{z_i^+}$ and $\check{c}_z - \check{c}_{z_i^-}$ for every variable, which is time-consuming since it involves evaluating $2n$ LPs at every node. Pseudo-cost branching instead estimates these values by averaging the LP objective value changes across all nodes in the tree where the $i^{th}$ variable was chosen to branch on. Pseudo-cost branching generally leads to larger trees than strong branching. In Figures 4.8a, 4.8b, and 4.8d, we average over 100 randomly sampled IPs. Since Figure 4.8c is choppier than these other three, we average over 2000 samples.

Finally, Figure 4.9 illustrates the average B&B tree size produced when tuning the parameter of a generalized product rule, averaged over 100 samples from each distribution. This

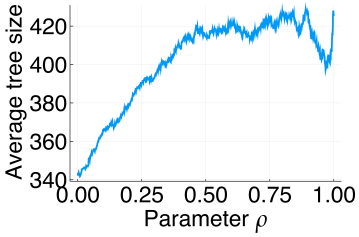
[4]When computing $\check{c}_{z_i^-}$ and $\check{c}_{z_i^-}$ for all candidate variables, we limit CPLEX to run at most 10 dual steepest-edge iterations.



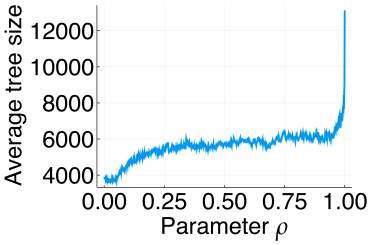

(a) CATS "arbitrary": 150 bids, 100 goods




(b) CATS "regions": 300 bids, 200 goods

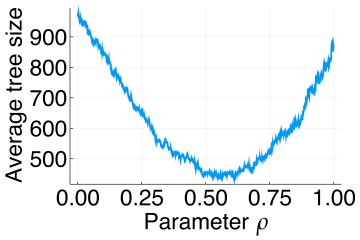



(c) Clustering

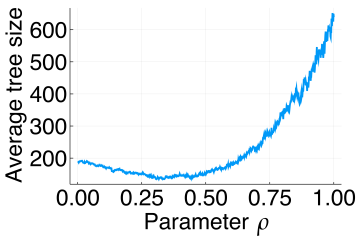



(d) Linear separators

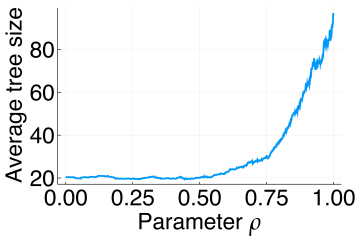
Figure 4.8: The average tree size produced by B&B when run with the linear scoring rule with parameter $\rho$ using pseudo-cost branching.



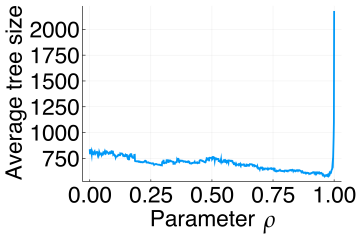

(a) CATS "arbitrary": 200 bids, 100 goods




(b) CATS "regions": 400 bids, 200 goods




(c) Clustering




(d) Linear separators

Figure 4.9: The average tree size produced by B&B when run with the product scoring rule with parameter $\rho$ (Equation (4.8)).

parameterized scoring rule has the form[5][6]

$$\text{score}(\mathcal{T}, z, i) = \text{score}_1(\mathcal{T}, z, i)^{1-\rho}\text{score}_2(\mathcal{T}, z, i)^{\rho}, \tag{4.8}$$

where

$$\text{score}_1(\mathcal{T}, z, i) = \max\left\{\min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}, 10^{-6}\right\}$$

and

$$\text{score}_2(\mathcal{T}, z, i) = \max\left\{\max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}, 10^{-6}\right\}.$$

This scoring rule is equivalent to the product scoring rule from Section 4.2 when $\rho = \frac{1}{2}$ because for any $i, j \in [n]$, $\text{score}_1(\mathcal{T}, z, i)^{\frac{1}{2}}\text{score}_2(\mathcal{T}, z, i)^{\frac{1}{2}} \geq \text{score}_1(\mathcal{T}, z, j)^{\frac{1}{2}}\text{score}_2(\mathcal{T}, z, j)^{\frac{1}{2}}$ if and only if $\max\left\{\check{c}_z - \check{c}_{z_i^-}, 10^{-6}\right\} \cdot \max\left\{\check{c}_z - \check{c}_{z_i^+}, 10^{-6}\right\} \geq \max\left\{\check{c}_z - \check{c}_{z_j^-}, 10^{-6}\right\} \cdot \max\left\{\check{c}_z - \check{c}_{z_j^+}, 10^{-6}\right\}$.

We now detail the application domains we analyze in our experiments.

**Combinatorial auctions.** We generate instances of the combinatorial auction winner determination problem under the OR-bidding language [Sandholm, 2002], which makes this problem equivalent to weighted set packing. The problem is NP-complete. We encode each instance as a binary MILP (see Example 4.2.1). We use the Combinatorial Auction Test Suite (CATS) [Leyton-Brown et al., 2000] to generate these instances. In Figures 4.7 and 4.9 use the "arbitrary" generator with 200 bids and 100 goods, resulting in MILPs with around 200 variables, and "regions" generator with 400 bids and 200 goods, resulting in MILPs with around 400 binary variables. In Figure 4.8, we use the "arbitrary" generator with 150 bids and 100 goods, resulting in MILPs with around 150 binary variables, and "regions" generator with 300 bids and 200 goods, resulting in MILPs with around 300 binary variables.

**Clustering.** Given $n$ points $P = \{p_1, \ldots, p_n\}$ 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: $\sum_{i=1}^{n} \min_{j\in[k]} d\left(p_i, c_j\right)^2$. We can formulate $k$-means clustering as a MILP by assigning a binary variable $x_i$ to each point $p_i$ where $x_i = 1$ if and only if $p_i$ is a center, as well as a binary variable $y_{ij}$ for each pair of points $p_i$ and $p_j$, where $y_{ij} = 1$ if and only if $p_j$ is a center and $p_j$ is the closest center to $p_i$. We want to solve the following problem:

$$\begin{array}{lll}
\min & \sum_{i,j\in[n]} d\left(p_i, p_j\right) y_{ij} & \\
\text{s.t.} & \sum_{i=1}^{n} x_i = k & \\
& \sum_{j=1}^{n} y_{ij} = 1 & \forall i \in [n] \\
& y_{ij} \leq x_j & \forall i, j \in [n] \\
& x_i \in \{0, 1\} & \forall i \in [n] \\
& y_{ij} \in \{0, 1\} & \forall i, j \in [n].
\end{array}$$

We generate 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

[5]Again, when computing $\check{c}_{z_i^-}$ and $\check{c}_{z_i^-}$ for all candidate variables, we limit CPLEX to run at most 10 dual steepest-edge iterations.

[6]As we write in Section 4.2, 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 $\min\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}^{1-\rho} \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}^{\rho}$ without comparing to $10^{-6}$. If one of these multiplicands equals 0, then the score equals 0, canceling out the value of the other multiplicand and thus losing the information encoded by that multiplicand.

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 in $\mathbb{R}^d$. These MILPs have 1260 binary variables.

**Agnostically learning linear separators.** Let $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N \in \mathbb{R}^d$ be labeled by $z_1, \ldots, z_N \in \{-1, 1\}$. Suppose we wish to learn a linear separator $\boldsymbol{w} \in \mathbb{R}^d$ that minimizes 0-1 loss, i.e.,

$$\sum_{i=1}^{N} \mathbf{1}_{\{z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle < 0\}}.$$

We can formulate this problem as a MILP as follows. Let $M > \max \|\boldsymbol{p}_i\|_1$.

$$\begin{array}{lll} \min & \sum_{i=1}^{n} x_i & \\ \text{s.t.} & z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle > -M x_i & \forall i \in [n] \\ & w[i] \in [-1, 1] & \forall i \in [n] \\ & x_i \in \{0, 1\} & \forall i \in [n]. \end{array}$$

Since $|\langle \boldsymbol{p}_i, \boldsymbol{w} \rangle| < M$, the inequality $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle > -M x_i$ ensures that if $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle > 0$, then $x_i$ will equal 0, but if $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle \leq 0$, then $x_i$ must equal 1.[7]

We generate problem instances with 100 points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{100}$ from the 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 15 random points and flip their labels so that there is no consistent linear separator. These MILPs have 100 binary variables.

**Discussion.** 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. For example, the optimal parameter in Figures 4.7b, 4.7c, and 4.7d is close to 0. However, $\rho = 0$ severely suboptimal in Figure 4.7a, resulting in trees that are twice the size of the trees obtained under the optimal parameter value.

## 4.5 Generalization beyond variable selection

In this section, we study the sample complexity of selecting high-performing parameters for generic tree-based algorithms, which are a generalization of B&B. This abstraction allows us to provide guarantees for simultaneously optimizing key aspects of tree search beyond cut selection, including node selection and cutting plane selection. The results in the section are from ongoing research with Nina Balcan, Siddharth Prasad, and Tuomas Sandholm [Balcan et al., 2021b].

Tree search algorithms take place over a series of $\kappa$ *rounds* (analogous to the B&B tree-size cap $\kappa$ in the previous sections). There is a sequence of $t$ *steps* that the algorithm takes on each round. For example, in branch-and-cut, these steps include node selection, cut selection, and variable selection. The specific *action* the algorithm takes during each step (for example, which node to select, which cut to include, or which variable to branch on) typically depends on a

[7] In practice, we implement this constraint by enforcing that $z_i \langle \boldsymbol{p}_i, \boldsymbol{w} \rangle \geq -M x_i + \gamma$ for some tiny $\gamma > 0$ since MILP solvers cannot enforce strict inequalities.

*scoring rule*. We provided examples of scoring rules for variable selection in Section 4.2 and in Section 4.5.1, we give examples of scoring rules for cutting plane selection. The scoring rule weights each possible action and the algorithm performs the action with the highest weight. These actions (deterministically) transition the algorithm from one *state* to another. This high-level description of tree search is summarized by Algorithm 5. For each step $j \in [t]$, the number of possible actions is $T_j \in \mathbb{N}$. There is a scoring rule $\text{score}_j$, where $\text{score}_j(k, s) \in \mathbb{R}$ is the weight associated with the action $k \in \left[T_j\right]$ when the algorithm is in the state $s$.

**Algorithm 5** Tree search

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

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

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

There are often several scoring rules one could use, and it is not clear which to use in which scenarios. As in the previous section, we provide guarantees for learning combinations of these scoring rules for the particular application at hand. More formally, for each step $j \in [t]$, rather than just a single scoring rule $\text{score}_j$ as in Step 4, there are $d_j$ scoring rules $\text{score}_{j,1}, \ldots, \text{score}_{j,d_j}$. Given parameters $\boldsymbol{\rho}_j = \left(\rho_j[1], \ldots, \rho_j[d_j]\right) \in \mathbb{R}^{d_j}$, the algorithm takes the action $k \in [T_j]$ that maximizes $\sum_{i=1}^{d_j} \rho_j[i]\text{score}_{j,i}(k, s)$. There is a distribution $\mathcal{D}$ over inputs $z$ to Algorithm 5. For example, when this framework is instantiated for branch-and-cut, $z$ is an integer program $(\boldsymbol{c}, A, \boldsymbol{b})$. There is a utility function $u_{\boldsymbol{\rho}}(z) \in [-H, H]$ that measures the utility of the algorithm parameterized by $\boldsymbol{\rho} = (\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_t)$ on input $z$. For example, this utility function might measure the size of the search tree that the algorithm builds. We assume that this utility function is *final-state-constant* (which is a generalization of *tree-constant* from Section 4.3.1):

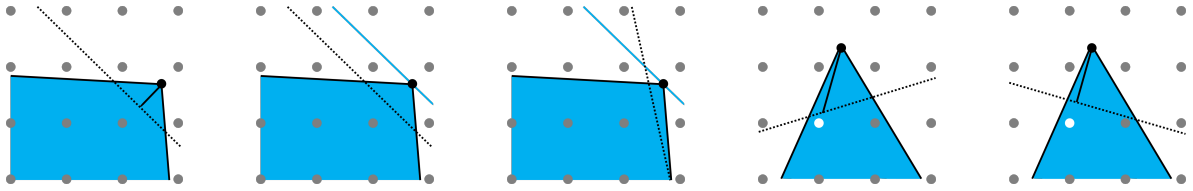
**Definition 4.5.1.** Let $\boldsymbol{\rho} = (\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_t)$ and $\boldsymbol{\rho}' = (\boldsymbol{\rho}'_1, \ldots, \boldsymbol{\rho}'_t)$ be two parameter vectors. Suppose that we run Algorithm 5 on input $z$ once using the scoring rule $\text{score}_j = \sum_{i=1}^{d_j} \rho_j[i]\text{score}_{j,i}$ and once using the scoring rule $\text{score}_j = \sum_{i=1}^{d_j} \rho'_j[i]\text{score}_{j,i}$. Suppose that on each run, we obtain the same final state $s_{\kappa,t+1}$. The utility function is *final-state-constant* if $u_{\boldsymbol{\rho}}(z) = u_{\boldsymbol{\rho}'}(z)$.

We provide a sample complexity bound for learning the parameters $\boldsymbol{\rho}$. The proof is a generalization of that of Theorem 4.3.21.

**Theorem 4.5.2.** *Let $d = \sum_{j=1}^{t} d_j$ denote the total number of tunable parameters of tree search. Then,*

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

In the context of integer programming, Theorem 4.5.2 not only recovers Theorem 4.3.21 for learning variable selection policies, but also yields a more general bound that simultaneously incorporates cutting plane selection, variable selection, and node selection. In branch-and-cut,



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

Figure 4.10: Illustration of scoring rules. In each figure, the blue region is the feasible region, the black dotted line is the cut in question, the blue solid line is orthogonal to the objective $\boldsymbol{c}$, the black dot is the LP optimal solution, and the white dot is the incumbent IP solution. Figure 4.10a illustrates efficacy, which is the length of the black solid line between the cut and the LP optimal solution. The cut in Figure 4.10b has better objective parallelism than the cut in Figure 4.10c. The cut in Figure 4.10d has a better directed cutoff distance than the cut in Figure 4.10e, but both have the same efficacy.

the first action of each round is to select a node. Since there are at most $2^{n+1} - 1$ nodes, $T_1 \leq 2^{n+1} - 1$. The second action is to choose a cutting plane. Let $\mathcal{C}$ be a finite family of cutting planes, such as the set of Gomory cutting planes. The last action is to choose a variable to branch on at that node, so $T_3 = n$. Applying Theorem 4.5.2, $\text{Pdim}(\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\}) = O(d\kappa n + d\kappa \log r + d \log d)$. Ignoring $T_1$ and $T_2$, thereby only learning the variable selection policy, recovers the $O(d\kappa \log n + d \log d)$ bound from Theorem 4.3.21.

### 4.5.1 Scoring rules for cutting plane selection

Cutting planes are a means of ensuring that at each iteration of branch-and-cut, the solution to the LP relaxation is as close to the optimal integral solution as possible. Formally, let $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$ denote the feasible region obtained by taking the LP relaxation of IP $\max\{\boldsymbol{c} \cdot \boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0, \boldsymbol{x} \in \mathbb{Z}^n\}$. Let $\mathcal{P}_I = \text{conv}(\mathcal{P} \cap \mathbb{Z}^n)$ denote the integer hull of $\mathcal{P}$. A *valid* cutting plane is any hyperplane $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \boldsymbol{\beta}$ such that if $\boldsymbol{x}$ is in the integer hull ($\boldsymbol{x} \in \mathcal{P}_I$), then $\boldsymbol{x}$ satisfies the inequality $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \boldsymbol{\beta}$. In other words, a valid cut does not remove any integral point from the LP relaxation's feasible region. A valid cutting plane *separates* $\boldsymbol{x} \in \mathcal{P} \setminus \mathcal{P}_I$ if it does not satisfy the inequality, or in other words, $\boldsymbol{\alpha}^T\boldsymbol{x} > \beta$. At any node of the search tree, branch-and-cut can add valid cutting planes that separate the optimal solution to the node's LP relaxation, thus improving the solution estimates used to prune the search tree. However, adding too many cuts will increase the time it takes to solve the LP relaxation at each node. Therefore, solvers such as SCIP [Gamrath et al., 2020], the leading open-source solver, bound the number of cuts that will be applied.

To give an example, a famous class of cutting planes is the family of *Chvátal-Gomory (CG) cuts*[8] [Chvátal, 1973, Gomory, 1958], which are parameterized by vectors $\boldsymbol{u} \in \mathbb{R}^m$. The CG cut defined by $\boldsymbol{u} \in \mathbb{R}^m$ is the hyperplane $\lfloor \boldsymbol{u}^T A \rfloor \boldsymbol{x} \leq \lfloor \boldsymbol{u}^T \boldsymbol{b} \rfloor$, which is guaranteed to be valid.

Some IP solvers such as SCIP use *scoring rules* to select among cutting planes, which are meant to measure the quality of a cut. Some commonly-used scoring rules include *efficacy* [Balas et al., 1996] ($\text{score}_1$), *objective parallelism* [Achterberg, 2007] ($\text{score}_2$), *directed cut-*

[8]The set of CG cuts is equivalent to the set of Gomory (fractional) cuts [Cornuéjols and Li, 2001], another commonly studied family of cutting planes with a slightly different parameterization.


(a) (b)

Figure 4.11: Illustrations of Example 4.6.1.

*off distance* [Gamrath et al., 2020] ($\text{score}_3$), and *integral support* [Wesselmann and Suhl, 2012] ($\text{score}_4$). *Efficacy* measures the distance between the cut $\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta$ and $\boldsymbol{x}^*_{\mathsf{LP}}$: $\text{score}_1(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = (\boldsymbol{\alpha}^T\boldsymbol{x}^*_{\mathsf{LP}} - \beta)/\left\|\boldsymbol{\alpha}\right\|_2$, as illustrated in Figure 4.10a. *Objective parallelism* measures the angle between the objective $\boldsymbol{c}$ and the cut's normal vector $\boldsymbol{\alpha}$: $\text{score}_2(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = \left|\boldsymbol{c}^T\boldsymbol{\alpha}\right| / (\left\|\boldsymbol{\alpha}\right\|_2 \left\|\boldsymbol{c}\right\|_2)$, as illustrated in Figures 4.10b and 4.10c. *Directed cutoff distance* measures the distance between the LP optimal solution and the cut in a more relevant direction than the efficacy scoring rule. Specifically, let $\overline{\boldsymbol{x}}$ be the *incumbent solution*, which is the best-known feasible solution to the input IP. The directed cutoff distance is the distance between the hyperplane $(\boldsymbol{\alpha}, \beta)$ and the current LP solution $\boldsymbol{x}^*_{\mathsf{LP}}$ along the direction of the incumbent $\overline{\boldsymbol{x}}$, as illustrated in Figures 4.10d and 4.10e: $\text{score}_3(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = \left\|\overline{\boldsymbol{x}} - \boldsymbol{x}^*_{\mathsf{LP}}\right\|_2 \cdot (\boldsymbol{\alpha}^T\boldsymbol{x}^*_{\mathsf{LP}} - \beta)/\left|\boldsymbol{\alpha}^T\left(\overline{\boldsymbol{x}} - \boldsymbol{x}^*_{\mathsf{LP}}\right)\right|$. Finally, to describe the *integral support* scoring rule, let $Z$ be the set of all indices $\ell \in [n]$ such that $\boldsymbol{\alpha}[\ell] \neq 0$. Let $\bar{Z}$ be the set of all indices $\ell \in Z$ such that the $\ell^{th}$ variable is constrained to be integral. This scoring rule is defined as $\text{score}_4(\boldsymbol{\alpha}^T\boldsymbol{x} \leq \beta) = \frac{|\bar{Z}|}{|Z|}$. Wesselmann and Suhl [2012] write that "one may argue that a cut having non-zero coefficients on many (possibly fractional) integer variables is preferable to a cut which consists mostly of continuous variables." SCIP uses the scoring rule $\frac{3}{5}\text{score}_1 + \frac{1}{10}\text{score}_2 + \frac{1}{2}\text{score}_3 + \frac{1}{10}\text{score}_4$ [Gamrath et al., 2020].

## 4.6 Constraint satisfaction problems

In this section, we describe tree search for constraint satisfaction problems. The generalization guarantee from Section 4.3.3 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 $\psi$ 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 4.6.1** (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 4.11a 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 $\psi$ 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 4.11b for a coloring that satisfies all constraints ($y_1 = y_4 =$ green, $y_2 =$ orange, and $y_3 =$ pink).

### 4.6.1 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.

### 4.6.2 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 [Bessiere and Régin, 1996]:** 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 [Haralick and Elliott, 1980]:** 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 5

# 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 thesis. 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 5.1.

Next, in Section 5.2, 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.

## 5.1 Profit maximization

The design of profit-maximizing[1] mechanisms is a fundamental problem with diverse applications including Internet retailing, advertising markets, strategic sourcing, and artwork sales. This problem has traditionally been studied under the assumption that there is a joint distribution from which the buyers' values are drawn and that the mechanism designer knows

[1] In this work, we study the standard setting of profit maximization with risk-neutral agents.

this distribution in advance. This assumption has led to groundbreaking theoretical results in the single-item setting [Myerson, 1981], but transitioning from theory to practice is challenging because the true distribution over buyers' values is typically unknown. Moreover, in the dramatically more challenging setting where there are multiple items for sale, the support of the distribution alone is often doubly exponential (even if there were just a single buyer with a finite type space[2]), so obtaining and storing the distribution is typically impossible.

We relax this strong assumption and instead assume that the mechanism designer only has a set of independent samples from the distribution, an approach introduced by Likhodedov and Sandholm [2004, 2005] and Sandholm and Likhodedov [2015]. Specifically, a single sample is a random draw from the distribution over buyers' values, listing each buyer's value for each set of items for sale. When the mechanism designer uses a set of samples rather than a description of the distribution to design a mechanism, we refer to this procedure as *sample-based mechanism design.* Sample-based mechanism design reflects current industry practices since many companies, such as online ad exchanges [He et al., 2014b, Muñoz Medina and Vassilvitskii, 2017], sponsored search platforms [Benisch et al., 2009, Edelman et al., 2007, Tang, 2017], travel companies [Yee and Ifrach, 2015], and resellers of returned items [Walsh et al., 2008], use historical purchase data to adjust the sales mechanism.

In this chapter, we present a general theory for deriving uniform convergence generalization guarantees in multi-item settings, as well as data-dependent guarantees when the distribution over buyers' values is well-behaved. In this setting, a generalization guarantee for a mechanism class $\mathcal{M}$ bounds the difference between the average profit over the samples and expected profit on the distribution for any mechanism in $\mathcal{M}$. Prior research has suggested a variety of optimization algorithms for mechanism classes we consider [Balcan et al., 2020d, Cai and Daskalakis, 2017a, Likhodedov and Sandholm, 2004, 2005, Sandholm and Likhodedov, 2015]. A mechanism designer can use our guarantees to ensure that for any mechanism in $\mathcal{M}$ he considers, average profit over the training set will be close to expected profit.

This chapter is part of a line of research that studies how learning theory can be used to design and analyze mechanisms, beginning with seminal research by Balcan et al. [2005, 2008]. The majority of these papers have studied only single-parameter settings [Alon et al., 2017, Bubeck et al., 2017, Cole and Roughgarden, 2014, Devanur et al., 2016, Elkind, 2007, Gonczarowski and Nisan, 2017, Guo et al., 2019, Hartline and Taggart, 2016, Huang et al., 2015, Mohri and Muñoz Medina, 2014, Morgenstern and Roughgarden, 2015, Roughgarden and Schrijvers, 2016]. In contrast, we focus on multi-item mechanism design, as have recent papers by Cai and Daskalakis [2017a], Morgenstern and Roughgarden [2016], Muñoz Medina and Vassilvitskii [2017], Syrgkanis [2017], and Gonczarowski and Weinberg [2018].

### 5.1.1 Our contributions

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

[2]When each buyer's values are independent from every other buyer's values, the number of support points is $nk^{2^m}$, where $n$ is the number of buyers, $k$ is the number of discrete value levels a buyer can assign to a bundle, and $m$ is the number of items. This is because each of the $2^m$ bundles can take any of $k$ values. With correlated valuations, the prior has $k^{2^{nm}}$ support points.

| Category | Mechanism class | Valuations | Result |
|---|---|---|---|
| Pricing mechanisms | Item-pricing mechanisms | General, unit-demand, additive | Lemmas 5.1.16, 5.1.27, A.0.7, A.0.8 |
| | Two-part tariffs | General | Lemma 5.1.10 |
| | Non-linear pricing mechanisms | General | Lemmas 5.1.13, 5.1.15 |
| Auctions | Second-price auctions with reserves | Additive | Lemmas 5.1.17, 5.1.26, A.0.9 |
| | Affine maximizer auctions | General | Lemma 5.1.19 |
| | Virtual valuation combinatorial auctions | General | Lemma 5.1.19 |
| | Mixed-bundling auctions with reserves | General | Lemma 5.1.18 |
| Randomized mechanisms | Lotteries | Additive, unit-demand | Lemmas 5.1.21, 5.1.28 |

Table 5.1: Brief summary of some of the main mechanism classes we analyze in Sections 5.1.4 and 5.1.5 as well as Appendix A.

| Valuations | Auction class | Our bounds | Prior bounds |
|---|---|---|---|
| Additive or unit-demand | Length-$\ell$ lottery menu | $H\sqrt{\ell m \log(\ell m)/N}$ | N/A |
| Additive, item-independent[1] | Length-$\ell$ item lottery menu | $H\sqrt{\ell \log \ell / N}$ | N/A |

[1] Additive cost function

Table 5.2: Generalization bounds in big-$\tilde{O}$ notation for lotteries. We denote the maximum profit achievable by any mechanism in the class over the support of the buyers' valuation distribution by $H$. There are $m$ items, $N$ samples, and the cost function is general unless otherwise noted.

**A general theory that unifies diverse mechanism classes.** We provide a clean, easy-to-use, general theorem for deriving generalization guarantees and we demonstrate its application to a large number of widely-used mechanism classes. This chapter thus expands our understanding of uniform convergence in multi-item mechanism design, which had thus far focused on deriving guarantees for a small number of specific mechanism classes that are "simple" by design [Morgenstern and Roughgarden, 2016, Syrgkanis, 2017]. We uncover a key structural property shared by a variety of mechanisms which allows us to prove 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 seemingly disparate mechanisms share this structure, and thus our main theorem yields learnability guarantees.

Table 5.1 summarizes some of the main mechanism classes we analyze and Tables 5.2, 5.3, and 5.4 summarize our bounds.

| Valuations | Mechanism class | Price class | Our bounds | Prior bounds |
|---|---|---|---|---|
| General | Length-$\ell$ menus of two-part tariffs over $\kappa$ units | Anonymous | $H\sqrt{\ell \log(\kappa n \ell)/N}$ | N/A |
| | | Non-anonymous | $H\sqrt{n\ell \log(\kappa n \ell)/N}$ | N/A |
| | Non-linear pricing | Anonymous | $H\sqrt{m \prod_{i=1}^{m}(\kappa_i+1)/N}$[3] | N/A |
| | | Non-anonymous | $H\sqrt{nm \prod_{i=1}^{m}(\kappa_i+1)/N}$[3] | N/A |
| | Additively decomposable non-linear pricing | Anonymous | $H\sqrt{m \sum_{i=1}^{m}\kappa_i/N}$[3] | N/A |
| | | Non-anonymous | $H\sqrt{nm \sum_{i=1}^{m}\kappa_i/N}$[3] | N/A |
| | Item-pricing | Anonymous | $H\sqrt{m^2/N}$ | $H\sqrt{m^2/N}$[4] |
| | | Non-anonymous | $H\sqrt{nm(m+\log n)/N}$ | $H\sqrt{nm^2 \log n/N}$[4] |
| Unit-demand | Item-pricing | Anonymous | $H\sqrt{m \cdot \min\{m, \log(nm)\}/N}$ | $H\sqrt{m^2/N}$[4] |
| | | Non-anonymous | $H\sqrt{nm \log(nm)/N}$ | $H\sqrt{nm^2 \log n/N}$[4] |
| Additive | Item-pricing | Anonymous | $H\sqrt{m \log m/N}$ | $H\sqrt{m \log m/N}$[4], $(H/\delta)\sqrt{m \log(nN)/N}$[2] |
| | | Non-anonymous | $H\sqrt{nm \log(nm)/N}$ | $H\sqrt{nm \log(nm)/N}$[4], $(H/\delta)\sqrt{nm \log(N)/N}$[2] |
| Additive, item-independent[1] | Item-pricing | Anonymous | $H\sqrt{1/N}$ | $H\sqrt{m \log m/N}$[4], $(H/\delta)\sqrt{m \log(nN)/N}$[2] |
| | | Non-anonymous | $H\sqrt{n \log n/N}$ | $H\sqrt{nm \log(nm)/N}$[4], $(H/\delta)\sqrt{nm \log(N)/N}$[2] |

[1] Additive cost function; [2] Syrgkanis [2017]. The probability these bounds fail to hold is $\delta$. In all other bounds, $\delta$ appears in a log so we suppress it using big-$\tilde{O}$ notation; [3] $\kappa_i$ is an upper bound on the number of units available of item $i$; [4] Morgenstern and Roughgarden [2016].

Table 5.3: Generalization bounds in big-$\tilde{O}$ notation for pricing mechanisms. We denote the maximum profit achievable by any mechanism in the class over the support of the buyers' valuation distribution by $H$. There are $m$ items, $n$ buyers, and $N$ samples. The cost function is general unless otherwise noted.

We prove that our main theorem applies to lotteries, a general representation of randomized mechanisms. Randomized mechanisms are known to generate higher expected revenue than deterministic mechanisms in many settings [Conitzer and Sandholm, 2003, Dobzinski and Dughmi, 2009]. Our results imply, for example, that if the mechanism designer plans to offer a menu of $\ell$ lotteries over $m$ items to an additive or unit-demand buyer, the difference between any such menu's average profit over $N$ samples and its expected profit is $\tilde{O}\left(H\sqrt{\ell m/N}\right)$, 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. Additionally, we study *multi-part tariffs*, where there is an upfront fee and a price per unit. These tariffs and other non-linear pricing mechanisms have been studied in economics for decades [Feldstein, 1972, Oi, 1971, Wilson, 1993]. For instance, our main theorem guarantees that if there are $\kappa$ units of a single good for sale, the difference between any two-part tariff's average profit over $N$ samples and its expected profit is $\tilde{O}\left(H\sqrt{\kappa/N}\right)$.

Our main theorem implies generalization bounds for many auction classes, such as *sec-*

| Valuations | Auction class | Our bounds | Prior bounds |
|---|---|---|---|
| General | AMAs and $\lambda$-auctions | $H\sqrt{n^{m+1}m\log n/N}$ | $cH\sqrt{m/N}n^{m+2}\left(n^2+\sqrt{n^m}\right)$[5][6] |
| | VVCAs | $H\sqrt{n^2m2^m\log n/N}$ | $cH\sqrt{m/N}n^{m+2}\left(n^2+\sqrt{n^m}\right)$[5][6] |
| | MBARPs | $H\sqrt{m(\log n+m)/N}$ | $H\sqrt{m^3\log n/N}$[5] |
| Additive | Second price item auctions with anonymous reserve prices | $H\sqrt{m\log m/N}$ | $H\sqrt{m\log m/N}$[4] |
| | Second price item auctions with non-anonymous reserve prices | $H\sqrt{nm\log(nm)/N}$ | $H\sqrt{nm\log(nm)/N}$[4] |
| Additive, item-independent[1] | Second price item auctions with anonymous reserve prices | $H\sqrt{1/N}$ | $H\sqrt{m\log m/N}$[4] |
| | Second price item auctions with non-anonymous reserve prices | $H\sqrt{n\log n/N}$ | $H\sqrt{nm\log(nm)/N}$[4] |

[1] Additive cost function; [4] Morgenstern and Roughgarden [2016]; [5] Balcan et al. [2016]; [6] The value of $c > 1$ depends on the range of the auction parameters;

Table 5.4: Generalization bounds in big-$\tilde{O}$ notation for auctions. We denote the maximum profit achievable by any mechanism in the class over the support of the buyers' valuation distribution by $H$. There are $m$ items, $n$ buyers, and $N$ samples. The cost function is general unless otherwise noted.

*ond price auctions*. We also study several well-studied generalized VCG auctions, such as *affine maximizer auctions, virtual valuations combinatorial auctions*, and *mixed-bundling auctions* [Dobzinski and Sundararajan, 2008, Jehiel et al., 2007, Lavi et al., 2003, Roberts, 1979, Sandholm and Likhodedov, 2015].

A key challenge which differentiates our generalization guarantees from those typically found in machine learning is the sensitivity of these mechanisms to small changes in their parameters. For example, changing the price of a good can cause a steep drop in profit if the buyer no longer wants to buy it. Meanwhile, for many well-understood function classes in machine learning, there is a close connection between the distance in parameter space between two parameter vectors and the distance in function space between the two corresponding functions. Understanding this connection is often the key to quantifying the class's intrinsic complexity. Intrinsic complexity can be quantified by pseudo-dimension, for example, which allows us to derive learnability guarantees. Since profit functions do not exhibit this predictable behavior, we must carefully analyze the structure of the mechanisms we study in order to derive our generalization guarantees.

**Data-dependent generalization guarantees for profit maximization.** We strengthen our main theorem when the distribution over buyers' values is "well-behaved," proving generalization guarantees that 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 buyers either. These guarantees hold when the buyers are additive with values drawn from *item-independent distributions* (buyer $i_1$'s value for item $j$ is independent from her value for item $j'$, but her value for item $j$ may be arbitrarily correlated with buyer $i_2$'s value for item $j$). Buyers with item-independent value distributions have been studied extensively in prior research [Babaioff et al., 2017, Cai and Daskalakis, 2017a,

Cai et al., 2016, Chawla et al., 2007, Goldner and Karlin, 2016, Hart and Nisan, 2012, Yao, 2014]. This could model buyers at, for example, antique auctions and art auctions (as long as there are no collections to try to assemble or the collections are sold as atomic lots).

Cai and Daskalakis [2017a] provide learning algorithms for buyers with valuations drawn from product distributions, which are item-independent. As we describe in Section 5.1.7, we improve a generalization bound by Cai and Daskalakis [2017a] and Morgenstern and Roughgarden [2016] for learning a mechanism with expected revenue that is nearly a constant fraction of optimal when the buyers have additive values drawn from a product distribution. The improvement is by a multiplicative factor of $O(\sqrt{m})$, where $m$ is 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.

### 5.1.2 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 [Conitzer and Sandholm, 2002, 2004, Sandholm, 2003]. 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 [Likhodedov and Sandholm, 2004, 2005, Sandholm and Likhodedov, 2015]. 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 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.

Balcan et al. [2005, 2008] were the first to study the connection between learning theory and revenue maximization, employing classic learning-theoretic tools such as *covering numbers*. They showed how to use an algorithm $\mathcal{A}$ that returns a high-revenue, manipulable mechanism in order to find a high-revenue, incentive-compatible mechanism. Their approach randomly splits the buyers into two groups. With the first group's bids as input, they use the algorithm $\mathcal{A}$ to find a high-revenue mechanism and they field that mechanism on the second group (and vice versa). While the mechanism is not incentive compatible for the first group, it is incentive

compatible for the second. They prove that the mechanism's revenue over the first group's bids nearly matches the revenue obtained from the second group, so long as the number of buyers is sufficiently large compared to the *covering number* of the mechanism class being optimized over. Balcan et al. [2005, 2008] study settings with unrestricted supply, whereas we primarily focus on settings with limited supply.

More recent research has provided generalization guarantees when there is limited supply, with a particular focus on single-parameter (e.g., single-item) settings [Alon et al., 2017, Bubeck et al., 2017, Chawla et al., 2014, Cole and Roughgarden, 2014, Devanur et al., 2016, Elkind, 2007, Gonczarowski and Nisan, 2017, Guo et al., 2019, Hartline and Taggart, 2016, Huang et al., 2015, Mohri and Muñoz Medina, 2014, Morgenstern and Roughgarden, 2015, Roughgarden and Schrijvers, 2016]. From an algorithmic perspective, Devanur et al. [2016], Gonczarowski and Nisan [2017], Guo et al. [2019], and Hartline and Taggart [2016] provide computationally efficient algorithms for learning nearly-optimal single-item auctions in various settings. In contrast, we study multi-parameter settings.

Balcan et al. [2014] drew on classic tools from learning theory to provide algorithms and generalization guarantees for a related problem in algorithmic game theory: learning agents' preferences. Specifically, they made connections to the concept of *generalized linear functions* from the structured prediction literature [e.g., Collins, 2000, which we discuss in Appendix A]. Their algorithms make use of past data describing the purchases of a utility-maximizing agent in order to predict the agent's future purchases.

Morgenstern and Roughgarden [2016] later used the same concept of generalized linear functions from the structured prediction literature to provide sample complexity guarantees for multi-item revenue maximization. They proposed a technique for bounding a mechanism class's pseudo-dimension that requires two steps; we describe these two steps at a high level here and refer the reader to Appendix A for more details. First, one must show that for any mechanism in the class, its allocation function is a $d$-dimensional linear function, for some $d \in \mathbb{Z}$ (see Appendix A for the formal definition). Next, fixing an arbitrary set of samples and an allocation per sample, one must bound the pseudo-dimension of the set of revenue functions across all mechanisms that induce those allocations. They show how these two steps imply a bound on the mechanism class's pseudo-dimension.

The guarantees presented in this chapter offer several advantages over Morgenstern and Roughgarden's approach. First, our main theorem depends on a structural property—the piecewise-linear form of the revenue function—that is not defined in terms of any learning theory concept (such as generalized linear functions or pseudo-dimension) and thus can be more readily applicable. Moreover, in several cases, Morgenstern and Roughgarden [2016] proved loose guarantees using structured prediction; in the appendix, they used a first-principles approach to prove stronger guarantees. Their structured prediction proof technique requires them to bound the total number of allocations a mechanism class can induce on a set of samples. Their bound is a bit loose, and we are able to tighten it using the techniques we develop in this chapter, as we detail in Appendix A. By combining our analysis techniques with tools from structured prediction, we are able to match the tighter bounds that Morgenstern and Roughgarden [2016] proved via a first-principals approach. This answers an open question posed by Morgenstern and Roughgarden [2016]. Finally, we apply our guarantees to a wide variety of mechanism classes, both simple and complex (as summarized in Table 5.1), whereas Morgenstern and Roughgarden [2016] applied their guarantees to three mechanism classes that are "simple" by design: item-pricing mechanisms, grand-bundle-pricing mechanisms (where an

agent can buy either the grand bundle consisting of all items or nothing at all), and second-price item auctions.

Syrgkanis [2017] also suggests a general technique for providing generalization guarantees which he applies to several "simple" mechanism classes: the same three as Morgenstern and Roughgarden [2016] as well as single-item *t-level auctions* [Morgenstern and Roughgarden, 2015]. His generalization guarantees apply only to *empirical revenue maximization* algorithms, which return the mechanism (from within a particular mechanism class) that maximizes average revenue over the samples. This is in contrast to our bounds (as well as those by Morgenstern and Roughgarden [2016]), which apply uniformly to every mechanism in a given class. This is crucial when it may not be computationally feasible to determine a mechanism with highest average revenue over the samples, but only an approximation. Given a set of samples $\mathcal{S}$ of size $N$, Syrgkanis's bound depends on a quantity he calls the *split-sample growth rate*, which counts how many different mechanisms the empirical revenue maximization algorithm can return on any sub-sample of $\mathcal{S}$ of size $N/2$. Another advantage of our generalization bounds (beyond applying uniformly to every mechanism in a given class) is that they grow only logarithmically with a term $\frac{1}{\delta}$, where $\delta$ is the probability that the bound fails to hold (as do those by Morgenstern and Roughgarden [2016]). In contrast, the bounds by Syrgkanis [2017] grow linearly in $\frac{1}{\delta}$.

In Section 5.1.7, we provide more details on how our results compare to those by Morgenstern and Roughgarden [2016] and Syrgkanis [2017], as well as a detailed comparison of our results to other papers on the sample complexity of multi-item revenue maximization [Balcan et al., 2016, Cai and Daskalakis, 2017a, Gonczarowski and Weinberg, 2018, Muñoz Medina and Vassilvitskii, 2017].

#### Dynamic mechanism design

A problem that is similar but distinct from ours is *dynamic pricing*, where prices are adjusted over a finite time horizon and the consumer demand function is unknown [e.g., Araman and Caldentey, 2009, Besbes and Zeevi, 2009, Broder and Rusmevichientong, 2012, Golrezaei et al., 2020, Kanoria and Nazerzadeh, 2020, all of whom study the single-item setting]. The goal is typically to minimize regret, which is the difference between the cumulative profit of the best prices in hindsight and that of the algorithm's selected prices.

#### Approximation guarantees for mechanism design

Although the revenue-maximizing multi-item mechanism remains elusive, a long line of research has shown that many of the mechanism classes we analyze can guarantee *approximately*-optimal revenue. Beginning with unit-demand buyers, Chawla et al. [2007] prove that under a single unit-demand buyer, item-pricing mechanisms can yield a constant fraction of optimal revenue. Given constraints on which allocations are feasible, Chawla et al. [2010] prove that under multiple unit-demand buyers, item-pricing mechanisms provide a constant fraction of optimal revenue.

Moving on to a single additive buyer with independent values, Hart and Nisan [2012, 2017] prove that item-pricing mechanisms provide a $O(\log^2 m)$ fraction of optimal revenue, an approximation ratio Li and Yao [2013] improve to $O(\log m)$, which is tight. Although even in this simple setting, item-pricing alone cannot always guarantee a constant fraction of optimal revenue, Babaioff et al. [2014] show that the *better* of selling items separately via an item-pricing

mechanism or selling the grand-bundle as a single unit can guarantee a constant fraction of optimal revenue. A constant factor approximation ratio also holds when the buyers' values exhibit specific types of correlations [Bateni et al., 2015], buyers with subadditive values [Rubinstein and Weinberg, 2015], and buyers with limited complementarities [Eden et al., 2021].

Finally, a line of research shows that in certain settings, lottery menus with a finite number of menu entries can provide a $(1-\epsilon)$-fraction of optimal revenue. For a single additive buyer with independent values, Babaioff et al. [2017, 2021] prove that a menu of length $(\log m/\epsilon)^{O(m)}$ suffices. Under a single unit-demand buyer, Kothari et al. [2019] introduce the notion of *symmetric menu complexity*, which is the number of menu entries the buyer may choose among, up to permutations of the items. They show that a quasi-polynomial symmetric menu complexity suffices to guarantee a $(1-\epsilon)$-fraction of optimal revenue.

### 5.1.3 Preliminaries and notation

We study the problem of selling $m$ heterogeneous items to $n$ buyers. We denote a bundle of items 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 items. 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 items 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 denoted as $c(Q)$. Suppose there are $\kappa_i$ units available of item $i$. Let $K = \prod_{i=1}^m (\kappa_i + 1)$. 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 use the notation $\mathcal{X}$ to denote the set of all valuation vectors $\boldsymbol{v}$. 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 dominant strategy incentive compatible, so we assume that the bids equal the buyers' valuations.

There is an unknown distribution $\mathcal{D}$ over buyers' values, which means the support of $\mathcal{D}$ is contained in the set $\mathcal{X}$. 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 buyer's values for multiple bundles may be correlated, and multiple buyers may have correlated values as well.

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 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 $\boldsymbol{v}$. For a distribution $\mathcal{D}$ over buyers' values, we denote the expected profit of the mechanism parameterized by $\boldsymbol{\rho}$ over $\mathcal{D}$ as $u_{\mathcal{D}}(\boldsymbol{\rho}) = \mathbb{E}_{\boldsymbol{v}\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right]$ and for a set of samples $\mathcal{S}$, we denote the average profit of the mechanism over $\mathcal{S}$ as $u_{\mathcal{S}}(\boldsymbol{\rho}) = \frac{1}{|\mathcal{S}|}\sum_{\boldsymbol{v}\in\mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v})$.

### 5.1.4 Worst-case generalization guarantees

Our guarantees apply to mechanism classes where for every valuation vector $\boldsymbol{v} \in \mathcal{X}$, the profit as a function of the parameters $\boldsymbol{\rho}$, denoted $u_{\boldsymbol{v}}(\boldsymbol{\rho})$, is piecewise linear. We begin by illustrating this property via several simple examples.

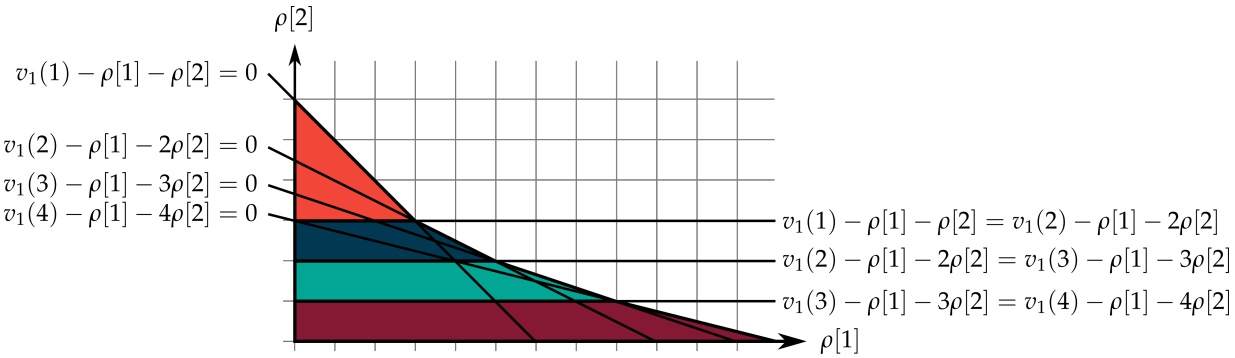



Figure 5.1: This figure illustrates the partition of the two-part tariff parameter space into piecewise-linear portions in the following scenario: there is one buyer whose value for one unit is $v_1(1) = 6$, value for two units is $v_1(2) = 9$, value for three units is $v_1(3) = 11$, and value for $i \geq 4$ 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 (the top region), and within, profit is linear in $\rho[1]$ and $\rho[2]$. By similar logic, the buyer will buy exactly two units in the blue region (the second-the-top region), exactly three units in the green region (the second-the-bottom region), and exactly four units in the red region (the bottom region), and profit is linear in $\rho[1]$ and $\rho[2]$ in each of these regions.

**Example 5.1.1** (Two-part tariffs)**.** In a *two-part tariff,* there are multiple units (i.e., copies) of a single item for sale. 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[3]. 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 [Feldstein, 1972, Oi, 1971, Wilson, 1993] and are prevalent throughout daily life. For example, health club membership programs often require an upfront membership fee plus a fee per month. Amusement parks often require an entrance fee with an additional payment per ride, as Oi [1971] analyzed in his paper "A Disneyland Dilemma." 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. Following the publication of the conference version of this paper [Balcan et al., 2018e], Balcan et al. [2020d] showed how to efficiently learn two-part tariffs that maximize average revenue over a training set, as well as *menus* of two-part tariffs, which we discuss in Section 5.1.4.

Revenue is a piecewise-linear function of the two-part tariff parameters $\rho[1]$ and $\rho[2]$ for the following reason. Suppose there are $\kappa$ units of the item 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 at most $\binom{\kappa+1}{2}$ 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 5.1 for an illustration.

[3]We generalize to multiple buyers later in this section.

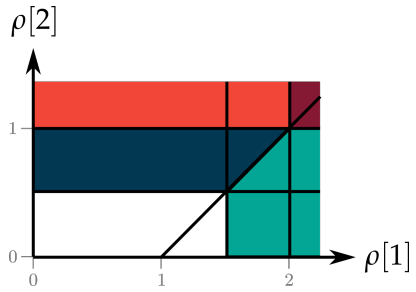



Figure 5.2: This figure illustrates the partition of the *item-pricing* parameter space into piecewise-linear portions in the following scenario: there are two items for sale and there are two buyers. Buyer 1's value for the first item is $v_1(1,0) = 2$, her value for the second item is $v_1(0,1) = 1$, and her value for both items is $v_1(1,1) = 2.5$. Buyer 2's values are $v_2(1,0) = 0, v_2(0,1) = 1$, and $v_2(1,1) = 1$. Suppose buyer 1 comes before buyer 2 in the ordering, which means that buyer 1 will first choose to buy the bundle of items that maximizes her utility and then buyer 2 will buy the bundle of remaining items that maximizes her utility. In the orange region, buyer 1 will buy item 1 because $v_1(1,0) - \rho[1] > v_1(0,1) - \rho[2]$, $v_1(1,0) - \rho[1] > v_1(1,1) - (\rho[1] + \rho[2])$, and $v_1(1,0) - \rho[1] > 0$. Buyer 2 will not buy anything because the only remaining item is item 2 and $v_2(0,1) - \rho[2] < 0$. By the same logic, in the red region, neither buyer will buy any item. In the blue region, buyer 1 will buy item 1 and buyer 2 will buy item 2. In the green region, buyer 1 will buy item 2 and buyer 2 will not buy anything. Finally, in the white region, buyer 1 will buy both items and buyer 2 will not buy anything.

**Example 5.1.2** (Item-pricing mechanisms)**.** Under an item-pricing mechanism, also known as a *posted-price mechanism*, there are multiple items for sale and multiple buyers. Unlike in Example 5.1.1, we assume there is a single unit of each item for sale. The mechanism designer sets a price per item and buyers buy their utility-maximizing bundles. These mechanisms are prevalent throughout the economics and computation literature (e.g., [Babaioff et al., 2014, Cai et al., 2016, Feldman et al., 2015]). In a bit more detail, an item-pricing mechanism can be defined by either *anonymous* or *non-anonymous* prices. Under anonymous prices, the seller sets a price $\rho[i]$ per item $i$. Under non-anonymous prices, there is a buyer-specific price per item. 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 items that maximizes his utility, then the next buyer in the ordering arrives and buys the bundle of remaining items that maximizes his utility, and so on.

Consider the simple case where the prices are anonymous[4]. For a given buyer $j$ and bundle pair $\boldsymbol{q}_1, \boldsymbol{q}_2 \in \{0,1\}^m$, buyer $j$ will prefer bundle $\boldsymbol{q}_1$ over bundle $\boldsymbol{q}_2$ so long as $v_j(\boldsymbol{q}_1) - \sum_{i:q_1[i]=1} \rho[i] > v_j(\boldsymbol{q}_2) - \sum_{i:q_2[i]=1} \rho[i]$. Therefore, for a fixed set of buyer values and for each buyer $j$, her preference ordering over the bundles is completely determined by the $\binom{2^m}{2}$ hyperplanes

$$\left\{ v_j(\boldsymbol{q}_1) - \sum_{i:q_1[i]=1} \rho[i] = v_j(\boldsymbol{q}_2) - \sum_{i:q_2[i]=1} \rho[i] : \boldsymbol{q}_1, \boldsymbol{q}_2 \in \{0,1\}^m \right\}.$$

Once the buyers' preference orderings are all fixed, the set of bundles they will each buy is also fixed. Furthermore, in any region of the price space where the bundles the buyers buy are

[4]We study non-anonymous prices as well later in this section.

fixed, profit is a linear function of the prices. See Figure 5.2 for an illustration.

We provide generalization guarantees that are closely dependent on the "complexity" of the partition splitting $\mathbb{R}^d$ into regions such that $u_v(\boldsymbol{\rho})$ is linear. Inspired by structure exhibited by many mechanism classes, such as Examples 5.1.1 and 5.1.2, 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 5.1.3** ($(d,t)$-delineable)**.** Fix a set of mechanisms defined parameters $\mathcal{P} \subseteq \mathbb{R}^d$ and let $\mathcal{U} = \{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\}$ 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 $u_v(\boldsymbol{\rho})$ is linear over $\mathcal{P}'$.

In Theorem 5.1.4, we relate pseudo-dimension to delineability. The proof is similar to that of Theorem 4.3.21 from Section 4.3.3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 5.1.4.** *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 5.1.4. We warm up with the classes from Examples 5.1.1 and 5.1.2.

**Lemma 5.1.5.** *Let $\mathcal{U} = \{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^2\}$ be the set of profit functions corresponding to the class of two-part tariffs over a single buyer and $\kappa$ units of a single good. The set $\mathcal{U}$ is $\left(2, \binom{\kappa+1}{2}\right)$-delineable.*

*Proof.* As we saw in Example 5.1.1, for any valuation vector $\boldsymbol{v}$, there are $\binom{\kappa+1}{2}$ hyperplanes splitting $\mathbb{R}^2$ into regions over which $u_v(\boldsymbol{\rho})$ is linear. □

**Lemma 5.1.6.** *Let $\mathcal{U} = \{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^{m+1}\}$ be the class of item-pricing mechanisms with anonymous prices. The set $\mathcal{U}$ is $\left(m, n\binom{2^m}{2}\right)$-delineable.*

*Proof.* This class's parameter space is $\mathbb{R}^m$, since there is one price per item. As we saw in Example 5.1.2, for any valuation vector $\boldsymbol{v}$, there are $n\binom{2^m}{2}$ hyperplanes splitting $\mathbb{R}^m$ into regions such that within any one region, $u_v(\boldsymbol{\rho})$ is linear. □

**Non-linear pricing mechanisms.**

Non-linear pricing mechanisms are specifically used to sell multiple units (i.e., copies) of a set of items. We discuss *two-part tariffs* and *general non-linear pricing mechanisms.* We make the following natural assumption which informally states that as the number of units in an allocation grows, the cost of that allocation will exceed the buyers' welfare. This implies delineability. Our assumption is formalized as follows.

**Assumption 5.1.7.** *There is some cap $\kappa_i$ per item $i$ such that it costs more to produce $\kappa_i$ units of item $i$ than the buyers will pay. In other words, there exists $(\kappa_1, \ldots, \kappa_m) \in \mathbb{R}^m$ such that for all $\boldsymbol{v}$ in the support of the distribution $\mathcal{D}$ over buyers' values 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 $\sum_{j=1}^n v_j(\boldsymbol{q}_j) - c(Q) < 0$.*

For example, suppose there are multiple units of a single item for sale and a single buyer whose value for $\tau$ units is always at most $8 + \tau$. Moreover, suppose the cost of producing $\tau$ units is $3\tau$. Then $\kappa_1 = 4$ because for $\tau \geq 5$, $8 + \tau < 3\tau$.

In the remainder of this section, we describe why this assumption implies $(d,t)$-delineability with $d < \infty$ and $t < \infty$.

**Menus of two-part tariffs.** Menus of two-part tariffs are a generalization of Example 5.1.1. At a high level, the seller offers the buyers $\ell$ different two-part tariffs and each buyer simultaneously chooses the tariff and number of units that maximizes his utility. Menus of two-part tariffs are common throughout daily life: consumers often choose among various membership tiers—typically with a larger upfront fee in exchange for lower future payments—for health clubs, wholesale stores like Costco, amusement parks, credit cards, and cellphone plans.

More formally, in the case of non-anonymous prices, let $\left(\rho_{1,j}^{(1)}, \rho_{2,j}^{(1)}\right), \ldots, \left(\rho_{1,j}^{(\ell)}, \rho_{2,j}^{(\ell)}\right)$ be the menu of two-part tariffs that the seller offers to buyer $j$. Here, $\rho_{1,j}^{(i)}$ is the upfront fee of the $i^{th}$ tariff and $\rho_{2,j}^{(i)}$ is the price per unit. If the prices are anonymous, then for each tariff $i$, $\rho_{1,1}^{(i)} = \cdots = \rho_{1,n}^{(i)}$ and $\rho_{2,1}^{(i)} = \cdots = \rho_{2,n}^{(i)}$. We assume each buyer simultaneously[5] chooses the tariff and the number of units maximizing his utility. In this context, an allocation is simply a vector $(q_1, \ldots, q_n)$ where $q_j \in \mathbb{Z}_{\geq 0}$ is the number of units buyer $j$ chooses to buy. If buyer $j$ chooses the tariff $t_j \in [\ell]$ and chooses to buy $q_j \geq 1$ units, he will pay $\rho_{1,j}^{(t_j)} + \rho_{2,j}^{(t_j)} \cdot q_j$. Thus, the seller's profit is

$$\sum_{j=1}^{n} \rho_{1,j}^{(t_j)} \cdot \mathbf{1}_{\{q_j \geq 1\}} + \rho_{2,j}^{(t_j)} \cdot q_j - c\left(Q\right)$$

where $Q = (q_1, \ldots, q_n)$. Note that this set of mechanisms is parameterized by vectors in $\mathbb{R}^{2n\ell}$ if there are non-anonymous prices and $\mathbb{R}^{2\ell}$ if the prices are anonymous.

Assumption 5.1.7 states that there is some cap $\kappa \in \mathbb{R}$ such that for all $\boldsymbol{v}$ in the support of the distribution $\mathcal{D}$ over buyers' values and all allocations $Q = (q_1, \ldots, q_n)$, if $\sum_{j=1}^{n} q_j > \kappa$, then $\sum_{j=1}^{n} v_j\left(q_j\right) - c\left(Q\right) < 0$. We also make the natural assumption that the mechanism designer will not choose prices that cause the seller to have negative utility. In other words, we assume he will select a *profit non-negative menu of two-part tariffs*, formalized as follows.

**Definition 5.1.8.** In the case of anonymous prices (respectively, non-anonymous), let $\mathcal{P} \subseteq \mathbb{R}^{2\ell}$ (respectively, $\mathcal{P}' \subseteq \mathbb{R}^{2n\ell}$) be the set of prices where no matter which tariff each buyer chooses and no matter how many units he buys, the seller will obtain non-negative utility. In other words, let $\mathcal{P}$ (respectively, $\mathcal{P}'$) be the set of mechanism parameters such that for each buyer $j \in [n]$, each tariff $t_j \in [\ell]$, and each allocation $Q = (q_1, \ldots, q_n)$, the seller's utility is non-negative:

$$\sum_{j=1}^{n} \rho_{1,j}^{(t_j)} \cdot \mathbf{1}_{\{q_j \geq 1\}} + \rho_{2,j}^{(t_j)} \cdot q_j - c\left(Q\right) \geq 0.$$

The set of *profit non-negative menus of two-part tariffs* is defined by parameters in $\mathcal{P}$ (respectively, $\mathcal{P}'$).

Under Assumption 5.1.7, we are guaranteed that no matter which profit non-negative parameters the mechanism designer chooses in $\mathcal{P}$ or $\mathcal{P}'$, if all buyers simultaneously choose the tariff and the number of units $(q_1, \ldots, q_n)$ that maximize their utilities, then $\sum_{j=1}^{n} q_j \leq \kappa$, as we prove in the following lemma.

**Lemma 5.1.9.** *No matter which parameters the mechanism designer chooses in $\mathcal{P}$ or $\mathcal{P}'$, if all buyers simultaneously choose the tariff and the number of units $(q_1, \ldots, q_n)$ that maximize their utilities, then $\sum_{j=1}^{n} q_j \leq \kappa$.*

[5]The fact that the buyers arrive simultaneously and that there are multiple units of each item for sale differentiates this setting from that of item-pricing, which we describe in Example 5.1.2 and later in this section.

*Proof.* We prove this lemma for non-anonymous prices, and the lemma for anonymous prices follow since they are a special case of non-anonymous prices. For a contradiction, suppose there exists a set of buyers' values $\boldsymbol{v}$ and a non-anonymous menu of two-part tariffs with parameters in $\mathcal{P}'$ such that if $t_j$ is the tariff that buyer $j$ chooses and $q_j$ is the number of units he chooses, $\sum_{j=1}^{n} q_j > \kappa$. Since the mechanisms are profit non-negative, we know that $\sum_{j=1}^{n} \rho_{1,j}^{(t_j)} \cdot \mathbf{1}_{\{q_j \geq 1\}} + \rho_{2,j}^{(t_j)} \cdot q_j - c\left(Q\right) \geq 0$, where $Q = (q_1, \ldots, q_n)$. We also know that each buyer's value for the units he bought is greater than the price: $\sum_{j=1}^{n} v_j(q_j) \geq \sum_{j=1}^{n} \rho_{1,j}^{(t_j)} \cdot \mathbf{1}_{\{q_j \geq 1\}} + \rho_{2,j}^{(t_j)} \cdot q_j$. Therefore, $\sum_{j=1}^{n} v_j(q_j) - c(Q) \geq 0$. However, this contradicts Assumption 5.1.7, so the lemma holds. □

This fact is crucial to proving delineability since the number of hyperplanes splitting the parameter space into regions where profit is linear depends on $\kappa$.

**Lemma 5.1.10.** *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 profit non-negative length-$\ell$ menus of two-part tariffs. The set $\mathcal{U}$ is $\left(2\ell, O\left(n\left(\kappa\ell\right)^2\right)\right)$-delineable. When the prices are non-anonymous, the set is $\left(2n\ell, O\left(n\left(\kappa\ell\right)^2\right)\right)$-delineable.*

*Proof.* A length-$\ell$ menu of two-part tariffs is defined by $2\ell$ parameters. The first 2 parameters (denoted $\left(\rho_0^{(1)}, \rho_1^{(1)}\right)$) define the first tariff in the menu, the second 2 parameters (denoted $\left(\rho_0^{(2)}, \rho_1^{(2)}\right)$) define the second tariff in the menu, and so on. Buyer $j$ will prefer to buy $q \geq 1$ units using $i^{th}$ menu entry (defined by the parameters $\left(\rho_0^{(i)}, \rho_1^{(i)}\right)$) so long as $v_j(q) - \left(\rho_0^{(i)} + \rho_1^{(i)} q\right) > v_j(q') - \left(\rho_0^{(i')} + \rho_1^{(i')} q'\right)$ for any $i' \neq i$ and $q' \neq q$. In total, these inequalities define $O\left(n\left(\kappa\ell\right)^2\right)$ hyperplanes in $\mathbb{R}^{2\ell}$. In any region defined by these hyperplanes, the menu entries and quantities demanded by all $n$ buyers are fixed. In any such region, profit is linear in the fixed fees and unit prices.

In the case of non-anonymous reserve prices, the same argument holds, except that every length-$\ell$ menu of two-part tariffs is defined by $2n\ell$ parameters: for each buyer, we must set the fixed fee and unit price for each of the $\ell$ menu entries. □

**General non-linear pricing mechanisms.** We study general non-linear pricing mechanisms under Wilson's *bundling interpretation* [Wilson, 1993]: if the prices are anonymous, there is a price per quantity vector $\boldsymbol{q}$ denoted $\rho\left(\mathbf{q}\right)$. The buyers simultaneously choose the bundles maximizing their utility: buyer $j$ will purchase the bundle that maximizes $v_j\left(\mathbf{q}\right) - \rho\left(\mathbf{q}\right)$. If the prices are non-anonymous, there is a price per quantity vector $\boldsymbol{q}$ and buyer $j \in [n]$ denoted $\rho_j\left(\mathbf{q}\right)$. These general non-linear pricing mechanisms include *multi-part tariffs* as a special case.

Without any assumptions, the parameter space of this mechanism class has an infinite number of dimensions, since the mechanism designer, in theory, could choose prices for each and every bundle $\boldsymbol{q} \in \mathbb{Z}_{\geq 0}^{m}$. In Lemma 5.1.12, we show that under Assumption 5.1.7, no buyer will ever choose a bundle $\boldsymbol{q}$ such that $q[i] > \kappa_i$ for any $i \in [m]$. This holds so long as the mechanism designer chooses a *profit non-negative non-linear pricing mechanism* (see Definition 5.1.11, which is similar to Definition 5.1.8). Thus, the seller only needs to carefully set the prices of the bundles $\boldsymbol{q}$ such that $q[i] \leq \kappa_i$ for all $i \in [m]$. Letting $K = \prod_{i=1}^{m}(\kappa_i + 1)$ be the number of such bundles, the set of anonymous prices the mechanism designer needs to carefully set is a subset

of $\mathbb{R}^K$ and the set of non-anonymous prices is a subset of $\mathbb{R}^{nK}$. We now provide the definition of profit non-negative non-linear pricing mechanisms.

**Definition 5.1.11.** In the case of anonymous prices (respectively, non-anonymous), let $\mathcal{P}$ (respectively, $\mathcal{P}'$) be the set of mechanism parameters such that for each buyer $j \in [n]$ and each allocation $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$, the seller's utility is non-negative:

$$\sum_{j=1}^{n} \rho_j(\boldsymbol{q}_j) - c\left(Q\right) \geq 0.$$

The set of profit non-negative non-linear pricing mechanisms is defined by parameters in $\mathcal{P}$ (respectively, $\mathcal{P}'$).

Under Assumption 5.1.7, we are guaranteed that no matter which profit non-negative parameters the mechanism designer chooses in $\mathcal{P}$ or $\mathcal{P}'$, if all buyers simultaneously choose the bundles that maximize their utilities, then $\sum_{j=1}^{n} q_j[i] \leq \kappa_i$ for all $i \in [m]$, as we show in the following lemma.

**Lemma 5.1.12.** *No matter which parameters the mechanism designer chooses in $\mathcal{P}$ or $\mathcal{P}'$, if all buyers simultaneously choose the bundles that maximize their utilities, then $\sum_{j=1}^{n} q_j[i] \leq \kappa_i$ for all $i \in [m]$.*

*Proof.* We prove this lemma for non-anonymous prices, and the lemma for anonymous prices follow since they are a special case of non-anonymous prices. For a contradiction, suppose there exists a set of buyers' values $\boldsymbol{v}$ and a non-anonymous non-linear pricing mechanism with parameters in $\mathcal{P}'$ such that if $\boldsymbol{q}_j$ is the bundle buyer $j$ chooses, $\sum_{j=1}^{n} q_j[i] > \kappa_i$ for some $i \in [m]$. Since the mechanisms are profit non-negative, we know that $\sum_{j=1}^{n} \rho_j(\boldsymbol{q}_j) - c\left(Q\right) \geq 0$, where $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$. We also know that each buyer's value for the units he bought is greater than the price: $\sum_{j=1}^{n} v_j(\boldsymbol{q}_j) \geq \sum_{j=1}^{n} \rho_j(\boldsymbol{q}_j)$. Therefore, $\sum_{j=1}^{n} v_j(\boldsymbol{q}_j) - c(Q) \geq 0$. However, this contradicts Assumption 5.1.7, so the lemma holds. □

As in the case of two-part tariffs, this fact is crucial to proving delineability since the number of hyperplanes splitting the parameter space into regions where profit is linear depends on $\kappa_1, \ldots, \kappa_m$. Moreover, under these assumptions, the set of mechanism parameters is effectively $K$-dimensional in the case of anonymous prices and $nK$-dimensional in the case of non-anonymous prices: without loss of generality, the mechanism designer might as well set the price of any bundle $\boldsymbol{q}$ such that $q[i] \geq \kappa_i$ for some $i \in [m]$ to $\infty$.

**Lemma 5.1.13.** *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.*

*Proof.* We begin by analyzing the case where there are anonymous prices. By Lemma 5.1.12, the mechanism designer might as well set the price of any bundle $\boldsymbol{q}$ such that $q[i] \geq \kappa_i$ for some $i \in [m]$ to $\infty$. Therefore, every non-linear pricing mechanism is defined by $d = \prod_{i=1}^{m} (\kappa_i + 1)$ parameters because that is the number of different bundles and there is a price per bundle. Buyer $j$ will prefer the bundle corresponding to the quantity vector $\boldsymbol{q}$ over the bundle corresponding to the quantity vector $\boldsymbol{q}'$ if $v_j(\boldsymbol{q}) - \rho(\boldsymbol{q}) \geq v_j(\boldsymbol{q}') - \rho(\boldsymbol{q}')$. Therefore, there are at most $\prod_{i=1}^{m} (\kappa_i + 1)^2$ hyperplanes in $\mathbb{R}^d$ determining each buyer's preferred bundle — one hyperplane per pair of bundles. This means that there are a total of $n \prod_{i=1}^{m} (\kappa_i + 1)^2$ hyperplanes in $\mathbb{R}^d$ such

that in any one region induced by these hyperplanes, the bundles demanded by all $n$ buyers are fixed and profit is linear in the prices of these $n$ bundles.

In the case of non-anonymous prices, the same argument holds, except that every non-linear pricing mechanism is defined by $n\prod_{i=1}^{m}(\kappa_i+1)$ parameters — one parameter per bundle-buyer pair. □

We prove polynomial bounds when prices are additive over items.

**Definition 5.1.14** (Additively decomposable non-linear pricing mechanisms)**.** Additively decomposable non-linear pricing mechanisms are a subset of non-linear pricing mechanisms where the prices are additive over the items. Specifically, if the prices are anonymous, there exist $m$ functions $\rho^{(i)} : [\kappa_i] \to \mathbb{R}$ for all $i \in [m]$ such that for every quantity vector $\boldsymbol{q}$, $\rho(\boldsymbol{q}) = \sum_{i:q[i]\geq 1}\rho^{(i)}(q[i])$. If the prices are non-anonymous, there exist $nm$ functions $\rho_j^{(i)} : [\kappa_i] \to \mathbb{R}$ for all $i \in [m]$ and $j \in [n]$ such that for every quantity vector $\boldsymbol{q}$, $\rho_j(\boldsymbol{q}) = \sum_{i:q[i]\geq 1}\rho_j^{(i)}(q[i])$.

**Lemma 5.1.15.** *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of additively decomposable non-linear pricing mechanisms with anonymous prices. Then $\mathcal{U}$ is $\left(\sum_{i=1}^{m}(\kappa_i+1), n\prod_{i=1}^{m}(\kappa_i+1)^2\right)$-delineable. When the prices are non-anonymous, the class is $\left(n\sum_{i=1}^{m}(\kappa_i+1), n\prod_{i=1}^{m}(\kappa_i+1)^2\right)$-delineable.*

*Proof.* In the case of anonymous prices, any additively decomposable non-linear pricing mechanism is defined by $d = \sum_{i=1}^{m}(\kappa_i+1)$ parameters. As in the proof of Lemma 5.1.13, there are a total of $n\prod_{i=1}^{m}(\kappa_i+1)^2$ hyperplanes in $\mathbb{R}^d$ such that in any one region induced by these hyperplanes, the bundles demanded by all $n$ buyers are fixed and profit is linear in the prices of these $n$ bundles.

In the case of non-anonymous prices, the same argument holds, except that every non-linear pricing mechanism is defined by $n\sum_{i=1}^{m}(\kappa_i+1)$ parameters — one parameter per item, quantity, and buyer tuple. □

**Item-pricing mechanisms.**

We now apply Theorem 5.1.4 to anonymous and non-anonymous item-pricing mechanisms. Unlike non-linear pricing mechanisms, in this setting, there is only a single unit of each item for sale. Under anonymous prices, the seller sets a price per item. Under non-anonymous prices, there is a buyer-specific price per item. Since there is only one unit of each item for sale, we cannot assume the buyers arrive simultaneously and buy the bundle of goods maximizing their utilities, as we did when studying non-linear pricing. After all, this may lead to over-demand. Rather, 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 items that maximizes his utility, then the next buyer in the ordering arrives and buys the bundle of remaining items that maximizes his utility, and so on. This assumption is prevalent throughout the economics and computation literature (e.g., [Babaioff et al., 2014, Cai et al., 2016, Feldman et al., 2015]).

**Lemma 5.1.16.** *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous item-pricing mechanisms with anonymous prices and additive[6] buyers. The set $\mathcal{U}$ is $(m, m)$-delineable. When the*

[6]In the two cases where the buyers have unit-demand and general valuations, we prove generalization bounds by connecting the hyperplane structure we investigate in this chapter to the structured prediction literature in machine learning. See Appendix A for details.

*prices are non-anonymous, the set is is $(nm, nm)$-delineable.*

*Proof.* In the case of anonymous prices, every item-pricing mechanisms is defined by $m$ prices $\boldsymbol{p} \in \mathbb{R}^m$, so the parameter space is $\mathbb{R}^m$. Let $j_i$ be the buyer with the highest value for item $i$. We know that item $i$ will be bought so long as $v_{j_i}(\boldsymbol{e}_i) \geq \rho(\boldsymbol{e}_i)$. Once the items bought are fixed, profit is linear. Therefore, there are $m$ hyperplanes splitting $\mathbb{R}^m$ into regions where profit is linear.

In the case of non-anonymous prices, the parameter space is $\mathbb{R}^{nm}$ since there is a price per buyer and per item. The items each buyer $j$ is willing to buy is defined by $m$ hyperplanes: $v_j(\boldsymbol{e}_i) \geq \rho_j(\boldsymbol{e}_i)$. So long as these preferences are fixed, profit is a linear function of the prices. Therefore, there are $nm$ hyperplanes splitting $\mathbb{R}^{nm}$ into regions where profit is linear. □

In Appendix A, we connect the hyperplane structure we investigate in this chapter to the structured prediction literature in machine learning (e.g., [Collins, 2000]), thus proving even stronger generalization bounds for item-pricing mechanisms under buyers with unit-demand and general valuations and answering an open question by Morgenstern and Roughgarden [2016]. Balcan et al. [2014] were the first to explore the connection between structured prediction and mechanism design, though in a different setting from us: they provided algorithms that make use of past data describing the purchases of a utility-maximizing agent to produce a hypothesis function that can accurately forecast the future behavior of the agent.

In Section 5.1.7, we compare these results with those from prior research [Morgenstern and Roughgarden, 2016, Syrgkanis, 2017].

**Auctions.**

We now present applications of Theorem 5.1.4 to auctions. In each setting, there is a single unit of each item for sale.

**Second price item auctions with item reserves.** These auctions are only incentive compatible for additive buyers, 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 buyer $j$. The buyers submit bids on the items. For each item $i$, the highest bidder $j$ wins the item if and only 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 $i$.

**Lemma 5.1.17.** *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.*

*Proof.* For a given valuation vector $\boldsymbol{v}$, let $j_i$ be the highest bidder for item $i$ and let $j_i'$ be the second highest bidder. Under anonymous prices, item $i$ will be bought so long as $v_{j_i}(\boldsymbol{e}_i) \geq \rho(\boldsymbol{e}_i)$. If buyer $j_i$ buys item $i$, his payment depends on whether or not $v_{j_i'}(\boldsymbol{e}_i) \geq \rho(\boldsymbol{e}_i)$. Therefore, there are $t = 2m$ hyperplanes splitting $\mathbb{R}^m$ into regions where profit is linear. In the case of non-anonymous prices, the only difference is that the parameter space is $\mathbb{R}^{nm}$. □

In Section 5.1.7, we provide a detailed comparison of this lemma with results from prior research [Devanur et al., 2016, Morgenstern and Roughgarden, 2016, Syrgkanis, 2017], in both the single- and multi-item setting.

**Mixed bundling auctions with reserve prices (MBARPs).** MBARPs [Jehiel et al., 2007, Tang and Sandholm, 2012a] 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 buyer 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 buyer receives the grand bundle, and then runs the VCG mechanism over this enlarged set of buyers. 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 $\lambda$ be a function such that $\lambda(Q) = \gamma$ if some buyer receives the grand bundle under allocation $Q$ and o otherwise. For an allocation $Q$, let $\boldsymbol{q}_Q$ be the items not allocated. Given a valuation vector $\boldsymbol{v}$, the MBARP allocation is

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

Using the notation

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

buyer $j$ pays

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

**Lemma 5.1.18.** *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.*

*Proof.* An MBARP is defined by $m+1$ parameters since there is one reserve per item and one allocation boost. Let $K = (n+1)^m$ be the total number of allocations. Fix some valuation vector $\boldsymbol{v}$. We claim that the allocation of any MBARP is determined by at most $(n+1)K^2$ hyperplanes in $\mathbb{R}^{m+1}$. To see why this is, let $Q^k = (\boldsymbol{q}_1^k, \ldots, \boldsymbol{q}_n^k)$ and $Q^\ell = (\boldsymbol{q}_1^\ell, \ldots, \boldsymbol{q}_n^\ell)$ be any two allocations and let $\boldsymbol{q}_{Q^k}$ and $\boldsymbol{q}_{Q^\ell}$ be the bundles of items not allocated. Consider the $\binom{K}{2}$ hyperplanes defined as

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

In the intersection of these $\binom{K}{2}$ hyperplanes, the allocation of the MBARP is fixed.

By a similar argument, it is straightforward to see that $K^2$ hyperplanes determine the allocation of any MBARP in this restricted space without any one bidder's participation. This

leads us to a total of $(n+1)K^2$ hyperplanes which partition the space of MBARP parameters in a way such that for any two parameter vectors in the same region, the auction allocations are the same, as are the allocations without any one bidder's participation. Once these allocations are fixed, profit is a linear function in this parameter space. □

In Table 5.4, we compare this lemma with results from prior research [Balcan et al., 2016].

**Affine maximizer auctions (AMAs).** AMAs are an expressive mechanism class: Roberts [1979] proved that AMAs are the only *ex post* truthful mechanisms over unrestricted value domains. Later, Lavi et al. [2003] proved that under natural assumptions, every truthful multi-item auction is an "almost" AMA, that is, an AMA for sufficiently high values.[7] An AMA is defined by a weight per buyer $w_j \in \mathbb{R}_{>0}$ and a boost per allocation $\lambda(Q) \in \mathbb{R}_{\geq 0}$. By increasing any $w_j$ or $\lambda(Q)$, the seller can increase buyer $j$'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., $Q^* = (\boldsymbol{q}_1^*, \ldots, \boldsymbol{q}_n^*) = \text{argmax}\left\{\sum_{j=1}^n w_j v_j(\boldsymbol{q}_j) + \lambda(Q) - c(Q)\right\}$. The payments have the same form as the VCG payments, with the parameters factored in to ensure truthfulness. Formally, using the notation

$$Q^{-j} = \left(\boldsymbol{q}_1^{-j}, \ldots, \boldsymbol{q}_n^{-j}\right) = \text{argmax}\left\{\sum_{\ell \neq j} w_\ell v_\ell(\boldsymbol{q}_\ell) + \lambda(Q) - c(Q)\right\},$$

each buyer $j$ pays

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

A virtual valuation combinational auction (VVCA) [Likhodedov and Sandholm, 2004] is an AMA where each $\lambda(Q)$ is split into $n$ terms such that $\lambda(Q) = \sum_{j=1}^n \lambda_j(Q)$ where $\lambda_j(Q) = c_{j,\boldsymbol{q}}$ for all allocations $Q$ that give buyer $j$ exactly bundle $\boldsymbol{q}$. Finally, $\lambda$-auctions [Jehiel et al., 2007] are a special case of AMAs where the buyer weights equal 1.

**Lemma 5.1.19.** *Let $\mathcal{U}$, $\mathcal{U}'$, and $\mathcal{U}''$ be the sets of profit functions corresponding to the classes of AMAs, VVCAs, and $\lambda$-auctions, respectively. The set $\mathcal{U}$ is $\left(2n(n+1) + (n+1)^{m+1}, (n+1)^{2m+1}\right)$-delineable, $\mathcal{U}'$ is $\left(n2^m(3+2n), (n+1)^{2m+1}\right)$-delineable, and $\mathcal{U}''$ is $\left((n+1)^m, (n+1)^{2m+1}\right)$-delineable.*

*Proof.* Let $K = (n+1)^m$ be the total number of allocations and let $\boldsymbol{p}$ be a parameter vector where the first $n$ components correspond to the bidder weights $w_j$ for $j \in [n]$, the next $n$ components correspond to $1/w_j$ for $j \in [n]$, the next $2\binom{n}{2}$ components correspond to $w_i/w_j$ for all $i \neq j$, the next $K$ components correspond to $\lambda(Q)$ for every allocation $Q$, and the final $nK$ components correspond to $\lambda(Q)/w_j$ for all allocations $Q$ and all bidders $j \in [n]$. In total, the dimension of this parameter space is at most $2n + 2n^2 + K + nK = O(nK)$. Let $\boldsymbol{v}$ be a valuation

[7] Surprisingly, even when the buyers have additive values, AMAs can generate higher revenue than running a separate Myerson auction for each item [Sandholm and Likhodedov, 2015], even though the buyers' values do not exhibit any complementarity or substitutability. While this may seem surprising, it is to be expected due to prior research in bundle pricing in catalog offers [Adams and Yellen, 1976, Bakos and Brynjolfsson, 1999, McAfee et al., 1989].

vector. We claim that this parameter space can be partitioned using $t = (n+1)K^2$ hyperplanes into regions where in any one region $\mathcal{P}'$, there exists a vector $\boldsymbol{k}$ such that $u_v(\boldsymbol{p}) = \boldsymbol{k} \cdot \boldsymbol{p}$ for all $\boldsymbol{p} \in \mathcal{P}'$.

To this end, an allocation $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$ will be the allocation of the AMA so long as $\sum_{i=1}^n w_i v_i(\boldsymbol{q}_i) + \lambda(Q) - c(Q) \geq \sum_{i=1}^n w_i v_i(\boldsymbol{q}'_i) + \lambda(Q') - c(Q')$ for all $Q' = (\boldsymbol{q}'_1, \ldots, \boldsymbol{q}'_n) \neq Q$. Since the number of different allocations is at most $K$, the allocation of the auction on $\boldsymbol{v}$ is defined by at most $K^2$ hyperplanes in $\mathbb{R}^d$. Similarly, the allocations $Q^{-1}, \ldots, Q^{-n}$ are also determined by at most $K^2$ hyperplanes in $\mathbb{R}^d$. Once these allocations are fixed, profit is a linear function of this parameter space.

The proof for VVCAs follows the same argument except that we redefine the parameter space to consist of vectors where the first $n$ components correspond to the bidder weights $w_j$ for $j \in [n]$, the next $n$ components correspond to $1/w_j$ for $j \in [n]$, the next $2\binom{n}{2}$ components correspond to $w_i/w_j$ for all $i \neq j$, the next $K' = n2^m$ components correspond to the bidder-specific bundle boosts $c_{j,\boldsymbol{q}}$ for every quantity vector $\boldsymbol{q}$ and bidder $j \in [n]$, and the final $nK'$ components correspond to $c_{k,\boldsymbol{q}}/w_j$ for every quantity vector $\boldsymbol{q}$ and every pair of bidders $j, k \in [n]$. The dimension of this parameter space is at most $2n + 2n^2 + K' + nK' \leq 2K' + nK' + K' + nK' = O(nK')$.

Finally, the proof for $\lambda$-auctions follows the same argument as the proof for AMAs except there are zero bidder weights. Therefore, the parameter space consists of vectors with $K$ components corresponding to $\lambda(Q)$ for every allocation $Q$. □

In Table 5.4, we compare this lemma with results from prior research [Balcan et al., 2016].

We now study two hierarchies of AMAs. In Section 5.1.6, 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 $\lambda$-auctions.** Let $\mathcal{Q}$ be a set of allocations. The set of $\mathcal{Q}$-boosted AMAs (resp., $\lambda$-auctions) consists of all AMAs (resp., $\lambda$-auctions) where only allocations in $\mathcal{Q}$ are boosted. In other words, if $\lambda(Q) > 0$, then $Q \in \mathcal{Q}$. (Recall that $\lambda$-auctions [Jehiel et al., 2007] are a special case of AMAs where the buyer weights equal 1.)

**Lemma 5.1.20.** *Let $\mathcal{U}$ and $\mathcal{U}'$ be the sets of profit functions corresponding to the classes of $\mathcal{Q}$-boosted AMAs and $\lambda$-auctions, respectively. Then $\mathcal{U}$ is $\left(3n(n+|\mathcal{Q}|), (n+1)^{2(m+1)}\right)$-delineable and $\mathcal{U}'$ is $\left(|\mathcal{Q}|, (n+1)\left(|\mathcal{Q}|+1\right)^2\right)$-delineable.*

*Proof.* Let $K = (n+1)^m$ be the total number of allocations and let $\boldsymbol{p}$ be a parameter vector where the first $n$ components correspond to the bidder weights $w_j$ for $j \in [n]$, the next $n$ components correspond to $1/w_j$ for $j \in [n]$, the next $2\binom{n}{2}$ components correspond to $w_i/w_j$ for all $i \neq j$, the next $|\mathcal{Q}|$ components correspond to $\lambda(Q)$ for every allocation $Q \in |\mathcal{Q}|$, and the final $n|\mathcal{Q}|$ components correspond to $\lambda(Q)/w_j$ for all allocations $Q \in \mathcal{Q}$ and all bidders $j \in [n]$. In total, the dimension of this parameter space is at most $2n + 2n^2 + |\mathcal{Q}| + n|\mathcal{Q}| < (n+2)(n+|\mathcal{Q}|) \leq 3n(n+|\mathcal{Q}|)$. We set $d = 3n(n+|\mathcal{Q}|)$. Fix some valuation vector $\boldsymbol{v}$. We claim that the allocation of any $\mathcal{Q}$-boosted AMA is determined by at most $(n+1)K^2$ hyperplanes in $\mathbb{R}^d$. To see why this is, the allocation will be $Q^j = \left(\boldsymbol{q}_1^j, \ldots, \boldsymbol{q}_n^j\right)$ where $\sum w_i v_i \left(\boldsymbol{q}_i^j\right) + \lambda\left(Q^j\right) - c(Q^j) \geq \sum w_i v_i\left(\boldsymbol{q}_i^k\right) + \lambda\left(Q^k\right) - c(Q^k)$ for all allocations $Q^k = \left(\boldsymbol{q}_1^k, \ldots, \boldsymbol{q}_n^k\right)$. This decision governing which of the $K$

possible allocations will be the AMA allocation is defined by the $K^2$ hyperplanes, one per pair of distinct allocations $Q^j$ and $Q^k$.

By a similar argument, it is straightforward to see that $K^2$ hyperplanes determine the allocation of any AMA in this restricted space without any one bidder's participation. This leads us to a total of $(n+1)K^2$ hyperplanes which partition the space of $\mathcal{Q}$-boosted AMA parameters in a way such that for any two parameter vectors in the same region, the auction allocations are the same, as are the allocations without any one bidder's participation. Once these allocations are fixed, profit is a linear function in this parameter space.

The proof for $\lambda$-auctions is very similar to that for AMAs. However, we claim that the allocation of any $\mathcal{Q}$-boosted $\lambda$-auction is determined by at most $(n+1)(|\mathcal{Q}|+1)^2$ hyperplanes in $\mathbb{R}^{|\mathcal{Q}|}$. This is because without the bidder weights, the allocation of the $\mathcal{Q}$-boosted $\lambda$-auction will either be a boosted allocation or the VCG allocation if it is not boosted. Therefore, there are only $(|\mathcal{Q}|+1)^2$ hyperplanes determining the allocation of the $\lambda$-auction, and the same number of hyperplanes determine the allocation of the $\lambda$-auction in this restricted space without any one bidder's participation. Once these allocations are fixed, profit is linear function of the $\lambda$-terms. □

**Lotteries.**

We now apply Theorem 5.1.4 to *lottery menus*. Lotteries are randomized mechanisms which are known to generate higher expected revenue than deterministic mechanisms in many settings (e.g., [Conitzer and Sandholm, 2003, Manelli and Vincent, 2006, Thanassoulis, 2004])

A *length-$\ell$ lottery menu* is a set

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

where $\boldsymbol{\rho}^{(0)} = \mathbf{0}$. We assume there is a single additive buyer, but 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 $\rho^{(j)}[m+1]$. Since $v_1(\boldsymbol{e}_i) \cdot \rho^{(j)}[i]$ is his value for item $i$ times the probability they get that item, his expected utility is $\boldsymbol{v} \cdot \left(\rho^{(j)}[1], \ldots, \rho^{(j)}[m]\right) - \rho^{(j)}[m+1]$. Given a buyer with values defined by $\boldsymbol{v}$, 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 $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \rho_{\boldsymbol{v}} - \mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{\rho}_{\boldsymbol{v}}}\left[c\left(\boldsymbol{q}\right)\right]$. Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^{\ell(m+1)}\right\}$.

The key challenge in bounding $\text{Pdim}\left(\mathcal{U}\right)$ is that $\mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{\rho}_{\boldsymbol{v}}}\left[c\left(\boldsymbol{q}\right)\right]$ is not a piecewise linear function of the parameters $\boldsymbol{\rho}^{(0)}, \ldots, \boldsymbol{\rho}^{(\ell)}$. To overcome this challenge, rather than bounding $\text{Pdim}\left(\mathcal{U}\right)$, 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 $\boldsymbol{w} \sim U\left([0,1]^m\right)$, the probability that $w[j]$ is smaller than $\rho_{\boldsymbol{v}}[j]$ is $\rho_{\boldsymbol{v}}[j]$. Therefore, $\mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{\rho}_{\boldsymbol{v}}}\left[c\left(\boldsymbol{q}\right)\right] = \mathbb{E}_{\boldsymbol{w}}\left[c\left(\sum_{j:w[j]<\rho_{\boldsymbol{v}}[j]} \boldsymbol{e}_j\right)\right]$. Given lottery parameters $\boldsymbol{\rho} \in \mathbb{R}^{\ell(m+1)}$, we define $\tilde{u}_{\boldsymbol{\rho}}\left(\boldsymbol{v}, \boldsymbol{w}\right) := \rho_{\boldsymbol{v}}[m+1] - c\left(\sum_{j:w[j]<\rho_{\boldsymbol{v}}[j]} \boldsymbol{e}_j\right)$ 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 $\sum_{j:w[j]<\rho_{\boldsymbol{v}}[j]} \boldsymbol{e}_j$ are defined by a set of hyperplanes.

**Lemma 5.1.21.** *For additive buyers, $\tilde{\mathcal{U}}$ is $\left(\ell(m+1), (\ell+1)^2 + m\ell\right)$-delineable.*

*Proof.* A length-$\ell$ lottery menu is defined by $\ell(m+1)$ parameters. The first $m+1$ parameters (denoted $\rho^{(1)}[1], \ldots, \rho^{(1)}[m+1]$) define the first lottery in the menu, the second $m+1$ parameters (denoted $\rho^{(2)}[1], \ldots, \rho^{(2)}[m+1]$) define the second lottery in the menu, and so on. The buyer will prefer the $j^{th}$ menu entry (defined by the parameters $\rho^{(j)}[1], \ldots, \rho^{(j)}[m+1]$) so long as $\boldsymbol{v} \cdot \left(\rho^{(j)}[1], \ldots, \rho^{(j)}[m]\right) - \rho^{(j)}[m+1] > \boldsymbol{v} \cdot \left(\rho^{(k)}[1], \ldots, \rho^{(k)}[m]\right) - \rho^{(k)}[m+1]$ for any $k \neq j$. In total, these inequalities define $\binom{\ell+1}{2}$ hyperplanes in $\mathbb{R}^{\ell(m+1)}$. In any region defined by these hyperplanes, the menu entry that the buyer prefers is fixed. Next, for each menu entry $\boldsymbol{\rho}^{(k)}$, there are $m$ hyperplanes determining the vector $\sum_{j:w[j]<\rho^{(k)}[j]} \boldsymbol{e}_j$, and thus the cost $c\left(\sum_{j:w[j]<\rho^{(k)}[j]} \boldsymbol{e}_j\right)$. These vectors have the form $w[j] = \rho^{(k)}[j]$. Thus, there are a total of $\ell m$ hyperplanes determining the costs. Let $\mathcal{H}$ be the union of all $(\ell+1)^2 + m\ell$ hyperplanes. Within any connected component of $\mathbb{R}^{\ell(m+1)} \setminus \mathcal{H}$, the menu entry that the buyer buys is fixed and for each menu entry, $c\left(\sum_{j:w[j]<\rho^{(k)}[j]} \boldsymbol{e}_j\right)$ is fixed. Therefore, profit is a linear function of the prices $\rho^{(1)}[m+1], \ldots, \rho^{(\ell)}[m+1]$. □

The following lemma 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 $\boldsymbol{\rho}$, $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{w}}\left[\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v}, \boldsymbol{w})\right]$.

**Lemma 5.1.22.** *With probability $1-\delta$ over the draw*

$$\left\{\left(\boldsymbol{v}^{(1)}, \boldsymbol{w}^{(1)}\right), \ldots, \left(\boldsymbol{v}^{(N)}, \boldsymbol{w}^{(N)}\right)\right\} \sim \left(\mathcal{D} \times U\left([0,1]\right)^m\right)^N,$$

*for all parameter settings $\boldsymbol{\rho} \in \mathbb{R}^{m+1}$,*

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

*Proof.* We know that with probability at least $1-\delta$ over the draw of a sample

$$\left\{\left(\boldsymbol{v}^{(1)}, \boldsymbol{w}^{(1)}\right), \ldots, \left(\boldsymbol{v}^{(N)}, \boldsymbol{w}^{(N)}\right)\right\} \sim \left(\mathcal{D} \times U([0,1])^m\right)^N,$$

for all parameter settings $\boldsymbol{\rho} \in \mathbb{R}^{m+1}$,

$$\left|\frac{1}{N}\sum_{j=1}^{N} \tilde{u}_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(j)}, \boldsymbol{w}^{(j)}\right) - \mathop{\mathbb{E}}_{\boldsymbol{v},\boldsymbol{w}\sim\mathcal{D}\times U([0,1])^m}\left[\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v}, \boldsymbol{w})\right]\right| = O\left(H\sqrt{\frac{1}{N}\left(\mathrm{Pdim}(\tilde{\mathcal{U}}) + \ln\frac{1}{\delta}\right)}\right).$$

We now claim that $\mathbb{E}_{\boldsymbol{v},\boldsymbol{w}\sim\mathcal{D}\times U([0,1])^m}\left[\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v}, \boldsymbol{w})\right] = \mathbb{E}_{\boldsymbol{v}\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right]$. By definition of $\tilde{u}_{\boldsymbol{\rho}}$,

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{w}}\left[\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v}, \boldsymbol{w})\right] &= \mathop{\mathbb{E}}_{\boldsymbol{w}}\left[\rho_{\boldsymbol{v}}[m+1] - c\left(\sum_{j:w[j]<\rho_{\boldsymbol{v}}[j]} \boldsymbol{e}_j\right)\right] \\
&= \rho_{\boldsymbol{v}}[m+1] - \sum_{\boldsymbol{r}\in\{0,1\}^m} c(\boldsymbol{r}) \prod_{j:r[j]=1} \Pr\left[w[j] < \rho_{\boldsymbol{v}}[j]\right] \prod_{j:r[j]=0} \Pr\left[w[j] \geq \rho_{\boldsymbol{v}}[j]\right] \\
&= \rho_{\boldsymbol{v}}[m+1] - \sum_{\boldsymbol{r}\in\{0,1\}^m} c(\boldsymbol{r}) \prod_{j:r[j]=1} \rho_{\boldsymbol{v}}[j] \prod_{j:r[j]=0} \left(1 - \rho_{\boldsymbol{v}}[j]\right).
\end{aligned}$$

From the other direction,

$$\begin{aligned} u_{\boldsymbol{\rho}}(\boldsymbol{v}) &= \rho_v[m+1] - \mathop{\mathbb{E}}_{\boldsymbol{q}\sim\boldsymbol{\rho}_v}[c(\boldsymbol{q})] \\ &= \rho_v[m+1] - \sum_{\boldsymbol{r}\in\{0,1\}^m} c(\boldsymbol{r}) \prod_{j:r[j]=1} \Pr[q[j]=1] \prod_{j:r[j]=0} \Pr[q[j]=0] \\ &= \rho_v[m+1] - \sum_{\boldsymbol{r}\in\{0,1\}^m} c(\boldsymbol{r}) \prod_{j:r[j]=1} \rho_v[j] \prod_{j:r[j]=0} (1-\rho_v[j]). \end{aligned}$$

Therefore, $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \mathbb{E}_{\boldsymbol{w}}\left[\tilde{u}_{\boldsymbol{\rho}}(\boldsymbol{v},\boldsymbol{w})\right]$. Therefore, the lemma statement holds. □

As we have seen in this section, a wide variety of mechanism classes are delineable. Therefore, Theorem 5.1.4 immediately implies a generalization guarantee for a diverse array of mechanism classes.

### 5.1.5 Data-dependent generalization guarantees

In this section, we provide two ways to strengthen the results in Section 5.1.4 when the buyers' values are additive and drawn from *item-independent* distributions. We say a distribution over buyers' values is item-independent if for all $i_1, i_2 \in [n]$ and $j \in [m]$, buyer $i_1$'s value for item $j$ is independent from her value for item $j'$, but her value for item $j$ may be arbitrarily correlated with buyer $i_2$'s value for item $j$ or $j'$. We also require that the mechanism class's 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. Surprisingly, our bounds do not depend on the number of items and under anonymous prices, they do not depend on the number of buyers either.

We make the notion of a distribution-dependent generalization guarantee more formal, as follows.

**Definition 5.1.23.** Let $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ be a set of profit functions parameterized by $\boldsymbol{\rho} \in \mathbb{R}^d$, for some $d$. A *distribution-dependent generalization guarantee* for $\mathcal{U}$ and a distribution $\mathcal{D}$ over buyers' values is a function $\epsilon_{\mathcal{U}}^{\mathcal{D}} : \mathbb{Z}_{\geq 1} \times (0,1) \to \mathbb{R}_{\geq 0}$ defined such that for any sample size $N \in \mathbb{Z}_{\geq 1}$ and any $\delta \in (0,1)$, with probability at least $1-\delta$ over the draw of a set $\mathcal{S} \sim \mathcal{D}^N$, for any parameter setting $\boldsymbol{\rho} \in \mathbb{R}^d$, the difference between the average value of $u_{\boldsymbol{\rho}}$ over $\mathcal{S}$ and the expected value of $u_{\boldsymbol{\rho}}$ over $\mathcal{D}$ is at most $\epsilon_{\mathcal{U}}^{\mathcal{D}}(N,\delta)$. In other words,

$$\Pr_{\mathcal{S}\sim\mathcal{D}^N}\left[\exists \boldsymbol{\rho} \in \mathbb{R}^d \text{ such that } \left|\frac{1}{N}\sum_{v\in\mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) - \mathop{\mathbb{E}}_{v\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right]\right| > \epsilon_{\mathcal{U}}^{\mathcal{D}}(N,\delta)\right] < \delta.$$

To obtain our data-dependent guarantees, we move from pseudo-dimension to Rademacher complexity [Bartlett and Mendelson, 2002, Koltchinskii, 2001]. 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 5.1.29).

In the following corollary of Theorem 5.1.4, we show that if the profit functions of a class $\mathcal{U}$ decompose additively into a number of simpler functions, then we can easily bound $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U})$ 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 buyers are additive and their valuations are drawn from item-independent distributions. This includes buyers with values drawn from product distributions as a special case, a setting which has been extensively studied in the mechanism design literature [e.g., Babaioff et al., 2017, Cai and Daskalakis, 2017a, Cai et al., 2016, Goldner and Karlin, 2016, Hart and Nisan, 2012, Yao, 2014].

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 $u_{\boldsymbol{\rho}}(\cdot) = u_{1,\boldsymbol{\rho}}(\cdot) + \cdots + u_{T,\boldsymbol{\rho}}(\cdot)$. We note that this decomposition is not necessarily into coordinates (for example, it is not necessary that each function $f_{i,M}$ corresponds to some buyer or good).

**Corollary 5.1.24.** *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 $\mathcal{S} \sim \mathcal{D}^N$,*

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

*Proof.* The corollary follows from the fact that for any function class $\mathcal{G}$ with range $[0, H]$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{G}) = O\left(H\sqrt{\frac{\text{Pdim}(\mathcal{G})}{|\mathcal{S}|}}\right)$ Dudley [1987], Pollard [1984], and the fact that for any sets $\mathcal{G}$ and $\mathcal{G}'$ of functions mapping an abstract domain $\mathcal{A}$ to $\mathbb{R}$ and any set $\mathcal{S} \subseteq \mathcal{A}$,

$$\widehat{\mathscr{R}}_{\mathcal{S}}\left(\left\{g + g' : g \in \mathcal{G}, g' \in \mathcal{G}'\right\}\right) \leq \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{G}) + \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{G}').$$

□

**Lemma 5.1.25.** *Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$. Let $\mathcal{F} = \left\{f_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\right\}$ be a set of functions mapping $\mathcal{X}$ to $\mathbb{R}$, parameterized by a set $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_d$. Suppose for $i \in [d]$, there exists a class $\mathcal{F}_i = \left\{f_{\rho}^{(i)} : \rho \in \mathcal{P}_i\right\}$ of functions mapping $\mathcal{X}_i$ to $\mathbb{R}$ such that for any $\boldsymbol{\rho} \in \mathcal{P}$, $f_{\boldsymbol{\rho}}$ decomposes additively as $f_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^{d} f_{\rho[i]}^{(i)}(v[i])$. Then*

$$\sup_{v \in \mathcal{X}, \boldsymbol{\rho} \in \mathcal{P}} f_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^{d} \sup_{v \in \mathcal{X}_i, \rho \in \mathcal{P}_i} f_{\rho}^{(i)}(v).$$

*Proof.* Recall that for any set $A \subseteq \mathbb{R}$, $s = \sup A$ if and only if:

1. For all $\epsilon > 0$, there exists $a \in A$ such that $a > s - \epsilon$, and
2. For all $a \in A$, $a \leq s$.

Let $t_i = \sup_{v \in \mathcal{X}_i, \rho \in \mathcal{P}_i} f_{\rho}^{(i)}(v)$ and let $t = \sum_{i=1}^{d} t_i$. We will show that $t = \sup_{\boldsymbol{v} \in \mathcal{X}, \boldsymbol{\rho} \in \mathcal{P}} f_{\boldsymbol{\rho}}(\boldsymbol{v})$.

First, we will show that condition (1) holds. In particular, we want to show that for all $\epsilon > 0$, there exists $\boldsymbol{v} \in \mathcal{X}$ and $\boldsymbol{\rho} \in \mathcal{P}$ such that $f_{\boldsymbol{\rho}}(\boldsymbol{v}) > t - \epsilon$. Since $t_i = \sup_{v \in \mathcal{X}_i, \rho \in \mathcal{P}_i} f_{\rho}^{(i)}(v)$, we know that there exists $v[i] \in \mathcal{X}_i, \rho_i \in \mathcal{P}$ such that $f_{\rho_i}^{(i)}(v[i]) > t_i - \epsilon/d$. Therefore, letting

$\boldsymbol{\rho} = (\rho_1, \ldots, \rho_d)$, we know that $f_{\boldsymbol{\rho}}(v_1, \ldots, v_d) = \sum_{i=1}^{d} f_{\rho_i}^{(i)}(v[i]) > \sum_{i=1}^{d} t_i - \epsilon = t - \epsilon$. Since $(v_1, \ldots, v_d) \in \mathcal{X}$ and $(\rho_1, \ldots, \rho_d) \in \mathcal{P}$, we may conclude that condition (1) holds.

Next, we will show that condition (2) holds. In particular, we want to show that for all $\boldsymbol{v} \in \mathcal{X}$ and $\boldsymbol{\rho} \in \mathcal{P}$, $f_{\boldsymbol{\rho}}(\boldsymbol{v}) \leq t$. We know that $f_{\rho[i]}^{(i)}(v[i]) \leq t_i$, which means that $f_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^{d} f_{\rho[i]}^{(i)}(v[i]) \leq \sum_{i=1}^{d} t_i = t$. Therefore, condition (2) holds. □

We now instantiate Corollary 5.1.24 for several mechanism classes.

**Lemma 5.1.26.** *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, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{n \log n / N}\right)$.*

*Proof.* We begin with anonymous second-price auctions, which are parameterized by a set $\mathcal{P} \subset \mathbb{R}^m$. Without loss of generality, we may write $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_m$, where $\mathcal{P}_i \subset \mathbb{R}$. Given a valuation vector $\boldsymbol{v}$ and an item $i$, let $\boldsymbol{v}(i) \in \mathbb{R}^n$ be all $n$ buyers' values for item $i$. Let $u_\rho(\boldsymbol{v}(i))$ be the profit obtained by selling item $i$ with a reserve price of $\rho$. Notice that for any $\boldsymbol{\rho} \in \mathcal{P}$, $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^{m} u_{\rho[i]}(\boldsymbol{v}(i))$. Let $\mathcal{X}_i$ be the support of the distribution over $\boldsymbol{v}(i)$ and let $H_i = \sup_{\rho \in \mathcal{P}_i, \boldsymbol{v}(i) \in \mathcal{X}_i} u_\rho(\boldsymbol{v}(i))$. Next, let $\mathcal{X}$ be the support of $\mathcal{D}$. By definition, since $H$ is the maximum profit achievable via second price auctions over valuation vectors from $\mathcal{X}$, we may write $H = \sup_{\boldsymbol{v} \in \mathcal{X}, \boldsymbol{\rho} \in \mathcal{P}} u_{\boldsymbol{\rho}}(\boldsymbol{v})$. Since $\mathcal{D}$ is item-independent, we know that $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$. Therefore, we may apply Lemma 5.1.25, which tells us that $H = \sum_{i=1}^{m} H_i$. Finally, each class of functions $\{u_\rho : \rho \in \mathcal{P}_i\}$ is $(1,2)$-delineable, since for $\boldsymbol{v}(i) \in \mathcal{X}_i$, $u_{\boldsymbol{v}(i)}(\rho)$ is linear so long as $\rho$ is larger than the largest component of $\boldsymbol{v}(i)$, between the second largest and largest component of $\boldsymbol{v}(i)$, or smaller than the second largest component of $\boldsymbol{v}(i)$. By Corollary 5.1.24, we may conclude that for any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) \leq O\left(H\sqrt{1/N}\right)$.

The bound on $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}')$ follows by almost the exact same logic, except for a few adjustments. First of all, the class is defined by $nm$ parameters coming from some set $\mathcal{P} \subseteq \mathbb{R}^{nm}$, since there are $n$ non-anonymous prices per item. Without loss of generality, we assume $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_m$, where $\mathcal{P}_i \subseteq \mathbb{R}^n$ is the set of non-anonymous prices for item $i$. Given a set of non-anonymous prices $\boldsymbol{\rho} \in \mathbb{R}^n$ for item $i$, let $u_{\boldsymbol{\rho}}(\boldsymbol{v}(i))$ be the profit of selling the item the bidders defined by $\boldsymbol{v}(i)$ given the reserve prices $\boldsymbol{\rho}$. Notice that $u_{\boldsymbol{v}(i)}(\boldsymbol{\rho})$ is linear so long as for each bidder $j$, $\rho[j]$ is either larger than their value for item $i$ or smaller than their value. Thus, the set $\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}_i\}$ is $(n,n)$-delineable. Defining each $H_i$ in the same way as before, Lemma 5.1.25 guarantees that $H = \sum_{i=1}^{m} H_i$. Therefore, by Corollary 5.1.24, we may conclude that for any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}') \leq O\left(H\sqrt{n \log n / N}\right)$. □

The following lemma follows from the same logic as Lemma 5.1.26.

**Lemma 5.1.27.** *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, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = O\left(H\sqrt{n \log n / N}\right)$.*

*Proof.* We begin with anonymous item-pricing mechanisms, which are parameterized by a set $\mathcal{P} \subset \mathbb{R}^m$. Without loss of generality, we may write $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_m$, where $\mathcal{P}_i \subset \mathbb{R}$. Given a valuation vector $\boldsymbol{v}$ and an item $i$, let $\boldsymbol{v}(i) \in \mathbb{R}^n$ be all $n$ buyers' values for item $i$. Let $u_\rho(\boldsymbol{v}(i))$

be the profit obtained by selling item $i$ at a price of $\rho$, i.e., $u_\rho(\boldsymbol{v}(i)) = \mathbf{1}_{\{||\boldsymbol{v}(i)||_\infty \geq \rho\}}(\rho - c(\boldsymbol{e}_i))$. Notice that for any $\boldsymbol{\rho} \in \mathcal{P}$, $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^m u_{\rho[i]}(\boldsymbol{v}(i))$. Let $\mathcal{X}_i$ be the support of the distribution over $\boldsymbol{v}(i)$ and let $H_i = \sup_{\rho \in \mathcal{P}_i, \boldsymbol{v}(i) \in \mathcal{X}_i} u_\rho(\boldsymbol{v}(i))$. Next, let $\mathcal{X}$ be the support of $\mathcal{D}$. By definition, since $H$ is the maximum profit achievable via item-pricing mechanisms over valuation vectors from $\mathcal{X}$, we may write $H = \sup_{\boldsymbol{v} \in \mathcal{X}, \boldsymbol{\rho} \in \mathcal{P}} u_{\boldsymbol{\rho}}(\boldsymbol{v})$. Since $\mathcal{D}$ is item-independent, we know that $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$. Therefore, we may apply Lemma 5.1.25, which tells us that $H = \sum_{i=1}^m H_i$. Finally, each class of functions $\left\{u_\rho : \rho \in \mathcal{P}_i\right\}$ is $(1,1)$-delineable, since for $\boldsymbol{v}(i) \in \mathcal{X}_i$, $u_{\boldsymbol{v}(i)}(\rho)$ is linear so long as $||\boldsymbol{v}(i)||_\infty \leq \rho$ or $||\boldsymbol{v}(i)||_\infty > \rho$. By Corollary 5.1.24, we may conclude that for any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) \leq O\left(H\sqrt{1/N}\right)$.

The bound on $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}')$ follows by almost the exact same logic, except for a few adjustments. First of all, the class is defined by $nm$ parameters coming from some set $\mathcal{P} \subseteq \mathbb{R}^{nm}$, since there are $n$ non-anonymous prices per item. Without loss of generality, we assume $\mathcal{P} = \mathcal{P}_1 \times \cdots \times \mathcal{P}_m$, where $\mathcal{P}_i \subseteq \mathbb{R}^n$ is the set of non-anonymous prices for item $i$. Given a set of non-anonymous prices $\boldsymbol{\rho} \in \mathbb{R}^n$ for item $i$, let $u_{\boldsymbol{\rho}}(\boldsymbol{v}(i))$ be the profit of selling the item to the buyers defined by $\boldsymbol{v}(i)$ given the prices $\boldsymbol{\rho}$. Notice that $u_{\boldsymbol{v}(i)}(\boldsymbol{\rho})$ is linear so long as for each buyer $j$, $\rho(\boldsymbol{e}_j)$ is either larger than their value for item $i$ or smaller than their value. Thus, the set $\left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}_i\right\}$ is $(n,n)$-delineable. Defining each $H_i$ in the same way as before, Lemma 5.1.25 guarantees that $H = \sum_{i=1}^m H_i$. Therefore, by Corollary 5.1.24, we may conclude that for any set of samples $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}') \leq O\left(H\sqrt{n \log n / N}\right)$. □

**Menus of item lotteries.** A *length-$\ell$ item lottery menu* is a set of $\ell$ 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 $\sum_{i=1}^m \rho_i^{(j_i)}[2]$.

**Lemma 5.1.28.** *Let $\mathcal{U}$ be the set profit functions corresponding to the class of length-$\ell$ item lottery menus. If the bidder is additive, $\mathcal{D}$ is item-independent, and the cost function is additive, then for any set $\mathcal{S} \sim \mathcal{D}^N$, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) \leq O\left(H\sqrt{\ell \log \ell / N}\right)$.*

*Proof.* For a given menu $M = (M_1, \ldots, M_m)$ of item lotteries, let $u_{M_i}(\boldsymbol{v})$ be the profit achieved from menu $M_i$. Since the cost function is additive, $u_{M_i}(\boldsymbol{v}) = \rho_{i,\boldsymbol{v}}[2] - \mathbb{E}_{q \sim \rho_{i,\boldsymbol{v}}[1]}[c(q)] = \rho_{i,\boldsymbol{v}}[2] - c(\boldsymbol{e}_i) \cdot \rho_{i,\boldsymbol{v}}[1]$, where $\boldsymbol{\rho}_{i,\boldsymbol{v}}$ is the lottery in $M_i$ that maximizes the buyer's utility. Notice that $u_M(\boldsymbol{v}) = \sum_{i=1}^m u_{M_i}(v(\boldsymbol{e}_i))$. Let $\mathcal{X}_i$ be the support of the distribution $\mathcal{D}_i$ over $v(\boldsymbol{e}_i)$ and let $H_i = \sup_{M_i, v(\boldsymbol{e}_i) \in \mathcal{X}_i} u_{M_i}(v(\boldsymbol{e}_i))$. By definition, since $H$ is the maximum profit achievable via item menus over valuation vectors from $\mathcal{X}$, we may write $H = \sup_{\boldsymbol{v} \in \mathcal{X}, M \in \mathcal{U}} u_M(\boldsymbol{v})$. Since $\mathcal{D}$ is a product distribution, we know that $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$. Therefore, we may apply Lemma 5.1.25, which tells us that $H = \sum_{i=1}^m H_i$. Finally, for each $i \in [n]$, the class of all single-item lotteries $M_i$ is $(2\ell, \ell^2)$-delineable, since for $v(\boldsymbol{e}_i) \in \mathcal{X}_i$, the lottery the buyer chooses depends on the $\binom{\ell+1}{2}$ hyperplanes $\rho_i^{(j)}[1]v(\boldsymbol{e}_i) - \rho_i^{(j)}[2] = \rho_i^{(j')}[1]v(\boldsymbol{e}_i) - \rho_i^{(j')}[2]$ for $j, j' \in \{0, \ldots, \ell\}$, and once the lottery is fixed, profit is a linear function. □

Finally, we prove lower bounds showing that one could not hope to prove the generalization guarantees implied by Lemmas 5.1.26 and 5.1.27 using pseudo-dimension alone.

**Theorem 5.1.29.** *Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous item-pricing mechanisms. The* $\text{Pdim}(\mathcal{U}) \geq m$. *If the prices are non-anonymous, then* $\text{Pdim}(\mathcal{U}) \geq nm$. *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.*

*Proof.* We construct a set $\mathcal{S}$ of $m$ single-buyer, $m$-item valuation vectors that can be shattered by $\mathcal{U}$. Let $\boldsymbol{v}^{(i)}$ be valuation vector where $v_1^{(i)}(\boldsymbol{e}_i) = 3$ and $v_1^{(i)}(\boldsymbol{e}_j) = 0$ for all $j \neq i$ and let $\mathcal{S} = \left\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(m)}\right\}$. For any $T \subseteq [m]$, let $\boldsymbol{\rho}_T$ be the prices defined such that the price of item $i$ is 2 if $i \in T$ and otherwise, its price is 0. If $i \in T$, then $u_{\boldsymbol{\rho}_T}(\boldsymbol{v}^{(i)}) = 2$ and otherwise, $u_{\boldsymbol{\rho}_T}(\boldsymbol{v}^{(i)}) = 0$. Therefore, the targets $t^{(1)} = \cdots = t^{(m)} = 1$ witness the shattering of $\mathcal{S}$ by $\mathcal{U}$. This example also proves that the pseudo-dimension of the class of second-price auctions with anonymous reserve prices is also at least $m$, since in the single-buyer case, this class is identical to $\mathcal{U}$.

Next, we construct a set $\mathcal{S}$ of $nm$ $n$-buyer, $m$-item valuation vectors that can be shattered by $\mathcal{U}'$. For $i \in [m]$ and $j \in [n]$, let $\boldsymbol{v}^{(i,j)}$ be valuation vector where $v_j^{(i,j)}(\boldsymbol{e}_i) = 3$ and $v_{j'}^{(i,j)}(\boldsymbol{e}_{i'}) = 0$ for all $(i', j') \neq (i, j)$. Let $\mathcal{S} = \left\{\boldsymbol{v}^{(i,j)}\right\}_{i\in[m], j\in[n]}$. For any $T \subseteq [m] \times [n]$, let $\boldsymbol{\rho}_T$ be the mechanism defined such that the price of item $i$ for buyer $j$ is 2 if $(i, j) \in T$ and otherwise, it is 0. If $(i, j) \in T$, then $u_{\boldsymbol{\rho}_T}(\boldsymbol{v}^{(i,j)}) = 2$ and otherwise, $u_{\boldsymbol{\rho}_T}(\boldsymbol{v}^{(i,j)}) = 0$. Therefore, the targets $t^{(i,j)} = 1$ for all $i \in [m], j \in [n]$ witness the shattering of $\mathcal{S}$ by $\mathcal{U}$. This example with the prices as reserve prices also proves that the pseudo-dimension of the class of second-price auctions with non-anonymous reserve prices is at least $nm$. □

### 5.1.6 Structural profit maximization

In this section, we use our results from Section 5.1.4 to provide tools for optimizing the *profit-generalization tradeoff*, taking inspiration from classic machine learning results on *structural risk minimization* [Blumer et al., 1987, Vapnik, 2013, Vapnik and Chervonenkis, 1974]. 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 $\epsilon_{\mathcal{M}}(N, \delta) = O\left(H\sqrt{\frac{1}{N}\left(\text{Pdim}(\mathcal{U}) + \ln\frac{1}{\delta}\right)}\right)$. 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 [Balcan et al., 2016] gave uniform convergence bounds for AMAs without taking advantage of the class's hierarchical structure. We illustrate[8] uniform convergence bounds in the left panel of Figure 5.3 with $t = 4$. On the $x$-axis, we illustrate the intrinsic complexity of the nested subclasses which can be measured using pseudo-dimension, for example. 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 orange solid line). In particular, the dot on the orange 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 $\hat{M}_j$ over $\mathcal{S}$ is at least as high as the average profit of $\hat{M}_i$ over $\mathcal{S}$, which is why the orange solid line is increasing. Similarly, the dot on the blue dotted line above $\mathcal{M}_i$ illustrates the expected profit of $\hat{M}_i$. This blue 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. The purple dashed line illustrates a naïve lower bound on

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

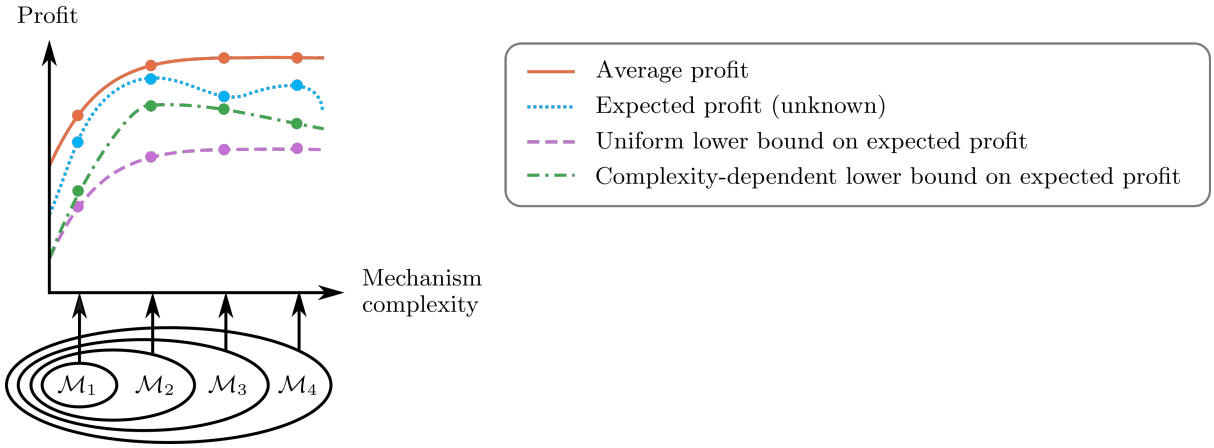



Figure 5.3: An illustration of uniform generalization guarantees versus stronger complexity-dependent bounds for a mechanism class $\mathcal{M}$ that decomposes into a nested sequence of subclasses $\mathcal{M}_1 \subseteq \mathcal{M}_2 \subseteq \mathcal{M}_3 \subseteq \mathcal{M}_4$. The $x$-axis is meant to measure each subclass's intrinsic complexity using a quantity such as pseudo-dimension. Given a fixed set of samples $\mathcal{S}$, the orange solid line plots 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 orange solid line above $\mathcal{M}_i$ illustrates the average profit of $\hat{M}_i$. Similarly, the dot on the blue dotted line above $\mathcal{M}_i$ illustrates the expected profit of $\hat{M}_i$. The purple dashed line 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 green dashed-dotted line 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 5.1.6.

the expected profit of $\hat{M}_i$ which is equal to the average profit of $\hat{M}_i$ over the samples $\mathcal{S}$ minus $\epsilon_{\mathcal{M}}(N, \delta)$. 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 $\delta$ 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 by the green dashed-dotted line in Figure 5.3, 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 stronger 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 5.1.30.** *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 5.1.31.** *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 $\boldsymbol{\rho}$,*

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

### 5.1.7 Comparison of our results to prior research

Here, we provide a brief comparison of this section's results to prior research. These works provide generalization guarantees for a subset of the mechanisms we study. We match or improve over the best-known guarantees for many of the special classes that these papers also studied.

Morgenstern and Roughgarden [2016] studied "simple" multi-item mechanisms: item-pricing mechanisms and second-price item auctions. See Appendix A and Tables 5.3 and 5.4 for a comparison of our guarantees.

Syrgkanis [2017] provided generalization guarantees specifically for the mechanism that maximizes average revenue over the samples. This is in contrast to our bounds, which apply uniformly to every mechanism in a given class. This is crucial when it may not be computationally feasible to determine a mechanism with highest average revenue over the samples, but only an approximation. Syrgkanis [2017] analyzed multi-item, multi-buyer item-pricing mechanisms when the buyers have additive valuations and there is no supplier cost. For a set $\mathcal{S}$ of samples, let $\hat{\boldsymbol{\rho}}$ be the set of anonymous prices that maximize average revenue over the samples. Syrgkanis [2017] proved that with probability $1-\delta$ over the draw of $\mathcal{S} \sim \mathcal{D}^N$,

$$\left| \mathop{\mathbb{E}}_{\boldsymbol{v}\sim\mathcal{D}} \left[u_{\hat{\boldsymbol{\rho}}}(\boldsymbol{v})\right] - \max_{\boldsymbol{\rho}\in\mathbb{R}^m} \mathop{\mathbb{E}}_{\boldsymbol{v}\sim\mathcal{D}} \left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right] \right| = O\left(\frac{H}{\delta}\sqrt{\frac{m\log\left(nN\right)}{N}}\right).$$

When $\mathcal{D}$ is item-independent, our bound of $O\left(H\sqrt{\log\left(1/\delta\right)/N}\right)$ improves over this bound. Otherwise, our bound of $O\left(H\sqrt{m\log\left(m\right)/N} + H\sqrt{\log\left(1/\delta\right)/N}\right)$ is incomparable. Letting $\hat{\boldsymbol{\rho}}'$ be the set of anonymous prices that maximize average revenue over the samples, Syrgkanis [2017] also proved that with probability $1-\delta$,

$$\left| \mathop{\mathbb{E}}_{\boldsymbol{v}\sim\mathcal{D}} \left[u_{\hat{\boldsymbol{\rho}}'}(\boldsymbol{v})\right] - \max_{\boldsymbol{\rho}\in\mathbb{R}^{nm}} \mathop{\mathbb{E}}_{\boldsymbol{v}\sim\mathcal{D}} \left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right] \right| = O\left(\frac{H}{\delta}\sqrt{\frac{nm\log N}{N}}\right).$$

Our bound of $O\left(H\sqrt{nm\log\left(nm\right)/N} + H\sqrt{\log\left(1/\delta\right)/N}\right)$ is incomparable.

Cai and Daskalakis [2017a] provided learning algorithms for buyers with valuations drawn from product distributions, which are item-independent[9]. At a high level, their algorithms return mechanisms with expected revenue that is nearly a constant fraction of the optimal expected revenue ($OPT$) obtainable by any randomized and Bayesian truthful mechanism. When the buyers have additive valuations, they proved that given $N$ samples, the difference between the expected revenue of the mechanism their algorithm outputs and $\frac{OPT}{32}$ is $\epsilon_{\mathcal{U}}^{\mathcal{D}}(N,\delta)$ (as defined in Definition 5.1.23), where $\mathcal{D}$ is the distribution over buyers' values and $\mathcal{U}$ is the set of profit functions for non-anonymous item-pricing mechanisms (Lemma 15 and Theorem 13 of the full paper by Cai and Daskalakis [2017b]). At the time, Morgenstern and Roughgarden's (2016) bound of $\epsilon_{\mathcal{U}}^{\mathcal{D}}(N,\delta) = O\left(H\sqrt{nm\log(nm)/N} + H\sqrt{\log(1/\delta)/N}\right)$ was the best-known bound.[10] Via Lemma 5.1.27, we improve this to $\epsilon_{\mathcal{U}}^{\mathcal{D}}(N,\delta) = O\left(H\sqrt{n\log n/N} + H\sqrt{\log(1/\delta)/N}\right)$, removing the dependence on the number of items. Cai and Daskalakis [2017a] also provided

[9]Item-independent distributions are discussed in Section 5.1.5.

[10]For this bound by Morgenstern and Roughgarden [2016] to hold, revenue must be contained in the interval $[0,H]$, as we assume throughout this chapter.

algorithms for buyers with other types of valuations, such as subadditive and XOS. In these cases, their algorithms return item-pricing mechanisms with entry fees. Our main theorem would provide pessimistic guarantees for these mechanisms due to the exponentially large number of parameters.

Muñoz Medina and Vassilvitskii [2017] studied single-buyer, multi-item pricing in a different model from ours, where there is no bound on the number of items but each item is defined by a feature vector. Their pricing algorithm has access to a bid predictor mapping from feature vectors to bids. They related their algorithm's performance to the bid predictor's accuracy, among other factors.

Devanur et al. [2016] studied several single-item auction classes, including the class of second price item auctions with non-anonymous reserves and no cost function. When the buyers' values are contained in the interval $[1, H]$, they proved that $N = O\left(\left(\frac{H}{\epsilon}\right)^2 \left(n \log \frac{H}{\epsilon} + \log \frac{1}{\delta}\right)\right)$ samples are sufficient to ensure that with probability $1 - \delta$ over the draw $\mathcal{S} \sim \mathcal{D}^N$, for all non-anonymous reserves $\boldsymbol{\rho} \in \mathbb{R}^n$, $\left|\frac{1}{N}\sum_{v \in \mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) - \mathbb{E}_{\boldsymbol{v} \sim \mathcal{D}} u_{\boldsymbol{\rho}}(\boldsymbol{v})\right| \leq \epsilon$ (Section 6.1 of the full paper by Devanur et al. [2017]). Our Lemma 5.1.17 for the multi-item case, specialized to the single-item setting, implies $O\left((H/\epsilon)^2 (n \log n + \log(1/\delta))\right)$ samples are sufficient, which is incomparable to Devanur et al.'s bound due to the log factors.

Gonczarowski and Weinberg [2018] study a setting where there are $n$ buyers with additive, independent values for $m$ items, as well as a generalization to Lipschitz valuation functions. They prove that $\text{poly}(n, m, H, 1/\epsilon)$ samples are sufficient to learn an approximately Bayesian incentive compatible mechanism whose revenue is within an $\epsilon$-factor of optimal. They prove the same result for approximately dominant strategy incentive compatible mechanisms as well. This is an information-theoretic result: the learning algorithm is not computationally efficient since it is not known how to efficiently find an $\epsilon$-approximately optimal mechanism in this setting where the number of types is exponential in the number of items. In contrast, we are able to handle arbitrarily correlated distributions over combinatorial valuations. Moreover, guarantees apply uniformly to any mechanism from within a variety of parameterized mechanism classes, so the mechanism designer can use them to bound the expected profit of the mechanism he obtains via *any* optimization procedure. However, there may not always be a mechanism in these parameterized classes that competes with the optimal incentive compatible mechanism. Meanwhile, Gonczarowski and Weinberg [2018] bound the number of samples sufficient to compete with the optimal incentive compatible mechanism, and thus are not restricted to a parameterized set of mechanisms, but require the buyers' values to be independent and Lipschitz.

## 5.2 Social welfare maximization

A large body of research in economics studies how to design mechanisms that help groups of agents come to collective decisions. For example, when children inherit an estate, how should they divide the property? When a jointly-owned company is dissolved, which partner should buy the others out? There is no one protocol that best answers these questions; the optimal mechanism depends on the setting at hand.

In a STOC'21 paper [Balcan et al., 2021a] with Nina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, and Tuomas Sandholm, we study a family of mechanisms called *neutral affine maximizers (NAMs)* Mishra and Sen [2012], Nath and Sandholm [2019], Roberts [1979]. A NAM

takes as input a set of agents' reported values for each possible outcome and returns one of those outcomes. A NAM can thus be thought of as an algorithm that the agents use to arrive at a single outcome. NAMs are *incentive compatible*, which means that each agent is incentivized to report his values truthfully. In order to satisfy incentive compatibility, each agent may have to make a payment. NAMs are also *budget-balanced* which means that the aggregated payments are redistributed among the agents.

Formally, we study a setting where there is a set 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 of his values as $\boldsymbol{v}_i \in \mathbb{R}^m$ and all $n$ agents' values as $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \in \mathbb{R}^{nm}$. In this case, the unknown distribution $\mathcal{D}$ is over vectors $\boldsymbol{v} \in \mathbb{R}^{nm}$.

A NAM 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. There is a *social choice function* $\psi_{\boldsymbol{\rho}} : \mathbb{R}^{nm} \to [m]$ which uses the values $\boldsymbol{v} \in \mathbb{R}^{nm}$ to choose an alternative $\psi_{\boldsymbol{\rho}}(\boldsymbol{v}) \in [m]$. In particular, $\psi_{\boldsymbol{\rho}}(\boldsymbol{v}) = \operatorname{argmax}_{j \in [m]} \sum_{i=1}^n \rho[i] v_i(j)$ maximizes the agents' weighted values. Each agent $i$ with zero weight $\rho[i] = 0$ is called a *sink agent* because his values do not influence the outcome. For every agent who is not a sink agent ($\rho[i] \neq 0$), their payment is defined as in the weighted version of the classic Vickrey-Clarke-Groves mechanism [Clarke, 1971, Groves, 1973, Vickrey, 1961]. To achieve budget balance, these payments are given to the sink agent(s). More formally, let $j^* = \psi_{\boldsymbol{\rho}}(\boldsymbol{v})$ 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

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

We aim to optimize the expected social welfare $\mathbb{E}_{\boldsymbol{v} \sim \mathcal{D}} \left[ \sum_{i=1}^n v_i\left(\psi_{\boldsymbol{\rho}}(\boldsymbol{v})\right) \right]$ of the NAM's outcome $\psi_{\boldsymbol{\rho}}(\boldsymbol{v})$, so we define the utility function $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^n v_i\left(\psi_{\boldsymbol{\rho}}(\boldsymbol{v})\right)$.

**Lemma 5.2.1.** *For any valuation vector $\boldsymbol{v} \in \mathbb{R}^{nm}$, let $u_{\boldsymbol{v}} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}$ denote the function $u_{\boldsymbol{v}}(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(\boldsymbol{v})$. There is a set $\mathcal{H}$ of at most $m^2$ hyperplanes such that in any connected component $R$ of $\mathbb{R}^n \setminus \mathcal{H}$, $u_{\boldsymbol{v}}(\boldsymbol{\rho})$ is constant across all $\boldsymbol{\rho} \in R$.*

*Proof.* Fix a valuation vector $\boldsymbol{v} \in \mathbb{R}^{nm}$. We know that for any two alternatives $j, j' \in [m]$, the alternative $j$ would be selected over $j'$ so long as

$$\sum_{i=1}^n \rho[i] v_i(j) > \sum_{i=1}^n \rho[i] v_i\left(j'\right). \tag{5.1}$$

Therefore, there is a set $\mathcal{H}$ of $\binom{m}{2}$ hyperplanes such that across all parameter vectors $\boldsymbol{\rho}$ in a single connected component of $\mathbb{R}^n \setminus \mathcal{H}$, the outcome of the NAM defined by $\boldsymbol{\rho}$ is fixed. When the outcome of the NAM is fixed, the social welfare is fixed as well. This means that for a single connected component $R$ of $\mathbb{R}^n \setminus \mathcal{H}$, there exists a real value $c_R$ such that $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = u_{\boldsymbol{v}}(\boldsymbol{\rho}) = c_R$ for all $\boldsymbol{\rho} \in R$. □

This fact implies the following pseudo-dimension bound. The proof is similar to that of Theorem 4.3.21 from Section 4.3.3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 5.2.2.** *The pseudo-dimension of* $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}, \{\rho[i] \mid i = 0\} \neq \emptyset\right\}$ *is* $O(n \ln(nm))$.

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

**Theorem 5.2.3.** *The pseudo-dimension of* $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}, \{\rho[i] \mid i = 0\} \neq \emptyset\right\}$ *is* $\Omega(n)$.

*Proof.* Let the number of alternatives $m = 2$ and without loss of generality, suppose that $n$ is even. To prove this theorem, we will identify a set of $N = \frac{n}{2}$ valuation vectors $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}$ that are shattered by the set $\mathcal{U}$ of social welfare functions.

Let $\epsilon$ be an arbitrary number in $\left(0, \frac{1}{2}\right)$. For each $\ell \in [N]$, define agent $i$'s values for the first and second alternatives under the $\ell^{th}$ valuation vector $\boldsymbol{v}^{(\ell)}$—namely, $v_i^{(\ell)}(1)$ and $v_i^{(\ell)}(2)$—as follows:

$$v_i^{(\ell)}(1) = \begin{cases} 1 & \text{if } \ell = i \\ 0 & \text{otherwise} \end{cases} \text{ and } v_i^{(\ell)}(2) = \begin{cases} \epsilon & \text{if } \ell = \frac{n}{2} + i \\ 0 & \text{otherwise.} \end{cases}$$

For example, if there are $n = 6$ agents, then across the $N = \frac{n}{2} = 3$ valuation vectors $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \boldsymbol{v}^{(3)}$, the agents' values for the first alternative are defined as

$$\begin{bmatrix} v_1^{(1)}(1) & \cdots & v_6^{(1)}(1) \\ v_1^{(2)}(1) & \cdots & v_6^{(2)}(1) \\ v_1^{(3)}(1) & \cdots & v_6^{(3)}(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

and their values for the second alternative are defined as

$$\begin{bmatrix} v_1^{(1)}(2) & \cdots & v_6^{(1)}(2) \\ v_1^{(2)}(2) & \cdots & v_6^{(2)}(2) \\ v_1^{(3)}(2) & \cdots & v_6^{(3)}(2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \epsilon & 0 & 0 \\ 0 & 0 & 0 & 0 & \epsilon & 0 \\ 0 & 0 & 0 & 0 & 0 & \epsilon \end{bmatrix}.$$

Let $\boldsymbol{b} \in \{0,1\}^N$ be an arbitrary bit vector. We will construct a NAM parameter vector $\boldsymbol{\rho}$ such that for any $\ell \in [N]$, if $b_\ell = 0$, then the outcome of the NAM given bids $\boldsymbol{v}^{(\ell)}$ will be the second alternative, so $u_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(\ell)}\right) = \epsilon$ because there is always exactly one agent who has a value of $\epsilon$ for the second alternative, and every other agent has a value of 0. Meanwhile, if $b_\ell = 0$, then the outcome of the NAM given bids $\boldsymbol{v}^{(\ell)}$ will be the first alternative, so $u_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(\ell)}\right) = 1$ because there is always exactly one agent who has a value of 1 for the first alternative, and every other agent has a value of 0. To do so, when $b_\ell = 0$, $\boldsymbol{\rho}$ must ignore the values of agent $\ell$ in favor of the values of agent $\frac{n}{2} + \ell$. After all, under $\boldsymbol{v}^{(\ell)}$, agent $\ell$ has a value of 1 for the first alternative and agent $\frac{n}{2} + \ell$ has a value of $\epsilon$ for the second alternative, and all other values are 0. By a similar argument, when $b_\ell = 1$, $\boldsymbol{\rho}$ must ignore the values of agent $\frac{n}{2} + \ell$ in favor of the values of agent $\ell$. Specifically, we define $\boldsymbol{\rho} \in \{0,1\}^n$ as follows: for all $\ell \in [N] = \left[\frac{n}{2}\right]$, if $b_\ell = 0$, then $\rho[\ell] = 0$ and $\rho\left[\frac{n}{2} + \ell\right] = 1$ and if $b_\ell = 1$, then $\rho[\ell] = 1$ and $\rho\left[\frac{n}{2} + \ell\right] = 0$. All other entries of $\boldsymbol{\rho}$ are set to 0.

We claim that if $b_\ell = 0$, then $u_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(\ell)}\right) = \epsilon$. To see why, we know that $\sum_{i=1}^n \rho[i] v_i^{(\ell)}(1) = \rho[\ell] v_\ell^{(\ell)}(1) = \rho[\ell] = 0$. Meanwhile, $\sum_{i=1}^n \rho[i] v_i^{(\ell)}(2) = \rho\left[\frac{n}{2} + \ell\right] v^{(\ell)}_{\frac{n}{2}+\ell}(1) = \epsilon$. Therefore, the outcome of the NAM is alternative 2. The social welfare of this alternative is $\epsilon$, so $u_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(\ell)}\right) = \epsilon$.

Next, we claim that if $b_\ell = 1$, then $u_\rho\left(\boldsymbol{v}^{(\ell)}\right) = 1$. To see why, we know that $\sum_{i=1}^{n} \rho[i] v_i^{(\ell)}(1) = \rho[\ell] v_\ell^{(\ell)}(1) = \rho[\ell] = 1$. Meanwhile, $\sum_{i=1}^{n} \rho[i] v_i^{(\ell)}(2) = \rho\left[\frac{n}{2} + \ell\right] v_{\frac{n}{2}+\ell}^{(\ell)}(1) = 0$. Therefore, the outcome of the NAM is alternative 1. The social welfare of this alternative is 1, so $u_\rho\left(\boldsymbol{v}^{(\ell)}\right) = 1$.

We conclude that the valuation vectors $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}$ that are shattered by the set $\mathcal{U}$ of social welfare functions with witnesses $z^{(1)} = \cdots = z^{(N)} = \frac{1}{2}$. □

Theorem 5.2.3 implies that the pseudo-dimension upper bound from Theorem 5.2.2 is tight up to logarithmic factors.

# Chapter 6

# Computational biology algorithms

In this chapter, we study algorithms that are used in practice for three biological problems: sequence alignment, RNA folding, and predicting topologically associated domains in DNA. In these applications, there are two unifying similarities. First, algorithmic performance is measured in terms of the distance between the algorithm's output and a ground-truth solution. In most cases, this solution is discovered using laboratory experimentation, so it is only available for the instances in the training set. Second, these algorithms use dynamic programming to maximize parameterized objective functions. This objective function represents a surrogate optimization criterion for the dynamic programming algorithm, whereas utility measures how well the algorithm's output resembles the ground truth. There may be multiple solutions that maximize this objective function, which we call *co-optimal*. Although co-optimal solutions have the same objective function value, they may have different utilities. To handle tie-breaking, we assume that in any region of the parameter space where the set of co-optimal solutions is fixed, the algorithm's output is also fixed, which is typically true in practice.

The results in this chapter are joint work with Nina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, and Tuomas Sandholm and appeared in STOC'21 [Balcan et al., 2021a].

## 6.1 Global pairwise sequence alignment

In pairwise sequence alignment, the goal is to line up strings in order to identify regions of similarity. In biology, for example, these similar regions indicate functional, structural, or evolutionary relationships between the sequences. Formally, let $\Sigma$ be an alphabet and let $S_1, S_2 \in \Sigma^n$ be two sequences. 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 $\mathrm{del}$ is a function that deletes every $-$, or *gap character*. There are many features of an alignment that one might wish to optimize, such as the number of *matches* ($\tau_1[i] = \tau_2[i]$), *mismatches* ($\tau_1[i] \neq \tau_2[i]$), *indels* ($\tau_1[i] = -$ or $\tau_2[i] = -$), and *gaps* (maximal sequences of consecutive gap characters in $\tau \in \{\tau_1, \tau_2\}$). We denote these features using functions $\ell_1, \ldots, \ell_d$ that map pairs of sequences $(S_1, S_2)$ and alignments $L$ to $\mathbb{R}$.

A common dynamic programming algorithm $A_\rho$ [Gotoh, 1982, Waterman et al., 1976] returns the alignment $L$ that maximizes the objective function

$$\rho[1] \cdot \ell_1(S_1, S_2, L) + \cdots + \rho[d] \cdot \ell_d(S_1, S_2, L), \tag{6.1}$$

where $\boldsymbol{\rho} \in \mathbb{R}^d$ is a parameter vector. We denote the output alignment as $A_{\boldsymbol{\rho}}(S_1, S_2)$. As Gusfield, Balasubramanian, and Naor [1994] wrote, "there is considerable disagreement among molecular biologists about the correct choice" of a parameter setting $\boldsymbol{\rho}$.

We assume that there is a utility function that characterizes an alignment's quality, denoted $u(S_1, S_2, L) \in \mathbb{R}$. For example, $u(S_1, S_2, L)$ might measure the distance between $L$ and a "ground truth" alignment of $S_1$ and $S_2$ [Sauder et al., 2000]. We then define $u_{\boldsymbol{\rho}}(S_1, S_2) = u\left(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2)\right)$ to be the utility of the alignment returned by the algorithm $A_{\boldsymbol{\rho}}$.

In the following lemma, we prove that for any pair of sequences, each function in $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ is a piecewise-constant function of the parameter vector $\boldsymbol{\rho}$. This will allow us to bound the pseudo-dimension of $\mathcal{U}$.

**Lemma 6.1.1.** *For any pair of sequences $S_1, S_2 \in \Sigma^n$, let $u_{S_1,S_2} : \mathbb{R}^d \to \mathbb{R}$ denote the function $u_{S_1,S_2}(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(S_1, S_2)$. There is a set $\mathcal{H}$ of at most $4^n n^{4n+2}$ hyperplanes such that in any connected component $R$ of $\mathbb{R}^d \setminus \mathcal{H}$, $u_{S_1,S_2}(\boldsymbol{\rho})$ is constant across all $\boldsymbol{\rho} \in R$.*

*Proof.* Fix a sequence pair $S_1$ and $S_2$. Let $\mathcal{L}$ be the set of alignments the algorithm returns as we range over all parameter vectors $\boldsymbol{\rho} \in \mathbb{R}^d$. In other words, $\mathcal{L} = \left\{A_{\boldsymbol{\rho}}(S_1, S_2) \mid \boldsymbol{\rho} \in \mathbb{R}^d\right\}$. In the following claim, we prove that $|\mathcal{L}| \leq 2^n n^{2n+1}$.

**Claim 6.1.2.** *Fix a pair of sequences $S_1, S_2 \in \Sigma^n$. There are at most $2^n n^{2n+1}$ alignments of $S_1$ and $S_2$.*

*Proof of Claim 6.1.2.* For any alignment $(\tau_1, \tau_2)$, we know that $|\tau_1| = |\tau_2|$ and for all $i \in [|\tau_1|]$, if $\tau_1[i] = -$, then $\tau_2[i] \neq -$ and vice versa. This means that $\tau_1$ and $\tau_2$ have the same number of gaps. To prove the upper bound, we count the number of alignments $(\tau_1, \tau_2)$ where $\tau_1$ and $\tau_2$ each have exactly $i$ gaps. There are $\binom{n+i}{i}$ choices for the sequence $\tau_1$. Given a sequence $\tau_1$, we can only pair a gap in $\tau_2$ with a non-gap in $\tau_1$. Since there are $i$ gaps in $\tau_2$ and $n$ non-gaps in $\tau_1$, there are $\binom{n}{i}$ choices for the sequence $\tau_2$ once $\tau_1$ is fixed. This means that there are $\binom{n+i}{i}\binom{n}{i} \leq 2^n n^{2n}$ alignments $(\tau_1, \tau_2)$ where $\tau_1$ and $\tau_2$ each have exactly $i$ gaps. Summing over $i \in [n]$, the total number of alignments is at most $2^n n^{2n+1}$. □

For any alignment $L \in \mathcal{L}$, the algorithm $A_{\boldsymbol{\rho}}$ will return $L$ if and only if

$$\rho[1] \cdot \ell_1(S_1, S_2, L) + \cdots + \rho[d] \cdot \ell_d(S_1, S_2, L) > \rho[1] \cdot \ell_1\left(S_1, S_2, L'\right) + \cdots + \rho[d] \cdot \ell_d\left(S_1, S_2, L'\right)$$

for all $L' \in \mathcal{L} \setminus \{L\}$. Therefore, there is a set $\mathcal{H}$ of at most $\binom{2^n n^{2n+1}}{2} \leq 4^n n^{4n+2}$ hyperplanes such that across all parameter vectors $\boldsymbol{\rho}$ in a single connected component of $\mathbb{R}^d \setminus \mathcal{H}$, the output of the algorithm parameterized by $\boldsymbol{\rho}$, $A_{\boldsymbol{\rho}}(S_1, S_2)$, is fixed. (As is standard, $\mathbb{R}^d \setminus \mathcal{H}$ indicates set removal.) This means that for any connected component $R$ of $\mathbb{R}^d \setminus \mathcal{H}$, there exists a real value $c_R$ such that $u_{\boldsymbol{\rho}}(S_1, S_2) = c_R$ for all $\boldsymbol{\rho} \in R$. □

This fact implies the following pseudo-dimension bound. The proof is similar to that of Theorem 4.3.21 from Section 4.3.3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 6.1.3.** *The pseudo-dimension of $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ is $O(nd \ln n + d \ln d)$.*

**Tighter guarantees for a structured algorithm subclass: the affine-gap model**

A line of prior work [Fernández-Baca et al., 2004, Gusfield et al., 1994, Pachter and Sturmfels, 2004a,b] analyzed a specific instantiation of the objective function (6.1) where $d = 3$. In this case, we can obtain a pseudo-dimension bound of $O(\ln n)$, which is exponentially better than the bound from Theorem 6.1.3. Given a pair of sequences $S_1, S_2 \in \Sigma^n$, the dynamic programming algorithm $A_{\boldsymbol{\rho}}$ returns the alignment $L$ maximizes 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 matches, $\textsc{ms}(S_1, S_2, L)$ is the number of mismatches, $\textsc{id}(S_1, S_2, L)$ is the number of indels, $\textsc{gp}(S_1, S_2, L)$ is the number of gaps, and $\boldsymbol{\rho} = (\rho[1], \rho[2], \rho[3]) \in \mathbb{R}^3$ is a parameter vector. We denote the output alignment as $A_{\boldsymbol{\rho}}(S_1, S_2)$. This is known as the *affine-gap scoring model*. We exploit specific structure exhibited by this algorithm family to obtain the exponential pseudo-dimension improvement. This useful structure guarantees that for any pair of sequences $S_1$ and $S_2$, there are only $O\left(n^{3/2}\right)$ different alignments the algorithm family $\left\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^3\right\}$ might produce as we range over parameter vectors [Fernández-Baca et al., 2004, Gusfield et al., 1994, Pachter and Sturmfels, 2004a]. This bound is exponentially smaller than our generic bound of $4^n n^{4n+2}$ from Claim 6.1.2.

**Lemma 6.1.4.** *For any pair of sequences $S_1, S_2 \in \Sigma^n$, let $u_{S_1,S_2} : \mathbb{R}^3 \to \mathbb{R}$ denote the function $u_{S_1,S_2}(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(S_1, S_2)$. There is a set $\mathcal{H}$ of $O(n^3)$ hyperplanes such that in any connected component $R$ of $\mathbb{R}^d \setminus \mathcal{H}$, $u_{S_1,S_2}(\boldsymbol{\rho})$ is constant across all $\boldsymbol{\rho} \in R$.*

*Proof.* Fix a sequence pair $S_1$ and $S_2$. Let $\mathcal{L}$ be the set of alignments the algorithm returns as we range over all parameter vectors $\boldsymbol{\rho} \in \mathbb{R}^3$. In other words, $\mathcal{L} = \{A_{\boldsymbol{\rho}}(S_1, S_2) \mid \boldsymbol{\rho} \in \mathbb{R}^3\}$. From prior research [Fernández-Baca et al., 2004, Gusfield et al., 1994, Pachter and Sturmfels, 2004a], we know that $|\mathcal{L}| = O\left(n^{3/2}\right)$. For any alignment $L \in \mathcal{L}$, the algorithm $A_{\boldsymbol{\rho}}$ will return $L$ if and only if

$$\begin{aligned} & \textsc{mt}(S_1, S_2, L) - \rho[1] \cdot \textsc{ms}(S_1, S_2, L) - \rho[2] \cdot \textsc{id}(S_1, S_2, L) - \rho[3] \cdot \textsc{gp}(S_1, S_2, L) \\ > \; & \textsc{mt}(S_1, S_2, L') - \rho[1] \cdot \textsc{ms}(S_1, S_2, L') - \rho[2] \cdot \textsc{id}(S_1, S_2, L') - \rho[3] \cdot \textsc{gp}(S_1, S_2, L') \end{aligned}$$

for all $L' \in \mathcal{L} \setminus \{L\}$. Therefore, there is a set $\mathcal{H}$ of at most $O\left(n^3\right)$ hyperplanes such that across all parameter vectors $\boldsymbol{\rho}$ in a single connected component of $\mathbb{R}^3 \setminus \mathcal{H}$, the output of the algorithm parameterized by $\boldsymbol{\rho}$, $A_{\boldsymbol{\rho}}(S_1, S_2)$, is fixed. □

Lemma 6.1.4 implies that $\text{Pdim}(\mathcal{U}) = O(\ln n)$. Again, the proof is similar to that of Theorem 4.3.21 from Section 4.3.3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 6.1.5.** *The pseudo-dimension of $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^3\right\}$ is $O(\ln n)$.*

We also prove that this pseudo-dimension bound is tight up to constant factors. In this lower bound proof, our utility function $u$ is the *Q score* between a given alignment $L$ of two sequences $(S_1, S_2)$ and the ground-truth alignment $L^*$ (the Q score is also known as the *SPS score* in the case of multiple sequence alignment [Edgar, 2010]). The $Q$ score between $L$ and the ground-truth alignment $L^*$ is the fraction of aligned letter pairs in $L^*$ that are correctly reproduced in $L$. For example, the following alignment $L$ has a Q score of $\frac{2}{3}$ because it correctly aligns the two pairs of Cs, but not the pair of Gs:

$$L = \begin{bmatrix} \text{G} & \text{A} & \text{T} & \text{C} & \text{C} \\ \text{A} & \text{G} & \text{-} & \text{C} & \text{C} \end{bmatrix} \qquad L^* = \begin{bmatrix} \text{-} & \text{G} & \text{A} & \text{T} & \text{C} & \text{C} \\ \text{A} & \text{G} & \text{-} & \text{-} & \text{C} & \text{C} \end{bmatrix}.$$

We use the notation $u(S_1, S_2, L) \in [0,1]$ to denote the Q score between $L$ and the ground-truth alignment of $S_1$ and $S_2$.

**Theorem 6.1.6.** *There exists a set $\left\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^3_{\geq 0}\right\}$ of co-optimal-constant algorithms and an alphabet $\Sigma$ such that the set of functions $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u\left(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2)\right) \mid \boldsymbol{\rho} \in \mathbb{R}^3_{\geq 0}\right\}$, which map sequence pairs $S_1, S_2 \in \cup_{i=1}^{n} \Sigma^i$ of length at most $n$ to $[0,1]$, has a pseudo-dimension of $\Omega(\log n)$.*

First, we provide a high-level proof sketch of this theorem before proving it in its entirety.

*Proof sketch.* In this proof sketch, we illustrate the way in which two sequences pairs can be shattered, and then describe how the proof can be generalized to $\Theta(\log n)$ sequence pairs.

**Setup.** Our setup consists of the following three elements: the alphabet, the two sequence pairs $\left(S_1^{(1)}, S_2^{(1)}\right)$ and $\left(S_1^{(2)}, S_2^{(2)}\right)$, and ground-truth alignments of these pairs. We detail these elements below:

1. Our alphabet consists of twelve characters: $\{\mathrm{a}_i, \mathrm{b}_i, \mathrm{c}_i, \mathrm{d}_i\}_{i=1}^{3}$.

2. The two sequence pairs are comprised of three subsequence pairs: $\left(t_1^{(1)}, t_2^{(1)}\right)$, $\left(t_1^{(2)}, t_2^{(2)}\right)$, and $\left(t_1^{(3)}, t_2^{(3)}\right)$, where

$$\begin{array}{ll} t_1^{(1)} = \mathrm{a}_1\mathrm{b}_1\mathrm{d}_1 & t_1^{(2)} = \mathrm{a}_2\mathrm{a}_2\mathrm{b}_2\mathrm{d}_2 \\ t_2^{(1)} = \mathrm{b}_1\mathrm{c}_1\mathrm{d}_1 & t_2^{(2)} = \mathrm{b}_2\mathrm{c}_2\mathrm{c}_2\mathrm{d}_2 \end{array}, \quad \text{and} \quad \begin{array}{l} t_1^{(3)} = \mathrm{a}_3\mathrm{a}_3\mathrm{a}_3\mathrm{b}_3\mathrm{d}_3 \\ t_2^{(3)} = \mathrm{b}_3\mathrm{c}_3\mathrm{c}_3\mathrm{c}_3\mathrm{d}_3 \end{array}. \tag{6.2}$$

   We define the two sequence pairs as

$$\begin{array}{ll} S_1^{(1)} = t_1^{(1)}t_1^{(2)}t_1^{(3)} = \mathrm{a}_1\mathrm{b}_1\mathrm{d}_1\mathrm{a}_2\mathrm{a}_2\mathrm{b}_2\mathrm{d}_2\mathrm{a}_3\mathrm{a}_3\mathrm{a}_3\mathrm{b}_3\mathrm{d}_3 \\ S_2^{(1)} = t_2^{(1)}t_2^{(2)}t_2^{(3)} = \mathrm{b}_1\mathrm{c}_1\mathrm{d}_1\mathrm{b}_2\mathrm{c}_2\mathrm{c}_2\mathrm{d}_2\mathrm{b}_3\mathrm{c}_3\mathrm{c}_3\mathrm{c}_3\mathrm{d}_3 \end{array} \quad \text{and} \quad \begin{array}{l} S_1^{(2)} = t_1^{(2)} = \mathrm{a}_2\mathrm{a}_2\mathrm{b}_2\mathrm{d}_2 \\ S_2^{(2)} = t_2^{(2)} = \mathrm{b}_2\mathrm{c}_2\mathrm{c}_2\mathrm{d}_2 \end{array}.$$

3. Finally, we define ground-truth alignments of the two sequence pairs $\left(S_1^{(1)}, S_2^{(1)}\right)$ and $\left(S_1^{(2)}, S_2^{(2)}\right)$. We define the ground-truth alignment of $\left(S_1^{(1)}, S_2^{(1)}\right)$ to be

$$\begin{array}{cccccccccccccccccc} \mathrm{a}_1 & \mathrm{b}_1 & - & \mathrm{d}_1 & \mathrm{a}_2 & \mathrm{a}_2 & \mathrm{b}_2 & - & - & \mathrm{d}_2 & \mathrm{a}_3 & \mathrm{a}_3 & \mathrm{a}_3 & \mathrm{b}_3 & - & - & - & \mathrm{d}_3 \\ \mathrm{b}_1 & - & \mathrm{c}_1 & \mathrm{d}_1 & - & - & \mathrm{b}_2 & \mathrm{c}_2 & \mathrm{c}_2 & \mathrm{d}_2 & \mathrm{b}_3 & - & - & - & \mathrm{c}_3 & \mathrm{c}_3 & \mathrm{c}_3 & \mathrm{d}_3 \end{array}. \tag{6.3}$$

   The most important properties of this alignment are that the $\mathrm{d}_j$ characters are always matching and the $\mathrm{b}_j$ characters alternate between matching and not matching. Similarly, we define the ground-truth alignment of the pair $\left(S_1^{(2)}, S_2^{(2)}\right)$ to be

$$\begin{array}{cccccc} \mathrm{a}_2 & \mathrm{a}_2 & \mathrm{b}_2 & - & - & \mathrm{d}_2 \\ - & - & \mathrm{b}_2 & \mathrm{c}_2 & \mathrm{c}_2 & \mathrm{d}_2 \end{array}.$$

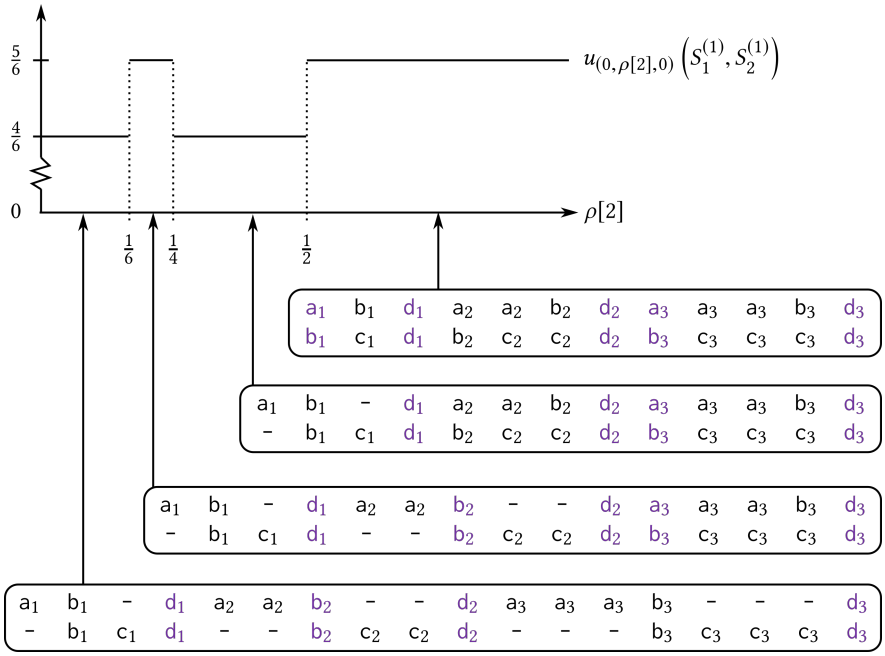



Figure 6.1: The form of $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right)$ as a function of the indel parameter $\rho[2]$. When $\rho[2] \leq \frac{1}{6}$, the algorithm returns the bottom alignment. When $\frac{1}{6} < \rho[2] \leq \frac{1}{4}$, the algorithm returns the alignment that is second to the bottom. When $\frac{1}{4} < \rho[2] \leq \frac{1}{2}$, the algorithm returns the alignment that is second to the top. Finally, when $\rho[2] > \frac{1}{2}$, the algorithm returns the top alignment. The purple characters denote which characters are correctly aligned according to the ground-truth alignment (Equation (6.3)).

**Shattering.** We now show that these two sequence pairs can be shattered. A key step is proving that the functions $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right)$ and $u_{(0,\rho[2],0)}\left(S_1^{(2)}, S_2^{(2)}\right)$ have the following form:

$$u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right) = \begin{cases} \frac{4}{6} & \text{if } \rho[2] \leq \frac{1}{6} \\ \frac{5}{6} & \text{if } \frac{1}{6} < \rho[2] \leq \frac{1}{4} \\ \frac{4}{6} & \text{if } \frac{1}{4} < \rho[2] \leq \frac{1}{2} \\ \frac{5}{6} & \text{if } \rho[2] > \frac{1}{2} \end{cases} \text{ and } u_{(0,\rho[2],0)}\left(S_1^{(2)}, S_2^{(2)}\right) = \begin{cases} 1 & \text{if } \rho[2] \leq \frac{1}{4} \\ \frac{1}{2} & \text{if } \rho[2] > \frac{1}{4} \end{cases}. \quad (6.4)$$

The form of $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right)$ is illustrated by Figure 6.1. It is then straightforward to verify that the two sequence pairs are shattered by the parameter settings $(0,0,0)$, $\left(0,\frac{1}{5},0\right)$, $\left(0,\frac{1}{3},0\right)$, and $(0,1,0)$ with the witnesses $z_1 = z_2 = \frac{3}{4}$. In other words, the mismatch and gap parameters are set to 0 and the indel parameter $\rho[2]$ takes the values $\left\{0, \frac{1}{5}, \frac{1}{3}, 1\right\}$.

**Proof sketch of Equation** (6.4)**.** The full proof that Equation (6.4) holds follows the following high-level reasoning:

1. First, we prove that under the algorithm's output alignment, the $\mathrm{d}_j$ characters will always be matching. Intuitively, this is because the algorithm's objective function will always be maximized when each subsequence $t_1^{(j)}$ is aligned with $t_2^{(j)}$.

2. Second, we prove that the characters $\mathrm{b}_j$ will be matched if and only if $\rho[2] \leq \frac{1}{2j}$. Intuitively, this is because in order to match these characters, we must pay with $2j$ indels. Since the objective function is $\textsc{mt}\left(S_1^{(1)}, S_2^{(1)}, L\right) - \rho[2] \cdot \textsc{id}\left(S_1^{(1)}, S_2^{(1)}, L\right)$, the 1 match will be worth the $2j$ indels if and only if $1 \geq 2j\rho[2]$.

These two properties in conjunction mean that when $\rho[2] > \frac{1}{2}$, none of the $\mathrm{b}_j$ characters are matched, so the characters that are correctly aligned (as per the ground-truth alignment (Equation (6.3))) in the algorithm's output are $(\mathrm{a}_1, \mathrm{b}_1)$, $(\mathrm{d}_1, \mathrm{d}_1)$, $(\mathrm{d}_2, \mathrm{d}_2)$, $(\mathrm{a}_3, \mathrm{b}_3)$, and $(\mathrm{d}_3, \mathrm{d}_3)$, as illustrated by purple in the top alignment of Figure 6.1. Since there are a total of 6 aligned letters in the ground-truth alignment, we have that the Q score is $\frac{5}{6}$, or in other words, $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right) = \frac{5}{6}$.

When $\rho[2]$ shifts to the next-smallest interval $\left(\frac{1}{4}, \frac{1}{2}\right]$, the indel penalty $\rho[2]$ is sufficiently small that the $\mathrm{b}_1$ characters will align. Thus we lose the correct alignment $(\mathrm{a}_1, \mathrm{b}_1)$, and the Q score drops to $\frac{4}{6}$. Similarly, if we decrease $\rho[2]$ to the next-smallest interval $\left(\frac{1}{6}, \frac{1}{4}\right]$, the $\mathrm{b}_2$ characters will align, which is correct under the ground-truth alignment (Equation (6.3)). Thus the Q score increases back to $\frac{5}{6}$. Finally, by the same logic, when $\rho[2] \leq \frac{1}{6}$, we lose the correct alignment $(\mathrm{a}_3, \mathrm{b}_3)$ in favor of the alignment of the $\mathrm{b}_3$ characters, so the Q score falls to $\frac{4}{6}$. In this way, we prove the form of $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right)$ from Equation (6.4). A parallel argument proves the form of $u_{(0,\rho[2],0)}\left(S_1^{(2)}, S_2^{(2)}\right)$.

**Generalization to shattering $\Theta(\log n)$ sequence pairs.** This proof intuition naturally generalizes to $\Theta(\log n)$ sequence pairs of length $O(n)$ by expanding the number of subsequences $t_i^{(j)}$ *a la* Equation (6.2). In essence, if we define $S_1^{(1)} = t_1^{(1)} t_1^{(2)} \cdots t_1^{(k)}$ and $S_2^{(1)} = t_2^{(1)} t_2^{(2)} \cdots t_2^{(k)}$ for a carefully-chosen $k = \Theta\left(\sqrt{n}\right)$, then we can force $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right)$ to oscillate $O(n)$ times. Similarly, if we define $S_1^{(2)} = t_1^{(2)} t_1^{(4)} \cdots t_1^{(k-1)}$ and $S_2^{(2)} = t_2^{(2)} t_2^{(4)} \cdots t_2^{(k-1)}$, then we can force $u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right)$ to oscillate half as many times, and so on. This construction allows us to shatter $\Theta(\log n)$ sequences. □

With this intuition in mind, we now provide the full proof of Theorem 6.1.6.

*Proof of Theorem 6.1.6.* To prove this theorem, we identify:

1. An alphabet $\Sigma$,
2. A set of $N = \Theta(\log n)$ sequence pairs $\left(S_1^{(1)}, S_2^{(1)}\right), \ldots, \left(S_1^{(N)}, S_2^{(N)}\right) \in \cup_{i=1}^{n} \Sigma^i \times \Sigma^i$,
3. A ground-truth alignment $L_*^{(i)}$ for each sequence pair $\left(S_1^{(i)}, S_2^{(i)}\right)$, and
4. A set of $N$ witnesses $z_1, \ldots, z_N \in \mathbb{R}$ such that for any subset $T \subseteq [N]$, there exists an indel penalty parameter $\rho[T]$ such that if $i \in [T]$, then $u_{0,\rho[T],0}\left(S_1^{(i)}, S_2^{(i)}\right) < z_i$ and if $i \notin [T]$, then $u_{0,\rho[T],0}\left(S_1^{(i)}, S_2^{(i)}\right) \geq z_i$.

We now describe each of these four elements in turn.

**The alphabet $\Sigma$.** Let $k = 2^{\lfloor \log \sqrt{n/2} \rfloor} - 1 = \Theta(\sqrt{n})$. The alphabet $\Sigma$ consists of $4k$ characters[1] we denote as $\{\mathrm{a}_i, \mathrm{b}_i, \mathrm{c}_i, \mathrm{d}_i\}_{i=1}^{k}$.

**The set of $N = \Theta(\log n)$ sequence pairs.** These $N$ sequence pairs are defined by a set of $k$ subsequence pairs $\left(t_1^{(1)}, t_2^{(1)}\right), \ldots, \left(t_1^{(k)}, t_2^{(k)}\right) \in \Sigma^* \times \Sigma^*$. Each pair $\left(t_1^{(i)}, t_2^{(i)}\right)$ is defined as follows:

- The subsequence $t_1^{(i)}$ begins with $i$ $\mathrm{a}_i$s followed by $\mathrm{b}_i\mathrm{d}_i$. For example, $t_1^{(3)} = \mathrm{a}_3\mathrm{a}_3\mathrm{a}_3\mathrm{b}_3\mathrm{d}_3$.
- The subsequence $t_2^{(i)}$ begins with 1 $\mathrm{b}_i$, followed by $i$ $\mathrm{c}_i$s, followed by 1 $\mathrm{d}_i$. For example, $t_2^{(3)} = \mathrm{b}_3\mathrm{c}_3\mathrm{c}_3\mathrm{c}_3\mathrm{d}_3$.

Therefore, $t_1^{(i)}$ and $t_2^{(i)}$ are both of length $i+2$.

We use these subsequence pairs to define a set of $N = \log(k+1) = \Theta(\log n)$ sequence pairs. The first sequence pair, $\left(S_1^{(1)}, S_2^{(1)}\right)$ is defined as follows: $S_1^{(1)}$ is the concatenation of all subsequences $t_1^{(1)}, \ldots, t_1^{(k)}$ and $S_2^{(1)}$ is the concatenation of all subsequences $t_2^{(1)}, \ldots, t_2^{(k)}$:

$$S_1^{(1)} = t_1^{(1)} t_1^{(2)} t_1^{(3)} \cdots t_1^{(k)} \quad \text{and} \quad S_2^{(1)} = t_2^{(1)} t_2^{(2)} t_2^{(3)} \cdots t_2^{(k)}.$$

Next, $S_1^{(2)}$ and $S_2^{(2)}$ are the concatenation of every $2^{nd}$ subsequence:

$$S_1^{(2)} = t_1^{(2)} t_1^{(4)} t_1^{(6)} \cdots t_1^{(k-1)} \quad \text{and} \quad S_2^{(2)} = t_2^{(2)} t_2^{(4)} t_2^{(6)} \cdots t_2^{(k-1)}.$$

Similarly, $S_1^{(3)}$ and $S_2^{(3)}$ are the concatenation of every $4^{th}$ subsequence:

$$S_1^{(3)} = t_1^{(4)} t_1^{(8)} t_1^{(12)} \cdots t_1^{(k-3)} \quad \text{and} \quad S_2^{(3)} = t_2^{(4)} t_2^{(8)} t_2^{(12)} \cdots t_2^{(k-3)}.$$

Generally speaking, $S_1^{(i)}$ and $S_2^{(i)}$ are the concatenation of every $\left(2^{i-1}\right)^{th}$ subsequence:

$$S_1^{(i)} = t_1^{(2^{i-1})} t_1^{(2\cdot 2^{i-1})} t_1^{(3\cdot 2^{i-1})} \cdots t_1^{(k+1-2^{i-1})} \quad \text{and} \quad S_2^{(i)} = t_2^{(2^{i-1})} t_2^{(2\cdot 2^{i-1})} t_2^{(3\cdot 2^{i-1})} \cdots t_2^{(k+1-2^{i-1})}.$$

To explain the index of the last subsequence of every pair, since $k+1$ is a power of two, we know that $k-1$ is divisible by 2, $k-3$ is divisible by 4, and more generally, $k+1-2^{i-1}$ is divisible by $2^{i-1}$.

We claim that there are a total of $N = \log(k+1)$ such sequence pairs. To see why, note that each sequence in the first pair $S_1^{(1)}$ and $S_2^{(1)}$ consists of $k$ subsequences. Each sequence in the second pair $S_1^{(2)}$ and $S_2^{(2)}$ consists of $\frac{k-1}{2}$ subsequences. More generally, each sequence in the $i^{th}$ pair $S_1^{\left(k+1-2^{i-1}\right)}$ and $S_2^{\left(k+1-2^{i-1}\right)}$ consists of $\frac{k+1-2^{i-1}}{2^{i-1}}$ subsequences. The final pair will will consist of only one subsequence, so $\frac{k+1-2^{i-1}}{2^{i-1}} = 1$, or in other words $i = \log(k+1)$.

We also claim that each sequence has length at most $n$. This is because the longest sequence pair is the first, $\left(S_1^{(1)}, S_2^{(1)}\right)$. By definition of the subsequences $t_j^{(i)}$, these two sequences are of length $\sum_{i=1}^{k}(i+2) = \frac{1}{2}k(k+5) \leq n$. Therefore, all $N$ sequence pairs are of length at most $n$.

[1]To simplify the proof, we use this alphabet of size $4k$, but we believe it is possible to adapt this proof to handle the case where there are only 4 characters in the alphabet.

**Example 6.1.7.** Suppose that $n = 128$. Then $k = 2^{\lfloor \log \sqrt{n/2} \rfloor} - 1 = 7$ and $N = \log(k+1) = 3$. The three sequence pairs have the following form[2]:

$$S_1^{(1)} = \mathrm{a_1b_1d_1a_2a_2b_2d_2a_3a_3a_3b_3d_3a_4a_4a_4a_4b_4d_4a_5a_5a_5a_5b_5d_5a_6a_6a_6a_6a_6a_6b_6d_6a_7a_7a_7a_7a_7a_7b_7d_7}$$
$$S_2^{(1)} = \mathrm{b_1c_1d_1b_2c_2c_2d_2b_3c_3c_3c_3d_3b_4c_4c_4c_4d_4b_5c_5c_5c_5c_5d_5b_6c_6c_6c_6c_6c_6d_6b_7c_7c_7c_7c_7c_7d_7}$$
$$S_1^{(2)} = \mathrm{a_2a_2b_2d_2a_4a_4a_4a_4b_4d_4a_6a_6a_6a_6a_6a_6b_6d_6}$$
$$S_2^{(2)} = \mathrm{b_2c_2c_2d_2b_4c_4c_4c_4c_4d_4b_6c_6c_6c_6c_6c_6c_6d_6}$$
$$S_1^{(3)} = \mathrm{a_4a_4a_4a_4b_4d_4}$$
$$S_2^{(3)} = \mathrm{b_4c_4c_4c_4c_4d_4}$$

**A ground-truth alignment for every sequence pair.** To define a ground-truth alignment for all $N$ sequence pairs, we first define two alignments per subsequence pair $\left(t_1^{(i)}, t_2^{(i)}\right)$. The resulting ground-truth alignments will be a concatenation of these alignments. The first alignment, which we denote as $\left(h_1^{(i)}, h_2^{(i)}\right)$, is defined as follows: $h_1^{(i)}$ begins with $i$ $\mathrm{a}_i$s, followed by 1 $\mathrm{b}_i$, followed by $i$ gap characters, followed by 1 $\mathrm{d}_i$; $h_2^{(i)}$ begins with $i$ gap characters, followed by 1 $\mathrm{b}_i$, followed by $i$ $\mathrm{c}_i$s, followed by 1 $\mathrm{d}_i$. For example,

$$\begin{array}{rcccccccc} h_1^{(3)} = & \mathrm{a_3} & \mathrm{a_3} & \mathrm{a_3} & \mathrm{b_3} & \text{-} & \text{-} & \text{-} & \mathrm{d_3} \\ h_2^{(3)} = & \text{-} & \text{-} & \text{-} & \mathrm{b_3} & \mathrm{c_3} & \mathrm{c_3} & \mathrm{c_3} & \mathrm{d_3} \end{array}.$$

The second alignment, which we denote as $\left(\ell_1^{(i)}, \ell_2^{(i)}\right)$, is defined as follows: $\ell_1^{(i)}$ begins with $i$ $\mathrm{a}_i$s, followed by 1 $\mathrm{b}_i$, followed by $i$ gap characters, followed by 1 $\mathrm{d}_i$; $\ell_2^{(i)}$ begins with 1 $\mathrm{b}_i$, followed by $i$ gap characters, followed by $i$ $\mathrm{c}_i$s, followed by 1 $\mathrm{d}_i$. For example,

$$\begin{array}{rcccccccc} \ell_1^{(3)} = & \mathrm{a_3} & \mathrm{a_3} & \mathrm{a_3} & \mathrm{b_3} & \text{-} & \text{-} & \text{-} & \mathrm{d_3} \\ \ell_2^{(3)} = & \mathrm{b_3} & \text{-} & \text{-} & \text{-} & \mathrm{c_3} & \mathrm{c_3} & \mathrm{c_3} & \mathrm{d_3} \end{array}.$$

We now use these $2k$ alignments to define a ground-truth alignment $L_*^{(i)}$ per sequence pair $\left(S_1^{(i)}, S_2^{(i)}\right)$. Beginning with the first pair $\left(S_1^{(1)}, S_2^{(1)}\right)$, where

$$S_1^{(1)} = t_1^{(1)} t_1^{(2)} t_1^{(3)} \cdots t_1^{(k)} \quad \text{and} \quad S_2^{(1)} = t_2^{(1)} t_2^{(2)} t_2^{(3)} \cdots t_2^{(k)},$$

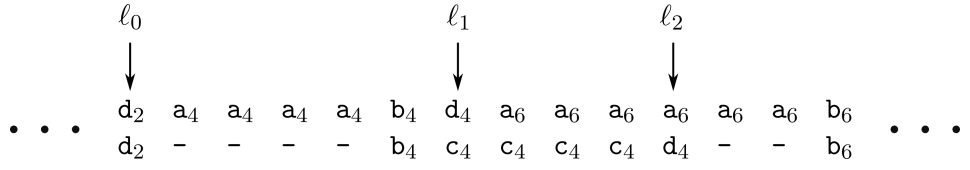
we define the alignment of $S_1^{(1)}$ to be $\ell_1^{(1)} h_1^{(2)} \ell_1^{(3)} h_1^{(4)} \cdots \ell_1^{(k)}$ and we define the alignment of $S_2^{(1)}$ to be $\ell_2^{(1)} h_2^{(2)} \ell_2^{(3)} h_2^{(4)} \cdots \ell_2^{(k)}$. Moving on to the second pair $\left(S_1^{(2)}, S_2^{(2)}\right)$, where

$$S_1^{(2)} = t_1^{(2)} t_1^{(4)} t_1^{(6)} \cdots t_1^{(k-1)} \quad \text{and} \quad S_2^{(2)} = t_2^{(2)} t_2^{(4)} t_2^{(6)} \cdots t_2^{(k-1)},$$

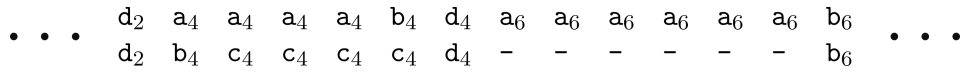
we define the alignment of $S_1^{(2)}$ to be $\ell_1^{(2)} h_1^{(4)} \ell_1^{(6)} h_1^{(8)} \cdots \ell_1^{(k-1)}$ and we define the alignment of $S_2^{(1)}$ to be $\ell_2^{(2)} h_2^{(4)} \ell_2^{(6)} h_2^{(8)} \cdots \ell_2^{(k-1)}$. Generally speaking, each pair $\left(S_1^{(i)}, S_2^{(i)}\right)$, where

$$S_1^{(i)} = t_1^{(2^{i-1})} t_1^{(2 \cdot 2^{i-1})} t_1^{(3 \cdot 2^{i-1})} \cdots t_1^{(k-2^{i-1}+1)} \quad \text{and} \quad S_2^{(i)} = t_2^{(2^{i-1})} t_2^{(2 \cdot 2^{i-1})} t_2^{(3 \cdot 2^{i-1})} \cdots t_2^{(k-2^{i-1}+1)},$$

[2]The maximum length of these six strings is 42, which is smaller than 128, as required.




(a) An initial alignment where the $\mathrm{d}_j$ characters are not aligned.




(b) An alignment where the gap characters in Figure 6.2a are shifted such that the $\mathrm{d}_j$ characters are aligned. The objective function value of both alignments is the same.

Figure 6.2: Illustration of Claim 6.1.8: we can assume that each $\mathrm{d}_j$ character in $S_1^{(i)}$ is matched to $\mathrm{d}_j$ in $S_2^{(i)}$.

is made up of $\frac{k+1}{2^{i-1}} - 1$ subsequences. Since $k+1$ is a power of two, this number of subsequences is odd. We define the alignment of $S_1^{(i)}$ to alternate between alignments of type $\ell_1^{(j)}$ and alignments of type $h_1^{(j')}$, beginning and ending with alignments of the first type. Specifically, the alignment $S_1^{(i)}$ is $\ell_1^{(2^{i-1})} h_1^{(2\cdot 2^{i-1})} \ell_1^{(3\cdot 2^{i-1})} h_1^{(4\cdot 2^{i-1})} \cdots \ell_1^{(k-2^{i-1}+1)}$. Similarly, we define the alignment of $S_2^{(i)}$ to be

$$\ell_2^{(2^{i-1})} h_2^{(2\cdot 2^{i-1})} \ell_2^{(3\cdot 2^{i-1})} h_2^{(4\cdot 2^{i-1})} \cdots \ell_2^{(k-2^{i-1}+1)}.$$

**The $N$ witnesses.** We define the $N$ values that witness the shattering of these $N$ sequence pairs to be $z_1 = z_2 = \cdots = z_N = \frac{3}{4}$.

**Shattering the $N$ sequence pairs.** Our goal is to show that for any subset $T \subseteq [N]$, there exists an indel penalty parameter $\rho[T]$ such that if $i \in [T]$, then $u_{0,\rho[T],0}\left(S_1^{(i)}, S_2^{(i)}\right) < \frac{3}{4}$ and if $i \notin [T]$, then $u_{0,\rho[T],0}\left(S_1^{(i)}, S_2^{(i)}\right) \geq \frac{3}{4}$. To prove this, we will use two helpful claims, Claims 6.1.8 and 6.1.9.

**Claim 6.1.8.** *For any pair $\left(S_1^{(i)}, S_2^{(i)}\right)$ and indel parameter $\rho[2] \geq 0$, there exists an alignment*

$$L \in \operatorname{argmax}_{L'} \textsc{mt}\left(S_1^{(i)}, S_2^{(i)}, L'\right) - \rho[2] \cdot \textsc{id}\left(S_1^{(i)}, S_2^{(i)}, L'\right)$$

*such that each $\mathrm{d}_j$ character in $S_1^{(i)}$ is matched to $\mathrm{d}_j$ in $S_2^{(i)}$.*

*Proof.* Let $L_0 \in \operatorname{argmax}_{L'} \textsc{mt}\left(S_1^{(i)}, S_2^{(i)}, L'\right) - \rho[2] \cdot \textsc{id}\left(S_1^{(i)}, S_2^{(i)}, L'\right)$ be an alignment such that some $\mathrm{d}_j$ character in $S_1^{(i)}$ is not matched to $\mathrm{d}_j$ in $S_2^{(i)}$. Denote the alignment $L_0$ as $(\tau_1, \tau_2)$. Let $j \in \mathbb{Z}$ be the smallest integer such that for some indices $\ell_1 \neq \ell_2$, $\tau_1[\ell_1] = \mathrm{d}_j$ and $\tau_2[\ell_2] = \mathrm{d}_j$. Next, let $\ell_0$ be the maximum index smaller than $\ell_1$ and $\ell_2$ such that $\tau_1[\ell_0] = \mathrm{d}_{j'}$ for some $j' \neq j$. We illustrate $\ell_0, \ell_1$, and $\ell_2$ in Figure 6.2a. By definition of $j$, we know that $\tau_1[\ell_0] = \tau_2[\ell_0] = \mathrm{d}_{j'}$. We also know there is at least one gap character in $\{\tau_1[i] : \ell_0 + 1 \leq i \leq \ell_1\} \cup \{\tau_2[i] : \ell_0 + 1 \leq i \leq \ell_2\}$ because otherwise, the $\mathrm{d}_j$ characters would be aligned in $L_0$. Moreover, we know there is at

most one match among these elements between characters other than $\mathrm{d}_j$ (namely, between the character $\mathrm{b}_j$). If we rearrange all of these gap characters so that they fall directly after $\mathrm{d}_j$ in both sequences, as in Figure 6.2b, then we may lose the match between the character $\mathrm{b}_j$, but we will gain the match between the character $\mathrm{d}_j$. Moreover, the number of indels remains the same, and all matches in the remainder of the alignment will remain unchanged. Therefore, this rearranged alignment has at least as high an objective function value as $L_0$, so the claim holds. □

Based on this claim, we will assume, without loss of generality, that for any pair $\left(S_1^{(i)}, S_2^{(i)}\right)$ and indel parameter $\rho[2] \geq 0$, under the alignment $L = A_{0,\rho[2],0}\left(S_1^{(i)}, S_2^{(i)}\right)$ returned by the algorithm $A_{0,\rho[2],0}$, all $\mathrm{d}_j$ characters in $S_1^{(i)}$ are matched to $\mathrm{d}_j$ in $S_2^{(i)}$.

**Claim 6.1.9.** *Suppose that the character $\mathrm{b}_j$ is in $S_1^{(i)}$ and $S_2^{(i)}$. The $\mathrm{b}_j$ characters will be matched in $L = A_{0,\rho[2],0}\left(S_1^{(i)}, S_2^{(i)}\right)$ if and only if $\rho[2] \leq \frac{1}{2j}$.*

*Proof.* Since all $\mathrm{d}_j$ characters are matched in $L$, in order to match $\mathrm{b}_j$, it is necessary to add exactly $2j$ gap characters: all $2j$ $\mathrm{a}_j$ and $\mathrm{c}_j$ characters must be matched with gap characters. Under the objective function $\textsc{mt}\left(S_1^{(i)}, S_2^{(i)}, L'\right) - \rho[2] \cdot \textsc{id}\left(S_1^{(i)}, S_2^{(i)}, L'\right)$, this one match will be worth the $2j\rho[2]$ penalty if and only if $1 \geq 2j\rho[2]$, as claimed. □

We now use Claims 6.1.8 and 6.1.9 to prove that we can shatter the $N$ sequence pairs $\left(S_1^{(1)}, S_2^{(1)}\right), \ldots, \left(S_1^{(N)}, S_2^{(N)}\right)$.

**Claim 6.1.10.** *There are $\frac{k+1}{2^{i-1}} - 1$ thresholds $\frac{1}{2(k+1)-2^i} < \frac{1}{2(k+1)-2\cdot 2^i} < \frac{1}{2(k+1)-3\cdot 2^i} < \cdots < \frac{1}{2^i}$ such that as $\rho[2]$ ranges from 0 to 1, when $\rho[2]$ crosses one of these thresholds, $u_{0,\rho[2],0}\left(S_1^{(i)}, S_2^{(i)}\right)$ switches from above $\frac{3}{4}$ to below $\frac{3}{4}$, or vice versa, beginning with $u_{0,0,0}\left(S_1^{(i)}, S_2^{(i)}\right) < \frac{3}{4}$ and ending with $u_{0,1,0}\left(S_1^{(i)}, S_2^{(i)}\right) > \frac{3}{4}$.*

*Proof.* Recall that

$$S_1^{(i)} = t_1^{(2^{i-1})} t_1^{(2\cdot 2^{i-1})} t_1^{(3\cdot 2^{i-1})} \cdots t_1^{(k-2^{i-1}+1)} \quad \text{and} \quad S_2^{(i)} = t_2^{(2^{i-1})} t_2^{(2\cdot 2^{i-1})} t_2^{(3\cdot 2^{i-1})} \cdots t_2^{(k-2^{i-1}+1)},$$

so in $S_1^{(i)}$ and $S_2^{(i)}$, the $\mathrm{b}_j$ characters are $\mathrm{b}_{2^{i-1}}, \mathrm{b}_{2\cdot 2^{i-1}}, \mathrm{b}_{3\cdot 2^{i-1}}, \ldots, \mathrm{b}_{k-2^{i-1}+1}$. Also, the reference alignment of $S_1^{(i)}$ is $\ell_1^{(2^{i-1})} h_1^{(2\cdot 2^{i-1})} \ell_1^{(3\cdot 2^{i-1})} h_1^{(4\cdot 2^{i-1})} \cdots \ell_1^{(k-2^{i-1}+1)}$ and the reference alignment of $S_2^{(i)}$ is

$$\ell_2^{(2^{i-1})} h_2^{(2\cdot 2^{i-1})} \ell_2^{(3\cdot 2^{i-1})} h_2^{(4\cdot 2^{i-1})} \cdots \ell_2^{(k-2^{i-1}+1)}.$$

We know that when the indel penalty $\rho[2]$ is equal to zero, all $\mathrm{d}_j$ characters will be aligned, as will all $\mathrm{b}_j$ characters. This means we will correctly align all $\mathrm{d}_j$ characters and we will correctly align all $\mathrm{b}_j$ characters in the $\left(h_1^{(j)}, h_2^{(j)}\right)$ pairs, but we will incorrectly align the $\mathrm{b}_j$ characters in the $\left(\ell_1^{(j)}, \ell_2^{(j)}\right)$ pairs. The number of $\left(h_1^{(j)}, h_2^{(j)}\right)$ pairs in this reference alignment is $\frac{k+1}{2^i} - 1$ and

the number of $\left(\ell_1^{(j)}, \ell_2^{(j)}\right)$ pairs is $\frac{k+1}{2^i}$. Therefore, the utility of the alignment that maximizes the number of matches equals the following:

$$u_{0,0,0}\left(S_1^{(i)}, S_2^{(i)}\right) = \frac{\frac{k+1}{2^{i-1}} - 1 + \frac{k+1}{2^i} - 1}{\frac{k+1}{2^{i-2}} - 2} = \frac{3(k+1) - 2^{i+1}}{4(k+1) - 2^{i+1}} < \frac{3}{4},$$

where the final inequality holds because $2^{i+1} \leq 2(k+1) < 3(k+1)$.

Next, suppose we increase $\rho[2]$ to lie in the interval $\left(\frac{1}{2(k+1)-2^i}, \frac{1}{2(k+1)-2\cdot 2^i}\right]$. Since it is no longer the case that $\rho[2] \leq \frac{1}{2(k-2^{i-1}+1)}$, we know that the $\mathrm{b}_{k-2^{i-1}+1}$ characters will no longer be matched, and thus we will correctly align this character according to the reference alignment. This means we will correctly align all $\mathrm{d}_j$ characters and we will correctly align all $\mathrm{b}_j$ characters in the $\left(h_1^{(j)}, h_2^{(j)}\right)$ pairs, but we will incorrectly align all but one of the $\mathrm{b}_j$ characters in the $\left(\ell_1^{(j)}, \ell_2^{(j)}\right)$ pairs. Therefore, the utility of the alignment that maximizes $\textsc{mt}(S_1, S_2, L) - \rho[2] \cdot \textsc{id}(S_1, S_2, L)$ is

$$u_{0,\rho[2],0}\left(S_1^{(i)}, S_2^{(i)}\right) = \frac{\frac{k+1}{2^{i-1}} - 1 + \frac{k+1}{2^i}}{\frac{k+1}{2^{i-2}} - 2} = \frac{3(k+1) - 2^i}{4(k+1) - 2^{i+1}} > \frac{3}{4},$$

where the final inequality holds because $2^{i+1} \leq 2(k+1) < 4(k+1)$.

Next, suppose we increase $\rho[2]$ to lie in the interval $\left(\frac{1}{2(k+1)-2\cdot 2^i}, \frac{1}{2(k+1)-3\cdot 2^i}\right]$. Since it is no longer the case that $\rho[2] \leq \frac{1}{2(k-2\cdot 2^{i-1}+1)}$, we know that the $\mathrm{b}_{k-2\cdot 2^{i-1}+1}$ characters will no longer be matched, and thus we will incorrectly align this character according to the reference alignment. This means we will correctly align all $\mathrm{d}_j$ characters and we will correctly align the $\mathrm{b}_j$ characters in all but one of the $\left(h_1^{(j)}, h_2^{(j)}\right)$ pairs, but we will incorrectly align all but one of the $\mathrm{b}_j$ characters in the $\left(\ell_1^{(j)}, \ell_2^{(j)}\right)$ pairs. Therefore, the utility of the alignment that maximizes $\textsc{mt}(S_1, S_2, L) - \rho[2] \cdot \textsc{id}(S_1, S_2, L)$ is

$$u_{0,\rho[2],0}\left(S_1^{(i)}, S_2^{(i)}\right) = \frac{\frac{k+1}{2^{i-1}} - 1 + \frac{k+1}{2^i}}{\frac{k+1}{2^{i-2}} - 2} > \frac{3}{4}.$$

In a similar fashion, every time $\rho[2]$ crosses one of the thresholds $\frac{1}{2(k+1)-2^i} < \frac{1}{2(k+1)-2\cdot 2^i} < \frac{1}{2(k+1)-3\cdot 2^i} < \cdots < \frac{1}{2^i}$, the utility will shift from above $\frac{3}{4}$ to below or vice versa, as claimed. □

The above claim demonstrates that the $N$ sequence pairs are shattered, each with the witness $\frac{3}{4}$. After all, for every $i \in \{2, \ldots, N\}$ and every interval $\left(\frac{1}{2(k+1)-j2^i}, \frac{1}{2(k+1)-(j+1)2^i}\right)$ where $u_{0,\rho[2],0}\left(S_1^{(i)}, S_2^{(i)}\right)$ is uniformly above or below $\frac{3}{4}$, there exists a subpartition of this interval into the two intervals

$$\left(\frac{1}{2(k+1)-j2^i}, \frac{1}{2(k+1)-(2j+1)2^{i-1}}\right) \text{ and } \left(\frac{1}{2(k+1)-(2j+1)2^{i-1}}, \frac{1}{2(k+1)-(j+1)2^i}\right)$$

such that in the first interval, $u_{0,\rho[2],0}\left(S_1^{(i-1)}, S_2^{(i-1)}\right) < \frac{3}{4}$ and in the second,

$$u_{0,\rho[2],0}\left(S_1^{(i-1)}, S_2^{(i-1)}\right) > \frac{3}{4}.$$

Therefore, for any subset $T \subseteq [N]$, there exists an indel penalty parameter $\rho[T]$ such that if $i \in [T]$, then $u_{0,\rho[T],0}\left(S_1^{(i)}, S_2^{(i)}\right) < \frac{3}{4}$ and if $i \notin [T]$, then $u_{0,\rho[T],0}\left(S_1^{(i)}, S_2^{(i)}\right) > \frac{3}{4}$. □

## 6.2 RNA folding

RNA molecules have many essential roles including protein coding and enzymatic functions Holley et al. [1965]. RNA is assembled as a chain of *bases* denoted A, U, C, and G. It is often found as a single strand folded onto itself with non-adjacent bases physically bound together. RNA folding algorithms infer the way strands would naturally fold, shedding light on their functions. Given a sequence $S \in \{\mathrm{A}, \mathrm{U}, \mathrm{C}, \mathrm{G}\}^n$, we represent a folding by a set of pairs $\phi \subset [n] \times [n]$. If $(i,j) \in \phi$, then the $i^{th}$ and $j^{th}$ bases of $S$ bind together. Typically, the bases A and U bind together, as do C and G. Other matchings are likely less stable. We assume that the foldings do not contain any *pseudoknots*, which are pairs $(i,j),(i',j')$ that cross with $i < i' < j < j'$.

A well-studied algorithm returns a folding that maximizes a parameterized objective function Nussinov and Jacobson [1980]. At a high level, this objective function trades off between global properties of the folding (the number of binding pairs $|\phi|$) and local properties (the likelihood that bases would appear close together in the folding). Specifically, the algorithm $A_\rho$ uses dynamic programming to return the folding $A_\rho(S)$ that maximizes

$$\rho\,|\phi| + (1-\rho) \sum_{(i,j)\in\phi} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi\}}, \tag{6.5}$$

where $\rho \in [0,1]$ is a parameter and $M_{S[i],S[j],S[i-1],S[j+1]} \in \mathbb{R}$ is a score for having neighboring pairs of the letters $(S[i],S[j])$ and $(S[i-1],S[j+1])$. These scores help identify stable substructures.

We assume there is a utility function that characterizes a folding's quality, denoted $u(S,\phi)$. For example, $u(S,\phi)$ might measure the fraction of pairs shared between $\phi$ and a "ground-truth" folding, obtained via expensive computation or laboratory experiments. We then define $u_\rho(S) = u\left(S, A_\rho(S)\right)$ to be the utility of the folding returned by the algorithm $A_\rho$.

**Lemma 6.2.1.** *For any RNA sequence $S \in \Sigma^n$, let $u_S : [0,1] \to \mathbb{R}$ denote the function $u_S(\rho) = u_\rho(S)$. There is a set $T \leq \binom{|\Phi|}{2} \leq n^2$ intervals $[\rho_1,\rho_2),[\rho_2,\rho_3),\ldots,[\rho_T,\rho_{T+1}]$ with $\rho_1 := 0 < \rho_2 < \cdots < \rho_T < 1 := \rho_{T+1}$ such that for any one interval $I$, across all $\rho \in I$, $u_S(\rho)$ is constant.*

*Proof.* Fix a sequence $S$. Let $\Phi$ be the set of alignments that the algorithm returns as we range over all parameters $\rho \in \mathbb{R}$. In other words, $\Phi = \{A_\rho(S) \mid \rho \in [0,1]\}$. We know that every folding has length at most $n/2$. For any $k \in \{0,\ldots,n/2\}$, let $\phi_k$ be the folding of length $k$ that maximizes the right-hand-side of Equation (6.5):

$$\phi_k = \mathrm{argmax}_{\phi:|\phi|=k} \sum_{(i,j)\in\phi} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi\}}.$$

The folding the algorithm returns will always be one of $\{\phi_0,\ldots,\phi_{n/2}\}$, so $|\Phi| \leq \frac{n}{2}+1$.

Fix an arbitrary folding $\phi \in \Phi$. We know that $\phi$ will be the folding returned by the algorithm $A_\rho(S)$ if and only if

$$\begin{aligned} &\rho\,|\phi| + (1-\rho) \sum_{(i,j)\in\phi} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi\}} \\ \geq\ &\rho\,|\phi'| + (1-\rho) \sum_{(i,j)\in\phi'} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi'\}} \end{aligned} \tag{6.6}$$

for all $\phi' \in \Phi \setminus \{\phi\}$. Since these functions are linear in $\rho$, this means there is a set of $T \leq \binom{|\Phi|}{2} \leq n^2$ intervals $[\rho_1, \rho_2), [\rho_2, \rho_3), \ldots, [\rho_T, \rho_{T+1}]$ with $\rho_1 := 0 < \rho_2 < \cdots < \rho_T < 1 := \rho_{T+1}$ such that for any one interval $I$, across all $\rho \in I$, $A_\rho(S)$ is fixed. This means that for any one interval $[\rho_i, \rho_{i+1})$, there exists a real value $c_i$ such that $u_\rho(S) = c_i$ for all $\rho \in [\rho_i, \rho_{i+1})$. □

This fact implies the following pseudo-dimension bound. The proof is similar to that of Theorem 3.0.3 from Chapter 3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 6.2.2.** *The pseudo-dimension of* $\mathcal{U} = \{u_\rho : \rho \in [0,1]\}$ *is* $O(\ln n)$.

## 6.3 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. *Topologically associating domains (TADs)* are contiguous segments of the genome that fold into compact regions. More formally, given the genome length $n$, a TAD set is a set $T = \{(i_1, j_1), \ldots, (i_t, j_t)\} \subset [n] \times [n]$ such that $i_1 < j_1 < i_2 < j_2 < \cdots < i_t < j_t$. If $(i, j) \in T$, the bases within the corresponding substring physically interact more frequently with each other than with other bases. Disrupting TAD boundaries can affect the expression of nearby genes, which can trigger diseases such as congenital malformations and cancer Lupiáñez et al. [2016].

The contact frequency of any two genome locations, denoted by a matrix $M \in \mathbb{R}^{n \times n}$, can be measured via experiments Lieberman-Aiden et al. [2009]. A dynamic programming algorithm $A_\rho$ introduced by Filippova et al. [2014] returns the TAD set $A_\rho(M)$ that maximizes

$$\sum_{(i,j) \in T} s_\rho(i, j) - \mu_\rho(j - i), \tag{6.7}$$

where $\rho \geq 0$ is a 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 $s_\rho$ over all sub-matrices of length $d$ along the diagonal of $M$. We note that unlike the sequence alignment and RNA folding algorithms, the parameter $\rho$ appears in the exponent of the objective function.

We assume there is a utility function that characterizes the quality of a TAD set $T$, denoted $u(M, T) \in \mathbb{R}$. For example, $u(M, T)$ might measure the fraction of TADs in $T$ that are in the correct location with respect to a ground-truth TAD set. We then define $u_\rho(M) = u\left(M, A_\rho(M)\right)$ to be the utility of the TAD set returned by the algorithm $A_\rho$.

We will use the following corollary of Rolle's theorem to understand the structure of the utility functions in this section.

**Lemma 6.3.1** (Tossavainen [2006])**.** *Let $h$ be a polynomial-exponential sum of the form $h(x) = \sum_{i=1}^{t} a_i b_i^x$, where $b_i > 0$ and $a_i \in \mathbb{R}$. The number of roots of $h$ is upper bounded by $t$.*

Lemma 6.3.1 implies the following fact.

**Corollary 6.3.2.** *Let $h$ be a polynomial-exponential sum of the form*

$$h(x) = \sum_{i=1}^{t} \frac{a_i}{b_i^x},$$

*where $b_i > 0$ and $a_i \in \mathbb{R}$. The number of roots of $h$ is upper bounded by $t$.*

*Proof.* Note that $\sum_{i=1}^{t} \frac{a_i}{b_i^x} = 0$ if and only if

$$\left(\prod_{j=1}^{n} b_i^x\right) \sum_{i=1}^{t} \frac{a_i}{b_i^x} = \sum_{i=1}^{n} a_i \left(\prod_{j \neq i} b_i\right)^x = 0.$$

Therefore, the corollary follows from Lemma 6.3.1. □

We now show that the TAD algorithm's utility is a piecewise-constant function of its parameters.

**Lemma 6.3.3.** *For any matrix $M \in \mathbb{R}^{n \times n}$, let $u_M : \mathbb{R}_{\geq 0} \to \mathbb{R}$ denote the function $u_M(\rho) = u_\rho(M)$. There is a set $\tau \leq 2n^2 4^{n^2} + 1$ intervals $[\rho_1, \rho_2), [\rho_2, \rho_3), \ldots, [\rho_\tau, \rho_{\tau+1})$ with $\rho_1 := 0 < \rho_2 < \cdots < \rho_\tau < \infty := \rho_{\tau+1}$ such that for any one interval $I$, across all $\rho \in I$, $u_M(\rho)$ is constant.*

*Proof.* Fix a matrix $M$. We begin by rewriting Equation (6.7) as follows:

$$\begin{aligned} A_\rho(M) &= \operatorname*{argmax}_{T \subset [n] \times [n]} \sum_{(i,j) \in T} \left( \frac{1}{(j-i)^\rho} \left( \sum_{i \leq u < v \leq j} M_{uv} \right) - \frac{1}{n-j+i} \sum_{t=0}^{n-j+i} \frac{1}{(j-i)^\rho} \sum_{t \leq p < q \leq t+j-i} M_{pq} \right) \\ &= \operatorname*{argmax} \sum_{(i,j) \in T} \frac{1}{(j-i)^\rho} \left( \left( \sum_{i \leq u < v \leq j} M_{uv} \right) - \frac{1}{n-j+i} \sum_{t=0}^{n-j+i} \sum_{t \leq p < q \leq t+j-i} M_{pq} \right) \\ &= \operatorname*{argmax} \sum_{(i,j) \in T} \frac{c_{ij}}{(j-i)^\rho}, \end{aligned}$$

where

$$c_{ij} = \left( \sum_{i \leq u < v \leq j} M_{uv} \right) - \frac{1}{n-j+i} \sum_{t=0}^{n-j+i} \sum_{t \leq p < q \leq t+j-i} M_{pq}$$

is a constant that does not depend on $\rho$.

Let $\mathcal{T}$ be the set of TAD sets that the algorithm returns as we range over all parameters $\rho \geq 0$. In other words, $\mathcal{T} = \{A_\rho(M) \mid \rho \in \mathbb{R}_{\geq 0}\}$. Since each TAD set is a subset of $[n] \times [n]$, $|\mathcal{T}| \leq 2^{n^2}$. For any TAD set $T \in \mathcal{T}$, the algorithm $A_\rho$ will return $T$ if and only if

$$\sum_{(i,j) \in T} \frac{c_{ij}}{(j-i)^\rho} > \sum_{(i',j') \in T'} \frac{c_{i'j'}}{(j'-i')^\rho}$$

for all $T' \in \mathcal{T} \setminus \{T\}$. This means that as we range $\rho$ over the positive reals, the TAD set returned by algorithm $A_\rho(M)$ will only change when

$$\sum_{(i,j) \in T} \frac{c_{ij}}{(j-i)^\rho} - \sum_{(i',j') \in T'} \frac{c_{i'j'}}{(j'-i')^\rho} = 0 \tag{6.8}$$

for some $T, T' \in \mathcal{T}$. As a result of Rolle's Theorem (Corollary 6.3.2), we know that Equation (6.8) has at most $|T| + |T'| \leq 2n^2$ solutions. This means there are $t \leq 2n^2 \binom{|\mathcal{T}|}{2} \leq 2n^2 4^{n^2}$ intervals $[\rho_1, \rho_2), [\rho_2, \rho_3), \ldots, [\rho_t, \rho_{t+1})$ with $\rho_1 := 0 < \rho_2 < \cdots < \rho_t < \infty := \rho_{t+1}$ that partition $\mathbb{R}_{\geq 0}$ such that across all $\rho$ within any one interval $[\rho_i, \rho_{i+1})$, the TAD set returned by algorithm $A_\rho(M)$ is fixed. Therefore, there exists a real value $c_i$ such that $u_\rho(M) = c_i$ for all $\rho \in [\rho_i, \rho_{i+1})$. By definition, this means that $u_M(\rho) = u_\rho(M) = c_i$ as well. □

This fact implies the following pseudo-dimension bound. The proof is similar to that of Theorem 3.0.3 from Chapter 3, and our general theorem from Chapter 7 implies the pseudo-dimension bound as well.

**Theorem 6.3.4.** *The pseudo-dimension of* $\mathcal{U} = \{u_\rho : \rho \geq 0\}$ *is* $O(n^2)$.

# Chapter 7

# Pseudo-dimension bounds for piecewise-structured performance functions

In this section, we present a general theory which unifies our results so far in this thesis. Throughout this thesis, 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 4), 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 Lipschitz continuity 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*. This structure has also been observed in contexts beyond those studied in this thesis, including clustering and greedy algorithm configuration [Balcan et al., 2017, 2018c, 2020a, Gupta and Roughgarden, 2017], and in the context of online algorithm configuration [Balcan et al., 2018b, 2020b,c, Cohen-Addad and Kanade, 2017, Gupta and Roughgarden, 2017, as we describe in Section 7.1]. 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 thesis.

The results in this chapter are from a STOC'21 paper with Nina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, and Tuomas Sandholm [Balcan et al., 2021a].

## 7.1 Related research

Research published before the results in this chapter [Balcan et al., 2018b, Cohen-Addad and Kanade, 2017, Gupta and Roughgarden, 2017] as well as concurrent research [Balcan et al., 2020b,c] provides provable guarantees for online algorithm configuration. Regret minimization is impossible in the worst case [Gupta and Roughgarden, 2017], but prior research shows that under some conditions, it is possible. Gupta and Roughgarden [2017] study online algorithm configuration for the maximum weight independent set (MWIS) problem under a "smoothed adversary", where the adversary chooses a distribution per vertex from which its weight is drawn. They prove that their approach has small constant per-round regret. Cohen-Addad and Kanade [2017] provide no-regret algorithms for optimizing single-dimensional piecewise-constant functions under a smoothed adversary, where the adversary chooses a distribution per boundary from which that boundary is drawn. They show that this approach provides a no-regret algorithm for Gupta and Roughgarden's (2017) MWIS configuration problem under a smoothed adversary, as well as other greedy algorithm configuration problems defined by a single tunable parameter.

Balcan et al. [2018b, 2020b,c] provided no-regret algorithms for optimizing multi-dimensional piecewise-Lipschitz functions when the boundaries between pieces do not concentrate. Balcan et al. [2018b] also provided sample complexity bounds for the batch learning setting under the same assumptions. These papers proved that across many parameterized algorithms, the algorithm's performance is a piecewise-Lipschitz function of the parameters, and under mild assumptions, the boundaries do not concentrate. Therefore, their parameters can be optimized online in a no-regret fashion. Online learning guarantees imply sample complexity guarantees due to online-to-batch conversion, and Balcan et al. [2018b] also provide sample complexity guarantees based on dispersion using Rademacher complexity.

To prove that dispersion holds, one typically needs to show that under the distribution over problem instances, the dual functions' discontinuities do not concentrate. This argument is typically made by assuming that the distribution is sufficiently nice or—when applicable—by appealing to the random nature of the parameterized algorithm. Thus, for arbitrary distributions and deterministic algorithms, dispersion does not necessarily hold. In contrast, our results hold even when the discontinuities concentrate, and thus are applicable to a broader set of problems in the distributional learning model. In other words, the results from this chapter cannot be recovered using the techniques of prior research [Balcan et al., 2018b, 2020b,c, Cohen-Addad and Kanade, 2017, Gupta and Roughgarden, 2017]. The results in this chapter, however, only apply to batch learning, not online learning.

## 7.2 Generalization guarantees for data-driven algorithm design

In data-driven algorithm design, there are two closely related function classes. First, for each parameter setting $\boldsymbol{\rho} \in \mathcal{P}$, $u_{\boldsymbol{\rho}} : \mathcal{Z} \to \mathbb{R}$ measures performance as a function of the input $z \in \mathcal{Z}$. Similarly, for each input $z$, there is a function $u_z : \mathcal{P} \to \mathbb{R}$ defined as $u_z(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(z)$ that measures performance as a function of the parameter vector $\boldsymbol{\rho}$. The set $\{u_z \mid z \in \mathcal{Z}\}$ is equivalent to Assouad's notion of the *dual class* [Assouad, 1983].

**Definition 7.2.1** (Dual class [Assouad, 1983])**.** For any domain $\mathcal{Y}$ and set of functions $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{Y}}$, the *dual class* of $\mathcal{H}$ is defined as $\mathcal{H}^* = \left\{h_y^* : \mathcal{H} \to \mathbb{R} \mid y \in \mathcal{Y}\right\}$ where $h_y^*(h) = h(y)$. Each function

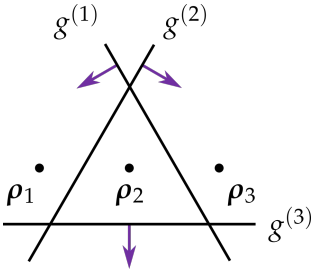



Figure 7.1: Boundary functions partitioning $\mathbb{R}^2$. The arrows indicate on which side of each function $g^{(i)}(\boldsymbol{\rho}) = 0$ and on which side $g^{(i)}(\boldsymbol{\rho}) = 1$. For example, $g^{(1)}(\boldsymbol{\rho}_1) = 1$, $g^{(1)}(\boldsymbol{\rho}_2) = 1$, and $g^{(1)}(\boldsymbol{\rho}_3) = 0$.

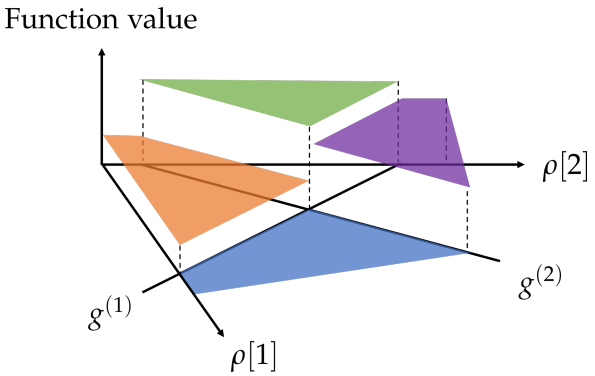



Figure 7.2: A piecewise-constant function over $\mathbb{R}^2_{\geq 0}$ with linear boundary functions $g^{(1)}$ and $g^{(2)}$.

$h^*_y \in \mathcal{H}^*$ fixes an input $y \in \mathcal{Y}$ and maps each function $h \in \mathcal{H}$ to $h(y)$. We refer to the class $\mathcal{H}$ as the *primal class.*

The set of functions $\{u_z \mid z \in \mathcal{Z}\}$ is equivalent to the dual class $\mathcal{U}^* = \{u^*_z : \mathcal{U} \to [0, H] \mid z \in \mathcal{Z}\}$ in the sense that for every parameter vector $\boldsymbol{\rho} \in \mathcal{P}$ and every problem instance $z \in \mathcal{Z}$, $u_z(\boldsymbol{\rho}) = u^*_z(u_{\boldsymbol{\rho}})$.

Many combinatorial algorithms share a clear-cut, useful structure: for each instance $z \in \mathcal{Z}$, the function $u_z$ is *piecewise structured*. For example, each function $u_z$ might be piecewise constant with a small number of pieces. Given the equivalence of the functions $\{u_z \mid z \in \mathcal{Z}\}$ and the dual class $\mathcal{U}^*$, the dual class exhibits this piecewise structure as well. We use this structure to bound the pseudo-dimension of the primal class $\mathcal{U}$.

Intuitively, a function $h : \mathcal{Y} \to \mathbb{R}$ is piecewise structured if we can partition the domain $\mathcal{Y}$ into subsets $\mathcal{Y}_1, \ldots, \mathcal{Y}_M$ so that when we restrict $h$ to one piece $\mathcal{Y}_i$, $h$ equals some well-structured function $f : \mathcal{Y} \to \mathbb{R}$. In other words, for all $y \in \mathcal{Y}_i$, $h(y) = f(y)$. We define the partition $\mathcal{Y}_1, \ldots, \mathcal{Y}_M$ using *boundary functions* $g^{(1)}, \ldots, g^{(k)} : \mathcal{Y} \to \{0, 1\}$. Each function $g^{(i)}$ divides the domain $\mathcal{Y}$ into two sets: the points it labels 0 and the points it labels 1. Figure 7.1 illustrates a partition of $\mathbb{R}^2$ by boundary functions. Together, the $k$ boundary functions partition the domain $\mathcal{Y}$ into at most $2^k$ regions, each one corresponding to a bit vector $\boldsymbol{b} \in \{0, 1\}^k$ that describes on which side of each boundary the region belongs. For each region, we specify a *piece function* $f_{\boldsymbol{b}} : \mathcal{Y} \to \mathbb{R}$ that defines the function values of $h$ restricted to that region. Figure 7.2 shows an example of a piecewise-structured function with two boundary functions and four piece

functions.

For many parameterized algorithms, every function in the dual class is piecewise structured. Moreover, across dual functions, the 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 captures this structure.

**Definition 7.2.2.** A function class $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{Y}}$ that maps a domain $\mathcal{Y}$ to $\mathbb{R}$ is $(\mathcal{F}, \mathcal{G}, k)$*-piecewise decomposable* for a class $\mathcal{G} \subseteq \{0,1\}^{\mathcal{Y}}$ of boundary functions and a class $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{Y}}$ of piece functions if the following holds: for every $h \in \mathcal{H}$, there are $k$ 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 $y \in \mathcal{Y}$, $h(y) = f_{\boldsymbol{b}_y}(y)$ where $\boldsymbol{b}_y = \left(g^{(1)}(y), \ldots, g^{(k)}(y)\right) \in \{0,1\}^k$.

Our main theorem shows that whenever the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable, we can bound the pseudo-dimension of $\mathcal{U}$ in terms of the VC-dimension of $\mathcal{G}^*$ and the pseudo-dimension of $\mathcal{F}^*$. Later, we show that for many common classes $\mathcal{F}$ and $\mathcal{G}$, we can easily bound the complexity of their duals.

**Theorem 7.2.3.** *Suppose that the dual function class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable with boundary functions $\mathcal{G} \subseteq \{0,1\}^{\mathcal{U}}$ and piece functions $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{U}}$. The pseudo-dimension of $\mathcal{U}$ is bounded as follows:*

$$\operatorname{Pdim}(\mathcal{U}) = O\left(\left(\operatorname{Pdim}(\mathcal{F}^*) + \operatorname{VCdim}(\mathcal{G}^*)\right) \ln\left(\operatorname{Pdim}(\mathcal{F}^*) + \operatorname{VCdim}(\mathcal{G}^*)\right) + \operatorname{VCdim}(\mathcal{G}^*) \ln k\right).$$

To help make the proof of Theorem 7.2.3 succinct, we extract a key insight in the following lemma. Given a set of functions $\mathcal{H}$ that map a domain $\mathcal{Y}$ to $\{0,1\}$, Lemma 7.2.4 bounds the number of binary vectors

$$(h_1(y), \ldots, h_N(y)) \tag{7.1}$$

we can obtain for any $N$ functions $h_1, \ldots, h_N \in \mathcal{H}$ as we vary the input $y \in \mathcal{Y}$. Pictorially, if we partition $\mathbb{R}^2$ using the functions $g^{(1)}$, $g^{(2)}$, and $g^{(3)}$ from Figure 7.1 for example, Lemma 7.2.4 bounds the number of regions in the partition. This bound depends not on the VC-dimension of the class $\mathcal{H}$, but rather on that of its dual $\mathcal{H}^*$. We use a classic lemma by Sauer [1972] to prove Lemma 7.2.4. Sauer's lemma [Sauer, 1972] bounds the number of binary vectors of the form $(h(y_1), \ldots, h(y_N))$ we can obtain for any $N$ elements $y_1, \ldots, y_N \in \mathcal{Y}$ as we vary the function $h \in \mathcal{H}$ by $(eN)^{\operatorname{VCdim}(\mathcal{H})}$. Therefore, it does not immediately imply a bound on the number of vectors from Equation (7.1). In order to apply Sauer's lemma, we must transition to the dual class.

**Lemma 7.2.4.** *Let $\mathcal{H}$ be a set of functions that map a domain $\mathcal{Y}$ to $\{0,1\}$. For any functions $h_1, \ldots, h_N \in \mathcal{H}$, the number of binary vectors $(h_1(y), \ldots, h_N(y))$ obtained by varying the input $y \in \mathcal{Y}$ is bounded as follows:*

$$\left|\left\{(h_1(y), \ldots, h_N(y)) \mid y \in \mathcal{Y}\right\}\right| \leq (eN)^{\operatorname{VCdim}(\mathcal{H}^*)}. \tag{7.2}$$

*Proof.* We rewrite the left-hand-side of Equation (7.2) as $\left|\left\{\left(h_y^*(h_1), \ldots, h_y^*(h_N)\right) \,\middle|\, y \in \mathcal{Y}\right\}\right|$. Since we fix $N$ inputs and vary the function $h_y^*$, the lemma follows from Sauer's lemma. □

We now prove Theorem 7.2.3.

*Proof of Theorem 7.2.3.* Fix an arbitrary set of problem instances $z_1, \ldots, z_N \in \mathcal{Z}$ and targets $t_1, \ldots, t_N \in \mathbb{R}$. We bound the number of ways that $\mathcal{U}$ can label the problem instances $z_1, \ldots, z_N$ with respect to the target thresholds $t_1, \ldots, t_N \in \mathbb{R}$. In other words we bound the size of the set

$$\left| \left\{ \begin{pmatrix} \operatorname{sign}\left(u_{\boldsymbol{\rho}}\left(z_1\right) - t_1\right) \\ \vdots \\ \operatorname{sign}\left(u_{\boldsymbol{\rho}}\left(z_N\right) - t_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P} \right\} \right| = \left| \left\{ \begin{pmatrix} \operatorname{sign}\left(u^*_{z_1}\left(u_{\boldsymbol{\rho}}\right) - t_1\right) \\ \vdots \\ \operatorname{sign}\left(u^*_{z_N}\left(u_{\boldsymbol{\rho}}\right) - t_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P} \right\} \right| \tag{7.3}$$

by $(ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$. Then solving for the largest $N$ such that

$$2^N \leq (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$$

gives a bound on the pseudo-dimension of $\mathcal{U}$. Our bound on Equation (7.3) has two main steps:

1. In Claim 7.2.5, we show that there are $M < (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}$ subsets $\mathcal{P}_1, \ldots, \mathcal{P}_M$ partitioning the parameter space $\mathcal{P}$ such that within any one subset, the dual functions $u^*_{z_1}, \ldots, u^*_{z_N}$ are simultaneously structured. In particular, for each subset $\mathcal{P}_j$, there exist piece functions $f_1, \ldots, f_N \in \mathcal{F}$ such that $u^*_{z_i}\left(u_{\boldsymbol{\rho}}\right) = f_i\left(u_{\boldsymbol{\rho}}\right)$ for all parameter settings $\boldsymbol{\rho} \in \mathcal{P}_j$ and $i \in [N]$. This is the partition of $\mathcal{P}$ induced by aggregating all of the boundary functions corresponding to the dual functions $u^*_{z_1}, \ldots, u^*_{z_N}$.

2. We then show that for any region $\mathcal{P}_j$ of the partition, as we vary the parameter vector $\boldsymbol{\rho} \in \mathcal{P}_j$, $u_{\boldsymbol{\rho}}$ can label the problem instances $z_1, \ldots, z_N$ in at most $(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$ ways with respect to the target thresholds $t_1, \ldots, t_N$. It follows that the total number of ways that $\mathcal{U}$ can label the problem instances $z_1, \ldots, z_N$ is bounded by $(ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$.

We now prove the first claim.

**Claim 7.2.5.** *There are $M < (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}$ subsets $\mathcal{P}_1, \ldots, \mathcal{P}_M$ partitioning the parameter space $\mathcal{P}$ such that within any one subset, the dual functions $u^*_{z_1}, \ldots, u^*_{z_N}$ are simultaneously structured. In particular, for each subset $\mathcal{P}_j$, there exist piece functions $f_1, \ldots, f_N \in \mathcal{F}$ such that $u^*_{z_i}\left(u_{\boldsymbol{\rho}}\right) = f_i\left(u_{\boldsymbol{\rho}}\right)$ for all parameter settings $\boldsymbol{\rho} \in \mathcal{P}_j$ and $i \in [N]$.*

*Proof of Claim 7.2.5.* Let $u^*_{z_1}, \ldots, u^*_{z_N} \in \mathcal{U}^*$ be the dual functions corresponding to the problem instances $z_1, \ldots, z_N$. Since $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable, we know that for each function $u^*_{z_i}$, there are $k$ boundary functions $g_i^{(1)}, \ldots, g_i^{(k)} \in \mathcal{G} \subseteq \{0,1\}^{\mathcal{U}}$ that define its piecewise decomposition. Let $\hat{\mathcal{G}} = \bigcup_{i=1}^{N} \left\{ g_i^{(1)}, \ldots, g_i^{(k)} \right\}$ be the union of these boundary functions across all $i \in [N]$. For ease of notation, we relabel the functions in $\hat{\mathcal{G}}$, calling them $g_1, \ldots, g_{kN}$. Let $M$ be the total number of $kN$-dimensional vectors we can obtain by applying the functions in $\hat{\mathcal{G}} \subseteq \{0,1\}^{\mathcal{U}}$ to elements of $\mathcal{U}$:

$$M := \left| \left\{ \begin{pmatrix} g_1\left(u_{\boldsymbol{\rho}}\right) \\ \vdots \\ g_{kN}\left(u_{\boldsymbol{\rho}}\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathcal{P} \right\} \right|. \tag{7.4}$$

By Lemma 7.2.4, $M < (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}$. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ be the binary vectors in the set from Equation (7.4). For each $i \in [M]$, let $\mathcal{P}_j = \left\{ \boldsymbol{\rho} \mid \left(g_1\left(u_{\boldsymbol{\rho}}\right), \ldots, g_{kN}\left(u_{\boldsymbol{\rho}}\right)\right) = \boldsymbol{b}_j \right\}$. By construction, for each set $\mathcal{P}_j$, the values of all the boundary functions $g_1\left(u_{\boldsymbol{\rho}}\right), \ldots, g_{kN}\left(u_{\boldsymbol{\rho}}\right)$ are constant

as we vary $\boldsymbol{\rho} \in \mathcal{P}_j$. Therefore, there is a fixed set of piece functions $f_1, \ldots, f_N \in \mathcal{F}$ so that $u^*_{z_i}(u_{\boldsymbol{\rho}}) = f_i(u_{\boldsymbol{\rho}})$ for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}_j$ and indices $i \in [N]$. Therefore, the claim holds. □

Claim 7.2.5 and Equation (7.3) imply that for every subset $\mathcal{P}_j$ of the partition,

$$\left| \left\{ \begin{pmatrix} \operatorname{sign}(u_{\boldsymbol{\rho}}(z_1) - t_1) \\ \vdots \\ \operatorname{sign}(u_{\boldsymbol{\rho}}(z_N) - t_N) \end{pmatrix} \;\middle|\; \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right| = \left| \left\{ \begin{pmatrix} \operatorname{sign}(f_1(u_{\boldsymbol{\rho}}) - t_1) \\ \vdots \\ \operatorname{sign}(f_N(u_{\boldsymbol{\rho}}) - t_N) \end{pmatrix} \;\middle|\; \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right|. \tag{7.5}$$

Lemma 7.2.4 implies that Equation (7.5) is bounded by $(eN)^{\operatorname{Pdim}(\mathcal{F}^*)}$. In other words, for any region $\mathcal{P}_j$ of the partition, as we vary the parameter vector $\boldsymbol{\rho} \in \mathcal{P}_j$, $u_{\boldsymbol{\rho}}$ can label the problem instances $z_1, \ldots, z_N$ in at most $(eN)^{\operatorname{Pdim}(\mathcal{F}^*)}$ ways with respect to the target thresholds $t_1, \ldots, t_N$. Because there are $M < (ekN)^{\operatorname{VCdim}(\mathcal{G}^*)}$ regions $\mathcal{P}_j$ of the partition, we can conclude that $\mathcal{U}$ can label the problem instances $z_1, \ldots, z_N$ in at most $(ekN)^{\operatorname{VCdim}(\mathcal{G}^*)}(eN)^{\operatorname{Pdim}(\mathcal{F}^*)}$ distinct ways relative to the targets $t_1, \ldots, t_N$. In other words, Equation (7.3) is bounded by $(ekN)^{\operatorname{VCdim}(\mathcal{G}^*)}(eN)^{\operatorname{Pdim}(\mathcal{F}^*)}$. On the other hand, if $\mathcal{U}$ shatters the problem instances $z_1, \ldots, z_N$, then the number of distinct labelings must be $2^N$. Therefore, the pseduo-dimension of $\mathcal{U}$ is at most the largest value of $N$ such that $2^N \leq (ekN)^{\operatorname{VCdim}(\mathcal{G}^*)}(eN)^{\operatorname{Pdim}(\mathcal{F}^*)}$, which implies that

$$N = O\left( (\operatorname{Pdim}(\mathcal{F}^*) + \operatorname{VCdim}(\mathcal{G}^*)) \ln (\operatorname{Pdim}(\mathcal{F}^*) + \operatorname{VCdim}(\mathcal{G}^*)) + \operatorname{VCdim}(\mathcal{G}^*) \ln k \right),$$

as claimed. □

We prove several lower bounds which show that Theorem 7.2.3 is tight up to logarithmic factors.

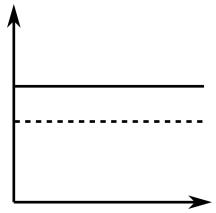
**Theorem 7.2.6.** *The following lower bounds hold:*

1. *There is a parameterized sequence alignment algorithm with* $\operatorname{Pdim}(\mathcal{U}) = \Omega(\log n)$ *for some* $n \geq 1$. *Its dual class* $\mathcal{U}^*$ *is* $(\mathcal{F}, \mathcal{G}, n)$*-piecewise decomposable for classes* $\mathcal{F}$ *and* $\mathcal{G}$ *with* $\operatorname{Pdim}(\mathcal{F}^*) = \operatorname{VCdim}(\mathcal{G}^*) = 1$.

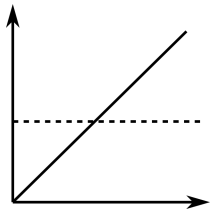
2. *There is a parameterized voting mechanism with* $\operatorname{Pdim}(\mathcal{U}) = \Omega(n)$ *for some* $n \geq 1$. *Its dual class* $\mathcal{U}^*$ *is* $(\mathcal{F}, \mathcal{G}, 2)$*-piecewise decomposable for classes* $\mathcal{F}$ *and* $\mathcal{G}$ *with* $\operatorname{Pdim}(\mathcal{F}^*) = 1$ *and* $\operatorname{VCdim}(\mathcal{G}^*) = n$.

*Proof.* In Theorem 6.1.6, we prove the result for sequence alignment, in which case $n$ is the maximum length of the sequences, $\mathcal{F}$ is the set of constant functions, and $\mathcal{G}$ is the set of threshold functions. In Theorem 5.2.3, we prove the result for voting mechanisms, in which case $n$ is the number of agents that participate in the mechanism, $\mathcal{F}$ is the set of constant functions, and $\mathcal{G}$ is the set of homogeneous linear separators in $\mathbb{R}^n$. □

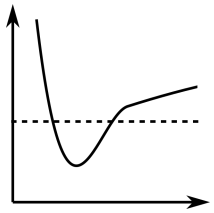
We now instantiate Theorem 7.2.3 in a general setting inspired by data-driven algorithm design. Let $\mathcal{U} = \{u_\rho \mid \rho \in \mathbb{R}\}$ be a set of utility functions defined over a single-dimensional parameter space. We often find that the dual functions are piecewise constant, linear, or polynomial. More generally, the dual functions are piecewise structured with piece functions that oscillate a fixed number of times. In other words, the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable where the boundary functions $\mathcal{G}$ are thresholds and the piece functions $\mathcal{F}$ oscillate a bounded number of times, as formalized below.



(a) Constant function (zero oscillations).



(b) Linear function (one oscillation).



(c) Inverse-quadratic function of the form $h(\rho) = \frac{a}{\rho^2} + b\rho + c$ (two oscillations).

Figure 7.3: Each solid line is a function with bounded oscillations and each dotted line is an arbitrary threshold. Many parameterized algorithms have piecewise-structured duals with piece functions from these families.

**Definition 7.2.7.** A function $h : \mathbb{R} \to \mathbb{R}$ has at most $B$ oscillations if for every $z \in \mathbb{R}$, the function $\rho \mapsto \mathbb{I}_{\{h(\rho) \geq z\}}$ is piecewise constant with at most $B$ discontinuities.

Figure 7.3 illustrates three common types of functions with bounded oscillations. In the following lemma, we prove that if $\mathcal{H}$ is a class of functions that map $\mathbb{R}$ to $\mathbb{R}$, each of which has at most $B$ oscillations, then $\text{Pdim}(\mathcal{H}^*) = O(\ln B)$.

**Lemma 7.2.8.** *Let $\mathcal{H}$ be a class of functions mapping $\mathbb{R}$ to $\mathbb{R}$, each of which has at most $B$ oscillations. Then* $\text{Pdim}(\mathcal{H}^*) = O(\ln B)$.

*Proof.* Suppose that $\text{Pdim}\,(\mathcal{H}^*) = N$. Then there exist functions $h_1, \ldots, h_N \in \mathcal{H}$ and witnesses $t_1, \ldots, t_N \in \mathbb{R}$ such that for every subset $T \subseteq [N]$, there exists a parameter setting $\rho \in \mathbb{R}$ such that $h_\rho^*(h_i) \geq t_i$ if and only if $i \in T$. We can simplify notation as follows: since $h(\rho) = h_\rho^*(h)$ for every function $h \in \mathcal{H}$, we have that for every subset $T \subseteq [N]$, there exists a parameter setting $\rho \in \mathbb{R}$ such that $h_i(\rho) \geq t_i$ if and only if $i \in T$. Let $\mathcal{P}^*$ be the set of $2^N$ parameter settings corresponding to each subset $T \subseteq [N]$. By definition, these parameter settings induce $2^N$ distinct binary vectors as follows:

$$\left| \left\{ \begin{pmatrix} \mathbb{I}_{\{h_1(\rho) \geq t_1\}} \\ \vdots \\ \mathbb{I}_{\{h_N(\rho) \geq t_N\}} \end{pmatrix} : \rho \in \mathcal{P}^* \right\} \right| = 2^N.$$

On the other hand, since each function $h_i$ has at most $B$ oscillations, we can partition $\mathbb{R}$ into $M \leq BN + 1$ intervals $I_1, \ldots, I_M$ such that for every interval $I_j$ and every $i \in [N]$, the function $\rho \mapsto \mathbb{I}_{\{h_i(\rho) \geq t_i\}}$ is constant across the interval $I_j$. Therefore, at most one parameter setting $\rho \in \mathcal{P}^*$ can fall within a single interval $I_j$. Otherwise, if $\rho, \rho' \in I_j \cap \mathcal{P}^*$, then

$$\begin{pmatrix} \mathbb{I}_{\{h_1(\rho) \geq t_1\}} \\ \vdots \\ \mathbb{I}_{\{h_N(\rho) \geq t_N\}} \end{pmatrix} = \begin{pmatrix} \mathbb{I}_{\{h_1(\rho') \geq t_1\}} \\ \vdots \\ \mathbb{I}_{\{h_N(\rho') \geq t_N\}} \end{pmatrix},$$

which is a contradiction. As a result, $2^N \leq BN + 1$. The lemma then follows from Lemma 2.1.4 in Section 2.1.1. □

Lemma 7.2.8 implies the following pseudo-dimension bound when the dual function class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable, where the boundary functions $\mathcal{G}$ are thresholds and the piece functions $\mathcal{F}$ oscillate a bounded number of times.

**Lemma 7.2.9.** *Let $\mathcal{U} = \{u_\rho \mid \rho \in \mathbb{R}\}$ be a set of utility functions and suppose the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-decomposable, where the boundary functions $\mathcal{G} = \{g_a \mid a \in \mathbb{R}\}$ are thresholds $g_a : u_\rho \mapsto \mathbb{I}_{\{a \leq \rho\}}$. Suppose for each $f \in \mathcal{F}$, the function $\rho \mapsto f(u_\rho)$ has at most $B$ oscillations. Then* $\text{Pdim}(\mathcal{U}) = O((\ln B)\ln(k \ln B))$.

*Proof.* First, we claim that $\text{VCdim}(\mathcal{G}^*) = 1$. For a contradiction, suppose $\mathcal{G}^*$ can shatter two functions $g_a, g_b \in \mathcal{G}^*$, where $a < b$. There must be a parameter setting $\rho \in \mathbb{R}$ where $g^*_{u_\rho}(g_a) = g_a(u_\rho) = \mathbb{I}_{\{a \leq \rho\}} = 0$ and $g^*_{u_\rho}(g_b) = g_b(u_\rho) = \mathbb{I}_{\{b \leq \rho\}} = 1$. Therefore, $b \leq \rho < a$, which is a contradiction, so $\text{VCdim}(\mathcal{G}^*) = 1$.

Next, we claim that $\text{Pdim}(\mathcal{F}^*) = O(\ln B)$. For each function $f \in \mathcal{F}$, let $h_f : \mathbb{R} \to \mathbb{R}$ be defined as $h_f(\rho) = f(u_\rho)$. By assumption, each function $h_f$ has at most $B$ oscillations. Let $\mathcal{H} = \{h_f \mid f \in \mathcal{F}\}$ and let $N = \text{Pdim}(\mathcal{H}^*)$. By Lemma 7.2.8, we know that $N = O(\ln B)$. We claim that $\text{Pdim}(\mathcal{H}^*) \geq \text{Pdim}(\mathcal{F}^*)$. For a contradiction, suppose the class $\mathcal{F}^*$ can shatter $N+1$ functions $f_1, \ldots, f_{N+1}$ using witnesses $t_1, \ldots, t_{N+1} \in \mathbb{R}$. By definition, this means that

$$\left| \left\{ \begin{pmatrix} \mathbb{I}_{\left\{f^*_{u_\rho}(f_1) \geq t_1\right\}} \\ \vdots \\ \mathbb{I}_{\left\{f^*_{u_\rho}(f_{N+1}) \geq t_{N+1}\right\}} \end{pmatrix} : \rho \in \mathcal{P} \right\} \right| = 2^{N+1}.$$

For any function $f \in \mathcal{F}$ and any parameter setting $\rho \in \mathbb{R}$, $f^*_{u_\rho}(f) = f(u_\rho) = h_f(\rho) = h^*_\rho(h_f)$. Therefore,

$$\left| \left\{ \begin{pmatrix} \mathbb{I}_{\left\{h^*_\rho(h_{f_1}) \geq t_1\right\}} \\ \vdots \\ \mathbb{I}_{\left\{h^*_\rho(h_{f_{N+1}}) \geq t_{N+1}\right\}} \end{pmatrix} : \rho \in \mathcal{P} \right\} \right| = \left| \left\{ \begin{pmatrix} \mathbb{I}_{\left\{f^*_{u_\rho}(f_1) \geq t_1\right\}} \\ \vdots \\ \mathbb{I}_{\left\{f^*_{u_\rho}(f_{N+1}) \geq t_{N+1}\right\}} \end{pmatrix} : \rho \in \mathcal{P} \right\} \right| = 2^{N+1},$$

which contradicts the fact that $\text{Pdim}(\mathcal{H}^*) = N$. Therefore, $\text{Pdim}(\mathcal{F}^*) \leq N = O(\ln B)$. The corollary then follows from Theorem 7.2.3. □

# 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), but 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 $L^\infty$-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. In Chapter 4, 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. In this chapter, we show that these dual functions can be approximated under the $L^\infty$-norm by simple functions (as in Figure 8.1), this chapter's 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 4. 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 [Balcan et al., 2020f].

## 8.1 Dual function approximability

Our goal is to provide generalization guarantees for the function class $\mathcal{U} = \{u_\rho \mid \rho \in \mathcal{P}\}$, where $\mathcal{P} \subseteq \mathbb{R}^d$ is a set of parameters and each function $u_\rho$ maps a set of problem instances $\mathcal{Z}$ to $[0, 1]$. As in Chapter 7, we use structure exhibited by 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 of Chapter 7 as follows: every function in the

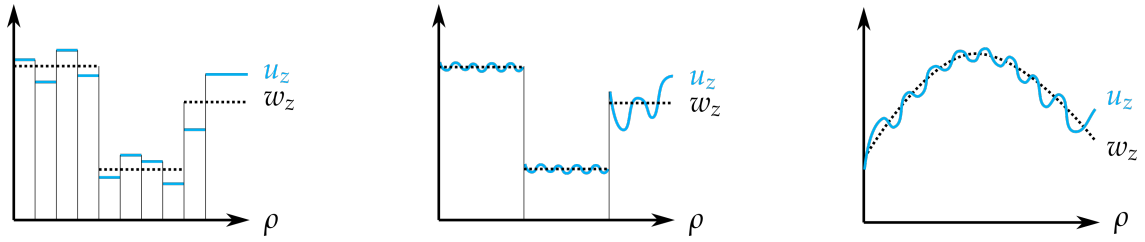



Figure 8.1: Examples of dual functions $u_z : \mathbb{R} \to \mathbb{R}$ (solid blue lines) which are approximated by simpler functions $w_z$ (dotted black lines).

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.

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** ($(\gamma, p)$-approximate)**.** Let $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P}\}$ and $\mathcal{W} = \{w_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P}\}$ 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 $\mathcal{W}^*$ $(\gamma, p)$*-approximates* the dual class $\mathcal{U}^*$ if for every element $z$, the distance between the functions $u_z$ and $w_z$ is at most $\gamma$ under the $L^p$-norm. For $p \in [1, \infty)$, this means that $\|u_z - w_z\|_p := \sqrt[p]{\int_{\mathcal{R}} |u_z(\boldsymbol{\rho}) - w_z(\boldsymbol{\rho})|^p \, d\boldsymbol{\rho}} \leq \gamma$ and when $p = \infty$, this means that $\|u_z - w_z\|_\infty := \sup_{\boldsymbol{\rho} \in \mathcal{P}} |u_z(\boldsymbol{\rho}) - w_z(\boldsymbol{\rho})| \leq \gamma$.

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

*Proof.* Let $\mathcal{S} = \{z_1, \ldots, z_N\}$ be an arbitrary subset of $\mathcal{Z}$. Fix an arbitrary vector $\boldsymbol{\rho} \in \mathcal{P}$ and index $i \in [N]$. Suppose that $\sigma[i] = 1$. Since $u_{z_i}(\boldsymbol{\rho}) \leq w_{z_i}(\boldsymbol{\rho}) + \|u_{z_i} - w_{z_i}\|_\infty$, we have that

$$\sigma[i] u_{z_i}(\boldsymbol{\rho}) \leq \sigma[i] w_{z_i}(\boldsymbol{\rho}) + \|u_{z_i} - w_{z_i}\|_\infty . \tag{8.1}$$

Meanwhile, suppose $\sigma[i] = -1$. Since $u_{z_i}(\boldsymbol{\rho}) \geq w_{z_i}(\boldsymbol{\rho}) - \|u_{z_i} - w_{z_i}\|_\infty$, we have that

$$\sigma[i] u_{z_i}(\boldsymbol{\rho}) = -u_{z_i}(\boldsymbol{\rho}) \leq -w_{z_i}(\boldsymbol{\rho}) + \|u_{z_i} - w_{z_i}\|_\infty = \sigma[i] w_{z_i}(\boldsymbol{\rho}) + \|u_{z_i} - w_{z_i}\|_\infty . \tag{8.2}$$

Combining Equations (8.1) and (8.2), we have that

$$\sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] w_{\boldsymbol{\rho}}(z_i) \geq \sum_{i=1}^{N} \sigma[i] w_{z_i}(\boldsymbol{\rho}) \geq \sum_{i=1}^{N} \sigma[i] u_{z_i}(\boldsymbol{\rho}) - \|u_{z_i} - w_{z_i}\|_\infty . \tag{8.3}$$

By definition of the supremum, Equation (8.3) implies that for every $\boldsymbol{\sigma} \in \{-1, 1\}^N$,

$$\sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] w_{\boldsymbol{\rho}}(z_i) \geq \sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] u_{\boldsymbol{\rho}}(z_i) - \sum_{i=1}^{N} \|u_{z_i} - w_{z_i}\|_\infty .$$

Therefore

$$\mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \{-1,1\}^N} \left[ \sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] w_{\boldsymbol{\rho}}(z_i) \right] \geq \mathop{\mathbb{E}}_{\boldsymbol{\sigma} \sim \{-1,1\}^N} \left[ \sup_{\boldsymbol{\rho} \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] u_{\boldsymbol{\rho}}(z_i) \right] - \sum_{i=1}^{N} \|u_{z_i} - w_{z_i}\|_\infty ,$$

so the theorem statement holds. □

If the class $\mathcal{W}^*$ $(\gamma, \infty)$-approximates the class $\mathcal{U}^*$, then $\frac{1}{|\mathcal{S}|} \sum_{z \in \mathcal{S}} \|u_z - w_z\|_\infty$ is at most $\gamma$. If this term is smaller than $\gamma$ 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 2.1.6 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 $\boldsymbol{\rho} \in \mathcal{P}$, the difference between the empirical average value of $u_{\boldsymbol{\rho}}$ over $\mathcal{S}$ and its expected value is at most $\tilde{O}\left( \frac{1}{N} \sum_{z \in \mathcal{S}} \|u_z - w_z\|_\infty + \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) + \sqrt{\frac{1}{N}} \right)$. 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 $\frac{1}{|\mathcal{S}|} \sum_{z \in \mathcal{S}} \|u_z - w_z\|_\infty$ 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 $j \geq 1$,*

$$\left| \frac{1}{N} \sum_{z \in \mathcal{S}} u_{\boldsymbol{\rho}}(z) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ u_{\boldsymbol{\rho}}(z) \right] \right| = \tilde{O}\left( \frac{1}{N} \sum_{z \in \mathcal{S}} \left\| u_z - w_{j,z} \right\|_\infty + \widehat{\mathscr{R}}_{\mathcal{S}}\left(\mathcal{W}_j\right) + \sqrt{\frac{1}{N}} \right). \tag{8.4}$$

*Proof.* We will prove that with probability at least $1-\delta$ over the draw of the training set $\mathcal{S} = \{z_1, \ldots, z_N\} \sim \mathcal{D}^N$, for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}$ and all $j \in \mathbb{N}$,

$$\left| \frac{1}{N} \sum_{z \in \mathcal{S}} u_{\boldsymbol{\rho}}(z) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ u_{\boldsymbol{\rho}}(z) \right] \right| \leq \frac{2}{N} \sum_{i=1}^{N} \left\| u_{z_i} - w_{j,z_i} \right\|_\infty + 2\widehat{\mathscr{R}}_{\mathcal{S}}\left(\mathcal{W}_j\right) + 3\sqrt{\frac{1}{2N} \ln \frac{(\pi j)^2}{3\delta}}.$$

For each integer $j \geq 1$, let $\delta_j = \frac{6\delta}{(\pi j)^2}$. From Theorems 2.1.6 and 8.2.1, we know that with probability at least $1-\delta_j$ over the draw of the training set $\mathcal{S} = \{z_1, \ldots, z_N\} \sim \mathcal{D}^N$, for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}$,

$$\begin{aligned} \left| \frac{1}{N} \sum_{z \in \mathcal{S}} u_{\boldsymbol{\rho}}(z) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ u_{\boldsymbol{\rho}}(z) \right] \right| &\leq \frac{2}{N} \sum_{i=1}^{N} \left\| u_{z_i} - w_{j,z_i} \right\|_\infty + 2\widehat{\mathscr{R}}_{\mathcal{S}}\left(\mathcal{W}_j\right) + 3\sqrt{\frac{1}{2N} \ln \frac{2}{\delta_j}} \\ &= \frac{2}{N} \sum_{i=1}^{N} \left\| u_{z_i} - w_{j,z_i} \right\|_\infty + 2\widehat{\mathscr{R}}_{\mathcal{S}}\left(\mathcal{W}_j\right) + 3\sqrt{\frac{1}{2N} \ln \frac{(\pi j)^2}{3\delta}}. \end{aligned}$$

Since $\sum_{i=1}^{\infty} \delta_j = \delta$, the corollary follows from a union bound over all $j \geq 1$. □

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 $\frac{1}{N} \sum_{z \in \mathcal{S}} \left\| u_z - w_{j,z} \right\|_\infty$ 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 (8.4).

*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 (8.4) 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. Our formal model is the same as that from Chapter 4, 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 define $u_{\boldsymbol{\rho}}(z)$ to be the size of the B&B tree CPLEX builds given the parameter setting $\boldsymbol{\rho}$ and input IP $z$, normalized to fall in $[0,1]$. We assume there is an upper bound $\kappa$ on the size of the largest tree we allow B&B to build before we terminate, as in prior research [Balcan et al., 2018a, Hutter et al., 2009, Kleinberg et al., 2017, 2019]. Later in this section, we describe our methodology for choosing $\kappa$.

We tune the parameter of B&B's variable selection policy (VSP), as described in Section 4.2 of Chapter 4. We study *score-based VSPs*, as defined in Section 4.2: Definition 4.2.3. We study how to learn a high-performing convex combination of any two scoring rules. We focus on four scoring rules—first defined in Chapter 4—in our experiments. To recall these scoring rules, we first revisit some notation from Chapter 4. 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 $\text{score}_L$, $\text{score}_S$, $\text{score}_A$, and $\text{score}_P$:

- $\text{score}_L(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, \check{c}_z - \check{c}_{z_i^-}\right\}$. Under $\text{score}_L$, 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 $\text{score}_S$, 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 [2009] recommended. It balances the optimistic approach to branching under $\text{score}_L$ with the pessimistic approach under $\text{score}_S$.
- $\text{score}_P(\mathcal{T}, z, i) = \max\left\{\check{c}_z - \check{c}_{z_i^+}, 10^{-6}\right\} \cdot \max\left\{\check{c}_z - \check{c}_{z_i^-}, 10^{-6}\right\}$. 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 $\text{score}_1$ and $\text{score}_2$. We define $u_\rho(z)$ to be the size of the tree CPLEX builds, capped at $\kappa$, divided by $\kappa$ (this way, $u_\rho(z) \in [0,1]$) 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.

Lemma 4.3.18 from Chapter 4 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 $\rho \in [0,1]$,

$$\left|\frac{1}{N}\sum_{z\in\mathcal{S}} u_\rho(z) - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_\rho(z)\right]\right| \leq 2\sqrt{\frac{2\ln(N\left(n^{2(\kappa+1)}-1\right)+1)}{N}} + 3\sqrt{\frac{1}{2N}\ln\frac{2}{\delta}}. \tag{8.5}$$

This worst-case bound can be large when $\kappa$ is large. We find that although the duals $u_z$ are piecewise-constant with many pieces, they can be approximated piecewise-constant functions

[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 $||\boldsymbol{c}||_1$.

with few pieces, as in Figure 8.1. As a result, we improve over Equation (8.5) 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 $(\gamma, \infty)$-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 $w_{j,z} \in \operatorname{argmin}_{h \in \mathcal{H}_j} \|u_z - h\|_\infty$, breaking ties in some fixed but arbitrary manner. The function $w_{j,z}$ can be found via dynamic programming, as we describe later in this section. We define the dual class $\mathcal{W}_j^* = \{w_{j,z} \mid z \in \mathcal{Z}\}$. 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, 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 $\mathcal{W}_j = \{w_{j,\rho} \mid \rho \in [0,1]\}$.

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 (8.5) and

$$2\gamma_j + 2\sqrt{\frac{2\ln(N(j-1)+1)}{N}} + \sqrt{\frac{2}{N}\ln\frac{2(\pi j)^2}{3\delta}}, \tag{8.6}$$

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.

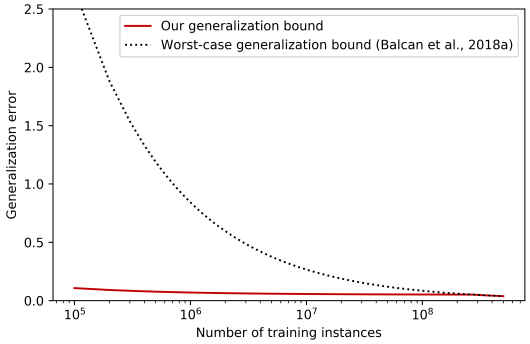
**Experiments.** As in Section 4.4 of Chapter 4, we analyze distributions over IPs formulating the combinatorial auction winner determination problem under the OR-bidding language [Sandholm, 2002], which we generate using the Combinatorial Auction Test Suite (CATS) [Leyton-Brown et al., 2000]. We use 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 4 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.

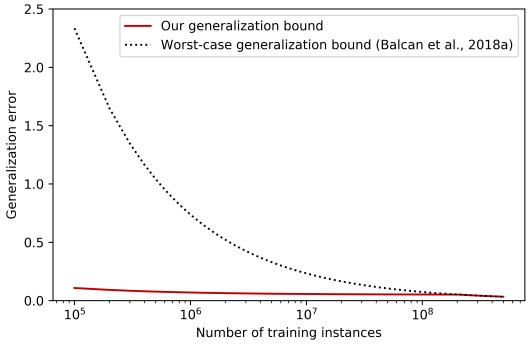
In Figures 8.2a-8.2c, we select $\text{score}_1, \text{score}_2 \in \{\text{score}_L, \text{score}_S, \text{score}_A, \text{score}_P\}$ and compare the worst-case and data-dependent bounds. First, we plot the worst-case bound from Equation (8.5), with $\delta = 0.01$, as a function of the number of training 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 (8.6), 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 $\mathcal{D}$.

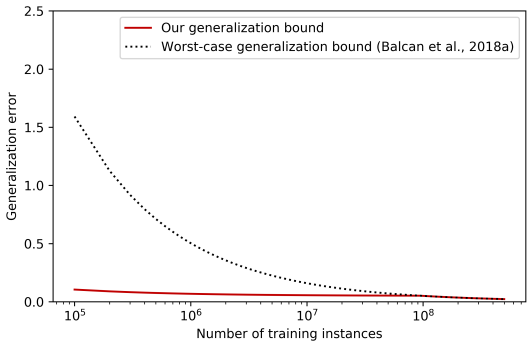
[2]We choose the range $j \in [1600]$ because under these distributions, the functions $u_z$ generally have at most 1600 pieces.




(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.

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 $j \in [1600]$,

$$\mathbb{E}\left[\left\|u_z - w_{j,z}\right\|_\infty\right] \leq \frac{1}{M}\sum_{i=1}^{M}\left\|u_{z_i} - w_{j,z_i}\right\|_\infty + \frac{1}{40}. \tag{8.7}$$

With thereby estimate our data-dependent bound Equation (8.6) using Equation (8.8), below:

$$\min_{j\in[1600]}\left\{2\left(\frac{1}{M}\sum_{i=1}^{M}\left\|u_{z_i} - w_{j,z_i}\right\|_\infty + \frac{1}{40}\right) + 2\sqrt{\frac{2\ln(N(j-1)+1)}{N}} + \sqrt{\frac{2}{N}\ln\frac{(20\pi j)^2}{3}}\right\}. \tag{8.8}$$

The only difference between these bounds is that Equation (8.6) relies on the left-hand-side of Equation (8.7) and Equation (8.8) relies on the right-hand-side of Equation (8.7) and sets $\delta = 0.005$.[3] In Figures 8.2a-8.2c, the red solid line equals the minimum of Equations (8.5) and (8.8) as a function of the number of training examples $N$.

[3]Like the worst-case bound, Equation (8.8) holds with probability 0.99, because with probability 0.995, Equation (8.7) holds, and with probability 0.995, the bound from Equation (8.6) holds.

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 (8.8)) 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.

**Selecting a tree size upper bound.** As we described in Section 8.2.2, we assume there is an upper bound $\kappa$ on the size of the largest tree we allow branch-and-bound to build before we terminate, as in prior research [Balcan et al., 2018a, Hutter et al., 2009, Kleinberg et al., 2017, 2019]. Given a parameter setting $\rho \in [0,1]$ and an integer program $z \in \mathcal{Z}$, we define $u_\rho(z)$ to be the size of the tree CPLEX builds, capped at $\kappa$, divided by $\kappa$ (this way, $u_\rho(z)$ is normalized, contained in the interval $[0,1]$).

We use a data-dependent approach to select $\kappa$. For any parameter $\rho \in [0,1]$ and integer program $z \in \mathcal{Z}$, let $f_\rho(z)$ be the size of the tree CPLEX builds (unnormalized). We draw $N = 6000$ integer programs $z_1, \ldots, z_N$ from the underlying distribution $\mathcal{D}$ and set $\kappa = \max_{\rho\in[0,1], i\in[N]} f_\rho(z_i)$. Classic results from learning theory guarantee that with high probability, for at most 8% of the integer programs sampled from $\mathcal{D}$, CPLEX will build a tree of size larger than $\kappa$ when parameterized by some $\rho \in [0,1]$. Specifically, since the VC dimension of threshold functions is 1, we have that with probability at least 0.99 over the draw of the $N$ samples, $\Pr_{z\sim\mathcal{D}}\left[\max_{\rho\in[0,1]} u_\rho(z) > \kappa\right] < 0.08$.

For the "arbitrary" distribution, when $\text{score}_1 = \text{score}_L$ and $\text{score}_2 = \text{score}_S$, $\kappa = 6341$, and when $\text{score}_1 = \text{score}_P$ and $\text{score}_2 = \text{score}_A$, $\kappa = 2931$. For the "regions" distribution, when $\text{score}_1 = \text{score}_L$ and $\text{score}_2 = \text{score}_S$, $\kappa = 7314$.

**Dynamic programming.** For any $k \in \mathbb{N}$, let $\mathcal{W}_k$ be the set of piecewise-constant functions with $k$ pieces mapping an interval $\mathcal{P} \subseteq \mathbb{R}$ to $\mathbb{R}$. In this section, we provide a dynamic programming algorithm which takes as input a piecewise-constant dual function $u_z : \mathcal{P} \to \mathbb{R}$ and a value $k \in \mathbb{N}$ and returns the value $\min_{w\in\mathcal{W}_k} \|u_z - w\|_\infty$. Since $u_z$ is piecewise-constant, the domain $\mathcal{P}$ can be partitioned into intervals $[a_1, a_2), [a_2, a_3) \ldots, [a_t, a_{t+1})$ such that for any interval $[a_i, a_{i+1})$, there exists a value $c_i \in \mathbb{R}$ such that $u_z(\rho) = c_i$ for all $\rho \in [a_i, a_{i+1})$.

We now provide an overview of the algorithm. See Algorithm 6 for the pseudo-code. The algorithm takes as input the partition $[a_1, a_2), \ldots, [a_t, a_{t+1})$ of the parameter space $\mathcal{P}$ and values $c_1, \ldots, c_t$ such that for any interval $[a_i, a_{i+1})$, $u_z(\rho) = c_i$ for all $\rho \in [a_i, a_{i+1})$. The algorithm begins by calculating upper and lower bounds on the value of the function $u_z$ across various subsets of its domain. In particular, for each $i, i' \in [t]$ such that $i \le i'$, the algorithm calculates the lower bound $\ell_{i,i'} = \min\{c_i, c_{i+1}, \ldots, c_{i'}\}$ and the upper bound $h_{i,i'} = \max\{c_i, c_{i+1}, \ldots, c_{i'}\}$. Algorithm 6 performs these calculations in $O(t^2)$ time.

Next, for each $i \in [t]$ and $j \in [k]$, the algorithm calculates a value $C(i,j)$ which equals the smallest $\ell_\infty$ norm between any piecewise constant function with $j$ pieces and the function $u_z$ when restricted to the interval $[a_1, a_{i+1})$. Since $\mathcal{P} = [a_1, a_{t+1})$, we have that $C(t,k)$—the value

**Algorithm 6** Piecewise-constant function fitting via dynamic programming

**Input:** Partition $[a_1, a_2), \ldots, [a_t, a_{t+1})$ of $\mathcal{P}$, values $c_1, \ldots, c_t$, and desired number of pieces $k \in \mathbb{N}$.
**for** $i \in [t]$ **do**
    Set $h_{i,i} = c_i$ and $\ell_{i,i} = c_i$.
    **for** $i' \in \{i+1, \ldots, t\}$ **do**
        **if** $c_{i'} < \ell_{i,i'-1}$ **then**
            Set $\ell_{i,i'} = c_{i'}$ and $h_{i,i'} = h_{i,i'-1}$.
        **else if** $c_{i'} > h_{i,i'-1}$ **then**
            Set $\ell_{i,i'} = \ell_{i,i'-1}$ and $h_{i,i'} = c_{i'}$.
        **else**
            Set $\ell_{i,i'} = \ell_{i,i'-1}$ and $h_{i,i'} = h_{i,i'-1}$.
**for** $i \in [t]$ **do**
    Set $C(i,1) = \frac{h_{1,i} - \ell_{1,i}}{2}$
**for** $j \in \{2, \ldots, k\}$ **do**
    **for** $i \in [t]$ **do**
        Set $C(i,j) = \min\left\{C(i,1), \min_{i' \in [i-1]}\left\{C(i', j-1) + \frac{h_{i'+1,i} - \ell_{i'+1,i}}{2}\right\}\right\}$
**Output:** $C(t,k)$.

our algorithm returns—equals $\min_{w \in \mathcal{W}_k} \|u_z - w\|_\infty$, as claimed. For all $i \in [t]$, $C(i,1) = \frac{h_{1,i} - \ell_{1,i}}{2}$ and for all $j \geq 2$,

$$C(i,j) = \min\left\{C(i,1), \min_{i' \in [i-1]}\left\{C(i', j-1) + \frac{h_{i'+1,i} - \ell_{i'+1,i}}{2}\right\}\right\}.$$

Algorithm 6 performs these calculations in $O(kt^2)$ time.

### 8.2.3 Rademacher complexity lower bound

In this section, we show that $(\gamma, p)$-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 $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) = 0$, yet for the primal $\mathcal{U}$ to have high Rademacher complexity.

Figures 8.3 and 8.4 help explain why there is this sharp constrast between the $L^\infty$- 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 $\mathcal{W} = \{w_\rho : \rho \in \mathbb{R}\}$ where $w_\rho(z) = \frac{1}{2}$ for every $z \in \mathcal{Z}$. It is easy to see that $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) = 0$ 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 $\int_{\mathcal{P}} \left|u_{z_i}(\rho) - \frac{1}{2}\right| d\rho$ are small. However, the approximation is not strong under the $L^\infty$-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 $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) = 0$, we have that $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$ when $\mathcal{S} = \{z_1, z_2\}$, which makes Theorem 2.1.6 meaningless. At a high level, this is because when $\sigma[1] = 1$, we

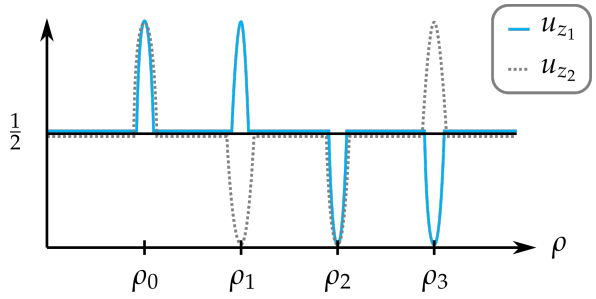



Figure 8.3: The dual functions $u_{z_1}$ and $u_{z_2}$ are well-approximated by the constant function $\rho \mapsto \frac{1}{2}$ under, for example, the $L^1$-norm because the integrals $\int_{\mathcal{P}} \left| u_{z_i}(\rho) - \frac{1}{2} \right| d\rho$ are small; for most $\rho$, $u_{z_i}(\rho) = \frac{1}{2}$. The approximation is not strong under the $L^\infty$-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.

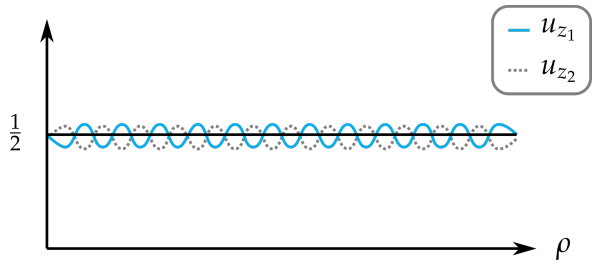



Figure 8.4: The dual functions $u_{z_1}$ and $u_{z_2}$ are well-approximated by the constant function $\rho \mapsto \frac{1}{2}$ under the $L^\infty$-norm since $\max_{\rho \in \mathcal{P}} \left| u_{z_i}(\rho) - \frac{1}{2} \right|$ is small. The function class $\mathcal{U}$ corresponding to these duals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

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 $\sigma[2]$. In summary, $(\gamma, p)$-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 $L^\infty$-norm for every parameter $\rho$. As a result, for any noise vector $\boldsymbol{\sigma} \in \{-1, 1\}^2$, $\sup_{\rho \in \mathbb{R}} \{\sigma[1] u_{z_1}(\rho) + \sigma[2] u_{z_2}(\rho)\}$ is close to $\sup_{\rho \in \mathbb{R}} \{\sigma[1] w_{z_1}(\rho) + \sigma[2] w_{z_2}(\rho)\}$. This implies that the Rademacher complexities $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W})$ and $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U})$ are close. This illustration exemplifies Theorem 8.2.1: $(\gamma, \infty)$-approximability implies strong Rademacher bounds.

We now prove that $(\gamma, p)$-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 $\mathcal{U}, \mathcal{W} \subset [0,1]^{\mathcal{Z}}$ such that the dual class $\mathcal{W}^*$ $(\gamma, p)$-approximates $\mathcal{U}^*$ and for any $N \geq 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) = 0$ and $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$.*

*Proof.* We begin by defining the classes $\mathcal{U}$ and $\mathcal{W}$. Let $\mathcal{P} = (0, \gamma^p]$, and $\mathcal{Z} = [\gamma^{-p}/2, \infty)$. For any $\rho \in \mathcal{P}$ and $z \in \mathcal{Z}$, let $u_\rho(z) = \frac{1}{2}(1 + \cos(\rho z))$ and $\mathcal{U} = \{u_\rho \mid \rho \in \mathcal{P}\}$. These sinusoidal functions are based on the intuition from Figure 8.3. As in Figure 8.3, for any $\rho$ and $z$, let $w_\rho(z) = \frac{1}{2}$ and $\mathcal{W} = \{w_\rho \mid \rho \in \mathcal{P}\}$. Since $\mathcal{W}$ consists of identical copies of a single function, $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{W}) = 0$ for any set $\mathcal{S} \subseteq \mathcal{Z}$. Meanwhile, in Lemma 8.2.9, we prove that for any $N \geq 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$.

In Lemma 8.2.8, we prove that the dual class $\mathcal{W}^*$ $(\gamma, p)$-approximates $\mathcal{U}^*$. To prove this, we first show that $\|u_z - w_z\|_2 \leq \frac{1}{4}\sqrt{2\gamma^p + \frac{1}{z}}$. When $p = 2$, we know $\frac{1}{z} \leq 2\gamma^2$, so $\|u_z - w_z\|_2 < \gamma$. Otherwise, we use our bound on $\|u_z - w_z\|_2$, Hölder's inequality, and the log-convexity of the $L^p$-norm to prove that $\|u_z - w_z\|_p \leq \gamma$. □

To prove Theorem 8.2.5, we rely on the following two well-known results.

**Theorem 8.2.6** (Hölder's inequality)**.** *Let $p_0$ and $p_1$ be two values in $[1, \infty]$ such that $\frac{1}{p_0} + \frac{1}{p_1} = 1$. Then for all functions $u$ and $w$, $\|uw\|_1 \leq \|u\|_{p_0} \|w\|_{p_1}$.*

**Theorem 8.2.7** (Interpolation)**.** *Let $p$ and $q$ be two values in $(0, \infty]$ and let $\theta$ be a value in $(0, 1)$. Let $p_\theta$ be defined such that $\frac{1}{p_\theta} = \frac{\theta}{p_1} + \frac{1-\theta}{p_0}$. Then for all functions $u$, $\|u\|_{p_\theta} \leq \|u\|_{p_1}^{\theta} \|u\|_{p_0}^{1-\theta}$.*

We now prove Lemma 8.2.8, which guarantees that the dual class $\mathcal{W}^*$ $(\gamma, p)$-approximates $\mathcal{U}^*$.

**Lemma 8.2.8.** *For any $\gamma \in \left(0, \frac{1}{4}\right)$ and $p \in [1, \infty)$, let $\mathcal{U}$ and $\mathcal{W}$ be the function classes defined in Theorem 8.2.5. The dual class $\mathcal{W}^*$ $(\gamma, p)$-approximates the dual class $\mathcal{U}^*$.*

*Proof.* For ease of notation, let $t = \gamma^p$, $a = \frac{1}{2\gamma^p}$, $\mathcal{P} = (0, t]$, and $\mathcal{X} = \left[\frac{1}{2\gamma^p}, \infty\right)$. Throughout this proof, we will use the following inequality:

$$\begin{aligned}\|u_z - w_z\|_2 &= \sqrt{\int_0^t \left(u_z(\rho) - w_z(\rho)\right)^2 d\rho} = \sqrt{\int_0^t \left(\frac{1}{2}\cos(\rho z)\right)^2 d\rho} = \frac{1}{4}\sqrt{2t + \frac{\sin(2tz)}{z}} \\ &\leq \frac{1}{4}\sqrt{2t + \frac{1}{z}}. \end{aligned} \tag{8.9}$$

First, suppose $p = 2$. Since $t = \gamma^2$ and $\frac{1}{z} \leq 2\gamma^2$, Equation (8.9) implies that $\|u_z - w_z\|_2 \leq \frac{1}{4}\sqrt{4\gamma^2} < \gamma$.

Next, suppose $p < 2$. We know that

$$\left\|\left(u_z - w_z\right)^p\right\|_1 = \int_0^t \left|\left(u_z(\rho) - w_z(\rho)\right)^p\right| d\rho = \int_0^t \left|u_z(\rho) - w_z(\rho)\right|^p d\rho = \|u_z - w_z\|_p^p. \tag{8.10}$$

From Equation (8.10) and Hölder's inequality (Theorem 8.2.6) with $u = \left(u_z - w_z\right)^p$, $w$ the constant function $w : \rho \mapsto 1$, $p_0 = \frac{2}{p}$, and $p_1 = \frac{2}{2-p}$, we have that

$$\begin{aligned}\|u_z - w_z\|_p^p &= \left\|\left(u_z - w_z\right)^p\right\|_1 \\ &\leq \|w\|_{\frac{2}{2-p}} \left\|\left(u_z - w_z\right)^p\right\|_{\frac{2}{p}} \\ &= \left(\int_0^t d\rho\right)^{\frac{2-p}{2}} \left\|\left(u_z - w_z\right)^p\right\|_{\frac{2}{p}} \\ &= t^{\frac{2-p}{2}} \left\|\left(u_z - w_z\right)^p\right\|_{\frac{2}{p}} \\ &= t^{\frac{2-p}{2}} \left(\int_0^t \left(u_z(\rho) - w_z(\rho)\right)^2 d\rho\right)^{\frac{p}{2}} \\ &= t^{\frac{2-p}{2}} \|u_z - w_z\|_2^p. \end{aligned}$$

Therefore,

$$\begin{aligned}
\|u_z - w_z\|_p &\le t^{\frac{1}{p}-\frac{1}{2}} \|u_z - w_z\|_2 \\
&\le \frac{t^{\frac{1}{p}-\frac{1}{2}}}{4}\sqrt{2t+\frac{1}{z}} && \text{(Equation (8.9))} \\
&= \frac{t^{\frac{1}{p}}}{4}\sqrt{2+\frac{1}{zt}} \\
&= \frac{\gamma}{4}\sqrt{2+\frac{1}{z\gamma^p}} && (t=\gamma^p) \\
&< \gamma, && \left(z \ge \frac{1}{2\gamma^p}\right)
\end{aligned}$$

Finally, suppose $p > 2$. Let $\theta = 1 - \frac{2}{p}$, $p_0 = 2$, and $p_1 = \infty$. By Theorem 8.2.7,

$$\begin{aligned}
\|u_z - w_z\|_p &\le \|u_z - w_z\|_2^{1-\theta} \\
&= \|u_z - w_z\|_2^{\frac{2}{p}} \\
&\le \sqrt[p]{\frac{t}{8}+\frac{1}{16z}} && \text{(Equation (8.9))} \\
&= \sqrt[p]{\frac{\gamma^p}{8}+\frac{1}{16z}} && (t=\gamma^p) \\
&\le \sqrt[p]{\frac{\gamma^p}{4}} && \left(z \ge \frac{1}{2\gamma^p}\right) \\
&< \gamma.
\end{aligned}$$

Therefore, for all $p \in [1,\infty)$ and all $z \in \mathcal{X}$, $\|u_z - w_z\|_p \le \gamma$, so the dual class $\mathcal{W}^*$ $(\gamma, p)$-approximates the dual class $\mathcal{U}^*$. □

Finally, we prove Lemma 8.2.9, which guarantees that for any $N \ge 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$.

**Lemma 8.2.9.** *For any $\gamma \in \left(0, \frac{1}{4}\right)$ and $p \in [1,\infty)$, let $\mathcal{U} = \left\{u_\rho \mid \rho \in (0,\gamma^p]\right\}$ be a class of functions with domain $\left[\frac{1}{2\gamma^p}, \infty\right)$ such that for all $\rho \in (0,\gamma^p]$ and $z \in \left[\frac{1}{2\gamma^p}, \infty\right)$, $u_\rho(z) = \frac{1}{2}(1+\cos(\rho z))$. For every $N \ge 1$, $\sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{2}$.*

*Proof.* This proof is similar to the proof that the VC-dimension of the function class $\{z \mapsto \text{sign}(\sin(\rho z)) \mid \rho \in \mathbb{R}\} \subseteq \{-1,1\}^{\mathbb{R}}$ is infinite (see, for example, Lemma 7.2 in the textbook by Anthony and Bartlett [2009]). To prove this lemma, we will show that for every $c \in (0,1/2)$, $\overline{\mathscr{R}}_N(\mathcal{U}) := \sup_{\mathcal{S}:|\mathcal{S}|=N} \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) \ge c$ (Claim 8.2.10). We also show that $\overline{\mathscr{R}}_N(\mathcal{U}) \le \frac{1}{2}$ (Claim 8.2.11). Therefore, the lemma statement follows.

**Claim 8.2.10.** *For every $c \in (0,1/2)$, $\overline{\mathscr{R}}_N(\mathcal{U}) \ge c$.*

*Proof of Claim 8.2.10.* Let $N$ be an arbitrary positive integer. We begin by defining several variables that we will use throughout this proof. Let $\mathcal{P} = (0,\gamma^p]$ and let $\alpha$ be any positive power of $\frac{1}{2}$ smaller than $\min\left\{\frac{1}{2\pi+1}, \frac{\arccos(2c)}{\pi+\arccos(2c)}\right\}$. Since $2c \in (0,1)$, $\frac{\arccos(2c)}{\pi+\arccos(2c)}$ is well-defined. Also,

since $\alpha \leq \frac{\arccos(2c)}{\pi+\arccos(2c)}$, we have that $\frac{\pi\alpha}{1-\alpha} \leq \arccos(2c) < \frac{\pi}{2}$. Finally, since the function cos is decreasing on the interval $[0, \pi/2]$, we have that $\frac{1}{2}\cos\frac{\pi\alpha}{1-\alpha} \geq c$. Let $z_i = \frac{\alpha^{-i}}{2\gamma^p}$ for $i \in [N]$. Since $\alpha < 1$, we have that $z_i \geq \frac{1}{2\gamma^p}$, so each $z_i$ is an element of the domain $\left[\frac{1}{2\gamma^p}, \infty\right)$ of the functions in $\mathcal{U}$.

We will show that for every assignment of the variables $\sigma[1], \ldots, \sigma[N] \in \{-1, 1\}$, there exists a parameter $\rho_0 \in (0, \gamma^p]$ such that

$$\frac{1}{N} \sup_{\rho \in (0,\gamma^p]} \sum_{i=1}^{N} \sigma[i] u_\rho(z_i) \geq \frac{1}{N} \sum_{i=1}^{N} \sigma[i] u_{\rho_0}(z_i) = \frac{1}{2N} \sum_{i=1}^{N} \sigma[i] \left(1 + \cos\left(\rho_0 z_i\right)\right) \geq c + \frac{1}{2} \sum_{i=1}^{N} \sigma[i].$$

This means that when $\mathcal{S} = \{z_1, \ldots, z_N\}$,

$$\overline{\mathscr{R}}_N(\mathcal{U}) \geq \widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{U}) = \frac{1}{N} \mathop{\mathbb{E}}_{\sigma} \left[ \sup_{\rho \in \mathcal{P}} \sum_{i=1}^{N} \sigma[i] u_\rho(z_i) \right] \geq c + \frac{1}{2} \mathop{\mathbb{E}}_{\sigma} \left[ \sum_{i=1}^{N} \sigma[i] \right] = c.$$

To this end, given an assignment of the variables $\sigma[1], \ldots, \sigma[N] \in \{-1, 1\}$, let $(b_1, \ldots, b_N) \in \{0, 1\}^N$ be defined such that

$$b_i = \begin{cases} 0 & \text{if } \sigma[i] = 1 \\ 1 & \text{otherwise} \end{cases}$$

and let

$$\rho_0 = 2\pi\gamma^p \left( \sum_{j=1}^{N} \alpha^j b_j + \alpha^{N+1} \right).$$

Since $0 < \rho_0 < 2\pi\gamma^p \sum_{j=1}^{\infty} \alpha^j = \frac{2\pi\gamma^p\alpha}{1-\alpha} \leq \gamma^p$, $\rho_0$ is an element of the parameter space $(0, \gamma^p]$. The inequality $\frac{2\pi\gamma^p\alpha}{1-\alpha} \leq \gamma^p$ holds because $\alpha \leq \frac{1}{2\pi+1}$, so $\frac{2\pi\alpha}{1-\alpha} \leq 1$.

Next, we evaluate $u_{\rho_0}(z_i) = \frac{1}{2}\left(1 + \cos(\rho_0 z_i)\right)$:

$$\begin{aligned}
\frac{1}{2}\left(1 + \cos(\rho_0 z_i)\right) &= \frac{1}{2} + \frac{1}{2}\cos\left(2\pi\gamma^p \left( \sum_{j=1}^{N} \alpha^j b_j + \alpha^{N+1} \right) \frac{\alpha^{-i}}{2\gamma^p}\right) \\
&= \frac{1}{2} + \frac{1}{2}\cos\left(\pi \left( \sum_{j=1}^{N} \alpha^j b_j + \alpha^{N+1} \right) \alpha^{-i}\right) \\
&= \frac{1}{2} + \frac{1}{2}\cos\left( \sum_{j=1}^{i-1} \alpha^{j-i}\pi b_j + \pi b_i + \sum_{j=i+1}^{N} \alpha^{j-i}\pi b_j + \alpha^{N+1-i}\pi \right) \\
&= \frac{1}{2} + \frac{1}{2}\cos\left(\pi \left( b_i + \sum_{j=1}^{N-i} \alpha^j b_{i+j} + \alpha^{N+1-i} \right)\right). \qquad (8.11)
\end{aligned}$$

The final equality holds because for every $j < i$, $\alpha^{j-i}$ is a positive power of 2, so $\alpha^{j-i}\pi b_j$ is a multiple of $2\pi$. We will use the following fact: since

$$0 < \sum_{j=1}^{N-i} \alpha^j b_{i+j} + \alpha^{N+1-i} \leq \sum_{j=1}^{N-i+1} \alpha^j < \sum_{j=1}^{\infty} \alpha^j = \frac{\alpha}{1-\alpha},$$

the argument of $\cos(\cdot)$ in Equation (8.11) lies strictly between $\pi b_i$ and $\pi b_i + \frac{\pi\alpha}{1-\alpha}$.

Suppose $b_i = 0$. Since $\alpha \leq \frac{1}{2}$, we know that $\frac{\pi\alpha}{1-\alpha} \leq \pi$. Therefore, $\cos(\cdot)$ is monotone decreasing on the interval $\left[0, \frac{\pi\alpha}{1-\alpha}\right]$. Moreover, we know that $\frac{1}{2}\cos\frac{\pi\alpha}{1-\alpha} \geq c$. Therefore, $u_{\rho_0}(z_i) = \frac{1}{2}(1+\cos(\rho_0 z_i)) \geq \frac{1}{2} + c$. Since $b_i = 0$, it must be that $\sigma[i] = 1$, so $\sigma[i]u_{\rho_0}(z_i) \geq c + \frac{1}{2} = c + \frac{\sigma[i]}{2}$. Meanwhile, suppose $b_i = 1$. The function $\cos(\cdot)$ is monotone increasing on the interval $\left[\pi, \pi + \frac{\pi\alpha}{1-\alpha}\right]$. Moreover, $\frac{1}{2}\cos\left(\pi + \frac{\pi\alpha}{1-\alpha}\right) = -\frac{1}{2}\cos\frac{\pi\alpha}{1-\alpha} \leq -c$. Therefore,

$$u_{\rho_0}(z_i) = \frac{1}{2}(1+\cos(\rho_0 z_i)) \leq \frac{1}{2} - c.$$

Since $b_i = 1$, it must be that $\sigma[i] = -1$, so $\sigma[i]u_{\rho_0}(z_i) \geq c - \frac{1}{2} = c + \frac{\sigma[i]}{2}$. Since this is true for any $i \in [N]$, we have that

$$\frac{1}{2N}\sum_{i=1}^{N}\sigma[i]\left(1+\cos\left(\rho_0 z_i\right)\right) \geq c + \frac{1}{2}\sum_{i=1}^{N}\sigma[i],$$

as claimed. □

We conclude this proof by showing that $\overline{\mathscr{R}}_N(\mathcal{U}) \leq \frac{1}{2}$.

**Claim 8.2.11.** *For any $N \geq 1$, $\overline{\mathscr{R}}_N(\mathcal{U}) \leq \frac{1}{2}$.*

*Proof of Claim 8.2.11.* Let $\mathcal{S} = \{z_1, \ldots, z_N\} \subset \left[\frac{1}{2\gamma^p}, \infty\right)$ be an arbitrary set of points. For any assignment of the variables $\sigma[1], \ldots, \sigma[N] \in \{-1, 1\}$, since $u_\rho(z_i) \in [0,1]$,

$$\sup_{\rho\in(0,\gamma^p]}\sum_{i=1}^{N}\sigma[i]u_\rho(z_i) \leq \sum_{i=1}^{N}\mathbf{1}_{\{\sigma[i]=1\}}.$$

Therefore,

$$\overline{\mathscr{R}}_N(\mathcal{U}) = \sup_{z_1,\ldots,z_N}\frac{1}{N}\mathop{\mathbb{E}}_{\sigma}\left[\sup_{\rho\in(0,\gamma^p]}\sum_{i=1}^{N}\sigma[i]u_\rho(z_i)\right] \leq \frac{1}{N}\mathop{\mathbb{E}}_{\sigma}\left[\sum_{i=1}^{N}\mathbf{1}_{\{\sigma[i]=1\}}\right] = \frac{1}{2},$$

as claimed. □

Together, Claims 8.2.10 and 8.2.11 imply that for every $N \geq 1$, $\overline{\mathscr{R}}_N(\mathcal{U}) = \frac{1}{2}$. □

*Remark* 8.2.12. 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

# Portfolio-based algorithm selection

In practice, the best parameter setting for one problem is rarely optimal for another. *Algorithm portfolios*—which are finite sets of parameter settings—are used in practice to deal with this variability. A portfolio is often used in conjunction with an *algorithm selector*, which is a function that determines which parameter setting in the portfolio to employ on any input problem instance. Portfolio-based algorithm selection has seen tremendous empirical success, fueling breakthroughs in combinatorial auction winner determination [Leyton-Brown, 2003, Sandholm, 2013], SAT [Xu et al., 2008], integer programming [Kadioglu et al., 2010, Xu et al., 2010], planning [Cenamor et al., 2016, Núñez et al., 2015], and many other domains.

Both the portfolio and the algorithm selector are often chosen using a training set of problem instances from the application domain at hand. The portfolio and algorithm selector are chosen to have strong average performance over the training set. In this chapter, we investigate the learned algorithm selector's generalization error, which is the difference between the performance of the configurations it selects on average over the training set and the expected performance of the configuration selects on a freshly-drawn instance.

There are multiple reasons the generalization error might be large in this setting: 1) the learning-theoretic complexity of the algorithm selector, 2) the size of the portfolio, and 3) the learning-theoretic complexity of the algorithm's performance as a function of its parameters. We provide end-to-end bounds on generalization error in terms of all three elements simultaneously. The variety of factors impacting generalization error differentiates this chapter from prior results in this thesis on generalization guarantees in algorithm configuration. That research focuses on bounding the generalization error of learning a *single* good parameter setting for the entire problem instance distribution, rather than a portfolio together with an algorithm selector that selects an algorithm (e.g., its parameter values) from the portfolio *for the specific instance at hand*. In the former case, generalization error only grows with (3)—just one of the sources of error we must contend with.

Our bounds apply to the setting where on any fixed input, the dual function measuring the algorithm's performance as a function of its parameters is piecewise constant with at most $t$ pieces, for some $t \in \mathbb{Z}$. We observed this structure in the earlier chapters of this thesis in the contexts of integer programming and computational biology. Given a training set of size $N$, we prove that the generalization error is bounded[1] by $\tilde{O}\left(\sqrt{\left(\bar{d} + \kappa \log t\right) / N}\right)$, where $\kappa$ is the size of the portfolio and $\bar{d}$ measures the *intrinsic complexity* of the algorithm selector, as we define in

[1]Here we assume that algorithmic performance is a quantity in $[0, 1]$, an assumption we relax in Section 9.1.

Section 9.2. We also prove that this bound is tight up to logarithmic factors: the generalization error can be as large as $\tilde{\Omega}\left(\sqrt{\left(\bar{d}+\kappa\right)/N}\right)$. This implies that even if the algorithm selector is extremely simple ($\bar{d}$ is small), overfitting cannot be avoided in the worst case when the portfolio size $\kappa$ is large. Moreover, we instantiate our guarantees for several commonly-used families of algorithm selectors [Hutter et al., 2014, Kadioglu et al., 2010, Xu et al., 2008].

Finally, via experiments in the context of integer programming configuration, we illustrate the inherent tradeoff our theory exposes: as we increase the portfolio size, we can hope to include a high-performing parameter setting for any given instance, but it become increasingly difficult to avoid overfitting. We incrementally increase the size of the portfolio and with each addition we train an algorithm selector using regression forest performance models. As the portfolio size increases, the algorithm selector's training performance continues to improve, but there comes a point where the test performance begins to worsen, meaning that the algorithm selector is overfitting to the training set.

The results in this section are joint work with Nina Balcan and Tuomas Sandholm and appeared in AAAI 2021 [Balcan et al., 2021c].

**Additional related research.** Gupta and Roughgarden [2017] also provide generalization guarantees for algorithm portfolios, though that paper focuses on the problem of learning a single parameter setting with high expected performance on the underlying distribution. They do provide guarantees for the more general problem of learning a mapping from instances to parameter settings in a few special cases, but do not study the problem of learning a portfolio in conjunction with learning a selector, which we do. They study settings where for each problem instance, a domain expert has defined a number of relevant features, as do we in Section 9.3. Their first result applies to learning an algorithm selector when the set of features is finite. In contrast, our results apply to infinite feature spaces. Their second set of results is tailored to the problem of learning empirical performance models and applies when the feature space is infinite. An empirical performance model is meant to predict how long a particular algorithm will take to run on a given input. An algorithm selector can use an empirical performance model by selecting the parameter setting with best predicted performance. Gupta and Roughgarden [2017] provide guarantees that bound the difference between the empirical performance model's expected error and average error over the training set. Their guarantees can be applied once the portfolio is already chosen. They do not study the problem of learning the portfolio itself, whereas we study the composite problem of learning the portfolio and the algorithm selector.

## 9.1 Problem formulation and road map

**Notation.** Our theoretical guarantees apply to algorithms parameterized by a real value $\rho \in \mathbb{R}$. We use the notation $\mathcal{Z}$ to denote the set of problem instances the algorithm may take as input. For example, $\mathcal{Z}$ might consist of integer programs (IPs) if we are configuring an IP solver. There is an unknown distribution $\mathcal{D}$ over problem instances in $\mathcal{Z}$.

To describe the performance of a parameterized algorithm, we adopt the notation previously introduced in this thesis. For every parameter setting $\rho \in \mathbb{R}$, there is a function $u_\rho : \mathcal{Z} \to [0, H]$ that measures, abstractly, the performance of the algorithm parameterized by $\rho$ given an input $z \in \mathcal{Z}$. For example, $u_\rho$ might measure runtime or the quality of the

algorithm's output. We use the notation $\mathcal{U} = \{u_\rho : \rho \in \mathbb{R}\}$ to denote the set of all performance functions.

Our bounds apply to the setting where on any fixed input $z$, the dual function $u_z : \mathbb{R} \to \mathbb{R}$ measuring the algorithm's performance as a function of the tunable parameter is piecewise constant with at most $t$ pieces, for some $t \in \mathbb{Z}$.

**Problem formulation.** A portfolio-based algorithm selection procedure relies on two key components: a *portfolio* and an *algorithm selector*. A portfolio is a set $\mathcal{P} = \{\rho_1, \ldots, \rho_\kappa\} \subseteq \mathbb{R}$ of $\kappa$ parameter settings. An algorithm selector is a mapping $f : \mathcal{Z} \to \mathcal{P}$ from problem instances $z \in \mathcal{Z}$ to parameter settings $f(z) \in \mathcal{P}$. In practice [Kadioglu et al., 2010, Sandholm, 2013, Xu et al., 2010], the portfolio and algorithm selector are typically learned using the following high-level procedure:

1. Choose a class $\mathcal{F}$ of algorithm selectors, each of which maps $\mathcal{Z}$ to $\mathbb{R}$. (In Section 9.3, we provide several examples of classes $\mathcal{F}$ used in practice.)
2. Draw a training set $\mathcal{S} = \{z_1, \ldots, z_N\} \sim \mathcal{D}^N$ of problem instances from the unknown distribution $\mathcal{D}$.
3. Use $\mathcal{S}$ to learn a portfolio $\hat{\mathcal{P}} = \{\rho_1, \ldots, \rho_\kappa\} \subseteq \mathbb{R}$.
4. Use $\mathcal{S}$ to learn an algorithm selector $\hat{f} \in \mathcal{F}$ that maps to parameter settings in the portfolio $\hat{\mathcal{P}}$.

Given an instance $z \in \mathcal{Z}$, the performance of the parameter setting selected by $\hat{f}$ is $u_{\hat{f}(z)}(z)$. We bound the expected quality $\mathbb{E}_{z\sim\mathcal{D}}\left[u_{\hat{f}(z)}(z)\right]$ of the learned algorithm selector.

**Road map.** We first analyze to what extent the average performance of the selector $\hat{f}$ over the training set generalizes to its expected performance on the distribution. We then use this analysis to relate the performance of the learned selector $\hat{f}$ and the optimal selector under the optimal choice of a portfolio. In particular, we bound the difference between $\mathbb{E}_{z\sim\mathcal{D}}\left[u_{\hat{f}(z)}(z)\right]$ and $\max_{\mathcal{P}:|\mathcal{P}|\leq\kappa} \mathbb{E}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}} u_\rho(z)\right]$. (Equivalently, if our goal is to minimize $u_\rho(z)$, we may replace each max with a min.)

## 9.2 Sample complexity bounds

In this section, we bound the difference between the average performance of any selector $f \in \mathcal{F}$ over the training set $\mathcal{S} \sim \mathcal{D}^N$ and its expected performance. Formally, we bound

$$\left| \frac{1}{N} \sum_{z\in\mathcal{S}} u_{f(z)}(z) - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_{f(z)}(z)\right] \right| \tag{9.1}$$

for any choice of an algorithm selector $f \in \mathcal{F}$. This will serve as a building block for our general analysis of portfolio-based algorithm selection.

Our bounds depend on both the number of dual function pieces $t$ and on the *intrinsic complexity* of the class of algorithm selectors $\mathcal{F}$. We use the following notion of the *multi-class projection* of $\mathcal{F}$ to define the class's intrinsic complexity.

**Definition 9.2.1.** Given a selector $f \in \mathcal{F}$, let $\rho_1 < \rho_2 < \cdots < \rho_{\bar{\kappa}}$ be the parameter settings $f$ maps to, with $\bar{\kappa} \leq \kappa$. The function $f$ defines a partition $Z_1, \ldots, Z_{\bar{\kappa}}$ of the problem instances $\mathcal{Z}$ where for any $z \in \mathcal{Z}$, if $f(z) = \rho_i$, then $z \in Z_i$. For each function $f \in \mathcal{F}$ there is therefore a corresponding multi-class function $\bar{f} : \mathcal{Z} \to [\kappa]$ that indicates which set of the partition the instance $z$ belongs to: $\bar{f}(z) = i$ when $z \in Z_i$. We use the notation $\bar{\mathcal{F}} = \left\{\bar{f} : f \in \mathcal{F}\right\}$ to denote the set of all such multi-class functions.

Defining this set of multi-class functions allows us to use classic tools from multi-class learning to reason about the algorithm selectors $\mathcal{F}$. In particular, our bounds depend on the *Natarajan [1989] dimension* of the class $\bar{\mathcal{F}}$, which is a natural extension of the classic VC dimension [Vapnik and Chervonenkis, 1971] to multi-class functions.

**Definition 9.2.2** (Natarajan dimension)**.** The set $\bar{\mathcal{F}}$ *multi-class shatters* a set of problem instances $z_1, \ldots, z_N$ if there exist labels $y_1, \ldots, y_N \in [\kappa]$ and $y'_1, \ldots, y'_N \in [\kappa]$ such that:

1. For every $i \in [N]$, $y_i \neq y'_i$, and
2. For any subset $C \subseteq [N]$, there exists a function $\bar{f} \in \bar{\mathcal{F}}$ such that $\bar{f}(z_i) = y_i$ if $i \in C$ and $\bar{f}(z_i) = y'_i$ otherwise.

The *Natarajan dimension* of $\bar{\mathcal{F}}$ is the cardinality of the largest set that can be multi-class shattered by $\bar{\mathcal{F}}$.

In Section 9.3, we bound the Natarajan dimension of $\bar{\mathcal{F}}$ for several commonly-used classes of algorithm selectors $\mathcal{F}$. We use Natarajan dimension to quantify the intrinsic complexity of the class of selectors, which in turn allows us to bound Equation (9.1) for every function $f \in \mathcal{F}$. To do so, we relate the Natarajan dimension of $\bar{\mathcal{F}}$ to the pseudo-dimension of the function class $\mathcal{U}_{\mathcal{F}} = \left\{z \mapsto u_{f(z)}(z) : f \in \mathcal{F}\right\}$. Every function in $\mathcal{U}_{\mathcal{F}}$ is defined by an algorithm selector $f \in \mathcal{F}$. On input $z \in \mathcal{Z}$, $u_{f(z)}(z)$ equals the utility of the algorithm parameterized by $f(z)$ on input $z$. We now prove a general bound on $\text{Pdim}\left(\mathcal{U}_{\mathcal{F}}\right)$, which allows us to bound Equation (9.1).

**Theorem 9.2.3.** *Suppose each dual function $u^*_z$ is piecewise-constant with at most $t$ pieces. Let $\bar{d}$ be the Natarajan dimension of $\bar{\mathcal{F}}$. Then* $\text{Pdim}\left(\mathcal{U}_{\mathcal{F}}\right) = \tilde{O}\left(\bar{d} + \kappa \log t\right)$.

At a high level, the $\tilde{O}\left(\bar{d}\right)$ term accounts for the intrinsic complexity of the algorithm selectors $\mathcal{F}$. The $O\left(\kappa \log t\right)$ term accounts for the complexity of composing selectors $f$ with the performance functions $u_\rho$. In Theorem 9.2.4, we prove this bound is tight up to logarithmic factors.

*Proof of Theorem 9.2.3.* Let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of problem instances that is shattered by $\mathcal{U}_{\mathcal{F}}$, as witnessed by the points $t_1, \ldots, t_N \in \mathbb{R}$. By definition, this means that

$$2^N = \left|\left\{\begin{pmatrix} \mathbf{1}_{\left\{u_{f(z_1)}(z_1) \leq t_1\right\}} \\ \vdots \\ \mathbf{1}_{\left\{u_{f(z_N)}(z_N) \leq t_N\right\}} \end{pmatrix} : f \in \mathcal{F}\right\}\right| \leq \left|\left\{\begin{pmatrix} u_{f(z_1)}(z_1) \\ \vdots \\ u_{f(z_N)}(z_N) \end{pmatrix} : f \in \mathcal{F}\right\}\right|. \tag{9.2}$$

Since each dual function $u^*_{z_i}$ is piecewise-constant with at most $t$ pieces, we know there are $M \leq Nt$ intervals $I_1, \ldots, I_M$ partitioning $\mathbb{R}$ where for any interval $I_j$ and any problem instance $z_i$, $u^*_{z_i}(\rho)$ is constant across all $\rho \in I_j$. We assume that the intervals are ordered so that if $j < j'$, then the points in $I_j$ are smaller than the points in $I_{j'}$

Let $J = (j_1, \ldots, j_{\bar{\kappa}}) \in [M]^{\bar{\kappa}}$ be a vector of $\bar{\kappa} \leq \kappa$ interval indices with $j_1 \leq j_2 \leq \cdots \leq j_{\bar{\kappa}}$. Let $\mathcal{F}_J \subseteq \mathcal{F}$ be the set of functions $f \in \mathcal{F}$ with the following property: letting $\rho_1 < \rho_2 < \cdots < \rho_{\bar{\kappa}}$ be the parameter settings $f$ maps to (i.e., $\{f(z) : z \in \mathcal{Z}\} = \{\rho_1, \ldots \rho_{\bar{\kappa}}\}$), we have that the $i^{th}$ parameter setting is in the $i^{th}$ interval: $\rho_1 \in I_{j_1}, \ldots, \rho_{\bar{\kappa}} \in I_{j_{\bar{\kappa}}}$. Since $I_1, \ldots, I_M$ partition $\mathbb{R}$ and since each function $f \in \mathcal{F}$ maps to at most $\kappa$ parameter settings, $\mathcal{F} = \cup_J \mathcal{F}_J$. Together with Equation (9.2), this means that

$$2^N \leq \sum_{\bar{\kappa}=1}^{\kappa} \sum_{J \in [M]^{\bar{\kappa}}} \left| \left\{ \begin{pmatrix} u_{f(z_1)}(z_1) \\ \vdots \\ u_{f(z_N)}(z_N) \end{pmatrix} : f \in \mathcal{F}_J \right\} \right|. \tag{9.3}$$

Fix a particular set $J = (j_1, \ldots, j_{\bar{\kappa}}) \in [M]^{\bar{\kappa}}$ as defined above. For each algorithm selector $f \in \mathcal{F}_J$, let $f_0 : \mathcal{Z} \to J$ be a function that indicates which of the $\bar{\kappa}$ intervals $I_{j_1}, \ldots, I_{j_{\bar{\kappa}}}$ the parameter setting $f(z)$ falls in. In other words, $f_0(z) = j$ if and only if $f(z) \in I_j$. Recall that for any $i \in [N]$ and $j \in J$, $u_{f(z_i)}(z_i)$ is constant across all $f \in \mathcal{F}_J$ with $f(z_i) \in I_j$. Therefore, even if we only know which of the $\bar{\kappa}$ intervals $f(z_i)$ falls in and not the function $f$ itself, we can correctly infer the value $u_{f(z_i)}(z_i)$. Said another way, if we only know the value $f_0(z_i) \in [\bar{\kappa}]$, we can infer the value $u_{f(z_i)}(z_i)$. Aggregating this logic across all $N$ problem instances, given a vector $(f_0(z_1), \ldots, f_0(z_N))$ we can directly infer the vector $\left(u_{f(z_1)}(z_1), \ldots, u_{f(z_N)}(z_N)\right)$. This implies that

$$\left| \left\{ \begin{pmatrix} u_{f(z_1)}(z_1) \\ \vdots \\ u_{f(z_N)}(z_N) \end{pmatrix} : f \in \mathcal{F}_J \right\} \right| \leq \left| \left\{ \begin{pmatrix} f_0(z_1) \\ \vdots \\ f_0(z_N) \end{pmatrix} : f \in \mathcal{F}_J \right\} \right|. \tag{9.4}$$

Next, we use a similar logic to show that

$$\left| \left\{ \begin{pmatrix} f_0(z_1) \\ \vdots \\ f_0(z_N) \end{pmatrix} : f \in \mathcal{F}_J \right\} \right| \leq \left| \left\{ \begin{pmatrix} \bar{f}(z_1) \\ \vdots \\ \bar{f}(z_N) \end{pmatrix} : f \in \mathcal{F}_J \right\} \right|. \tag{9.5}$$

To see why, suppose we only know the value $\bar{f}(z_i)$ and not the function $f$ itself. For ease of notation, say $\ell = \bar{f}(z_i)$. By definition of $\bar{f}$, we know that $f(z_i)$ is the $\ell^{th}$-smallest parameter setting that the function $f$ maps to. By definition of the function $f_0$, this implies that $f_0(z_i) = j_\ell$. Therefore, if we only know the value $\bar{f}(z_i)$ and not the function $f$ itself, we can correctly infer the value $f_0(z_i)$. Again, aggregating this logic across all $N$ problem instances, given a vector $\left(\bar{f}(z_1), \ldots, \bar{f}(z_N)\right)$ we can directly infer the vector $(f_0(z_1), \ldots, f_0(z_N))$. This implies that Equation (9.5) holds.

Combining Equations 9.4 and (9.5) with Natarajan's lemma [Natarajan, 1989], we have that

$$\left| \left\{ \begin{pmatrix} u_{f(z_1)}(z_1) \\ \vdots \\ u_{f(z_N)}(z_N) \end{pmatrix} : f \in \mathcal{F}_J \right\} \right| \leq N^{\bar{d}} \bar{\kappa}^{2\bar{d}}.$$

Combining this fact, the fact that $M \leq Nt$, and Equation (9.3), we have that $2^N \leq \kappa (Nt)^\kappa N^{\bar{d}} \kappa^{2\bar{d}}$, which implies that $N = O\left(\left(\kappa + \bar{d}\right) \log\left(\kappa + \bar{d}\right) + \kappa \log t\right)$. □

Theorem 9.2.3 implies that with probability $1-\delta$ over the draw $\mathcal{S} \sim \mathcal{D}^N$, for any selector $f \in \mathcal{F}$,

$$\left| \frac{1}{N} \sum_{z \in \mathcal{S}} u_{f(z)}(z) - \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ u_{f(z)}(z) \right] \right| = O\left( H \sqrt{\frac{1}{N} \left( \bar{d} + \kappa \log t + \log \frac{1}{\delta} \right)} \right). \tag{9.6}$$

This theorem quantifies a fundamental tradeoff: as the portfolio size increases, we can hope to obtain better and better empirical performance $\sum_{z \in \mathcal{S}} u_{f(z)}(z)$ but the generalization error $\tilde{O}\left( H \sqrt{\left( \bar{d} + \kappa \right) / N} \right)$ will worsen.

We now prove that Theorem 9.2.3 is tight up to logarithmic factors. The following theorem illustrates that even if the class of algorithm selectors is extremely simple (in that the Natarajan dimension of $\bar{\mathcal{F}}$ is 0), if the portfolio size (that is, the number $\kappa$ of parameters mapped to) is large, we cannot hope to avoid overfitting.

**Theorem 9.2.4.** *For any $\kappa, \bar{d} \geq 2$, there is a class of functions $\mathcal{U} = \left\{ u_\rho : \rho \in \mathbb{R} \right\}$ and a class of selectors $\mathcal{F}$ such that:*

1. *Each selector $f \in \mathcal{F}$ maps to $\leq \kappa$ parameter settings.*
2. *Each dual function $u_z^*$ is piecewise-constant with 1 discontinuity,*
3. *The Natarajan dimension of $\bar{\mathcal{F}}$ is at most $\bar{d}$, and*
4. *The pseudo-dimension of $\mathcal{U}_{\mathcal{F}}$ is $\Omega\left( \kappa + \bar{d} \right)$.*

*Proof.* Let $\mathcal{Z} = (0, 1]$. For each parameter setting $\rho \in \mathbb{R}$, define $u_\rho(z) = \mathbf{1}_{\{z \leq \rho\}}$. As claimed, each dual function $u_z^* : \mathbb{R} \to \mathbb{R}$ is piecewise-constant with 1 discontinuity. In this case, the function in $\mathcal{U}_{\mathcal{F}}$ map $\mathcal{Z}$ to $\{0, 1\}$. In the special case where the range of the function class is $\{0, 1\}$, pseudo-dimension is typically referred to as *VC dimension*, which we denote as $\text{VCdim}\left( \mathcal{U}_{\mathcal{F}} \right)$.

Let $\kappa, \bar{d} \geq 2$ be two arbitrary integers. We split this proof into two cases: $\bar{d} \geq \kappa$ and $\kappa > \bar{d}$. In both cases, we exhibit a class of selectors $\mathcal{F}$ that satisfies the properties in the theorem statement and we prove that $\text{VCdim}\left( \mathcal{U}_{\mathcal{F}} \right) \geq \max\left\{ \kappa, \bar{d} \right\} = \Omega\left( \kappa + \bar{d} \right)$.

**Claim 9.2.5.** *Suppose $\bar{d} \geq \kappa$. There exists a class of selectors $\mathcal{F}$ that satisfies the properties in the theorem statement and $\text{VCdim}\left( \mathcal{U}_{\mathcal{F}} \right) = \bar{d}$.*

*Proof of Claim 9.2.5.* Let $\mathcal{F} \subseteq \{0, 1\}^{\mathcal{Z}}$ be any set of binary functions with VC dimension $\bar{d}$. As required, each selector $f \in \mathcal{F}$ maps to at most $\kappa$ parameter settings ($|\{f(z) : z \in \mathcal{Z}\}| \leq 2 \leq \kappa$). Moreover, $\bar{\mathcal{F}} = \mathcal{F}$, so the Natarajan dimension of $\bar{\mathcal{F}}$ equals the VC dimension of $\mathcal{F}$, which is $\bar{d}$.

For any instance $z \in \mathcal{Z}$ and function $f \in \mathcal{F}$,

$$u_{f(z)}(z) = \begin{cases} 1 & \text{if } z \leq f(z) \\ 0 & \text{if } z > f(z). \end{cases}$$

Since $z \in (0, 1]$ and $f(z) \in \{0, 1\}$, this implies that $u_{f(z)}(z) = f(z)$. Therefore, $\text{VCdim}\left( \mathcal{U}_{\mathcal{F}} \right) = \text{VCdim}(\mathcal{F}) = \bar{d}$. □

**Claim 9.2.6.** *Suppose $\kappa > \bar{d}$. There exists a class of selectors $\mathcal{F}$ that satisfies the properties in the theorem statement and $\text{VCdim}\left( \mathcal{U}_{\mathcal{F}} \right) \geq \kappa$.*

*Proof of Claim 9.2.6.* We begin by partitioning $\mathcal{Z} = (0, 1]$ into $\kappa$ intervals $Z_1, \ldots, Z_\kappa$, where $Z_i = \left(\frac{i-1}{\kappa}, \frac{i}{\kappa}\right]$. For each set $T \subseteq [\kappa]$, define an selector $f_T : \mathcal{Z} \to \mathbb{R}$ as follows. For any $z \in \mathcal{Z} = (0, 1]$, let $i \in [\kappa]$ be the index of the interval $z$ lies in, i.e., $z \in Z_i$. We define

$$f_T(z) = \begin{cases} \frac{i}{\kappa} & \text{if } i \in T \\ \frac{i}{\kappa} - \frac{1}{2\kappa} & \text{if } i \notin T. \end{cases}$$

Let $\mathcal{F} = \{f_T : T \subseteq [\kappa]\}$. For every function $f \in \mathcal{F}$, $\bar{f}(z)$ equals the index $i \in [\kappa]$ such that $z \in Z_i$. Therefore, $\left|\bar{\mathcal{F}}\right| = 1$, so the Natarajan dimension of $\bar{\mathcal{F}}$ is $0 < \bar{d}$.

Define $\mathcal{S} = \left\{\frac{1}{\kappa}, \frac{2}{\kappa}, \ldots, \frac{\kappa-1}{\kappa}, 1\right\} \subset \mathcal{Z}$. We prove that $\mathcal{S}$ is shattered by $\mathcal{U}_\mathcal{F}$. Let $T \subseteq [\kappa]$ be an arbitrary subset. If $i \in T$, then $f_T\left(\frac{i}{\kappa}\right) = \frac{i}{\kappa}$, so

$$u_{f_T\left(\frac{i}{\kappa}\right)}\left(\frac{i}{\kappa}\right) = u_{\frac{i}{\kappa}}\left(\frac{i}{\kappa}\right) = \mathbf{1}_{\left\{\frac{i}{\kappa} \leq \frac{i}{\kappa}\right\}} = 1.$$

If $i \notin T$, then $f_T\left(\frac{i}{\kappa}\right) = \frac{i}{\kappa} - \frac{1}{2\kappa}$, so

$$u_{f_T\left(\frac{i}{\kappa}\right)}\left(\frac{i}{\kappa}\right) = u_{\frac{i}{\kappa} - \frac{1}{2\kappa}}\left(\frac{i}{\kappa}\right) = \mathbf{1}_{\left\{\frac{i}{\kappa} \leq \frac{i}{\kappa} - \frac{1}{2\kappa}\right\}} = 0.$$

Therefore, $\mathcal{S}$ is shattered by $\mathcal{U}_\mathcal{F}$, so the VC dimension of $\mathcal{U}_\mathcal{F}$ is at least $\kappa$. □

These two claims illustrate that $\text{VCdim}\left(\mathcal{U}_\mathcal{F}\right) \geq \max\left\{\kappa, \bar{d}\right\} = \Omega\left(\kappa + \bar{d}\right)$. □

In the proof of Theorem 9.2.4, each performance function $u_\rho$ maps to $\{0, 1\}$, so we effectively prove a lower bound on the *VC dimension* of $\mathcal{U}_\mathcal{F}$. Classic results from learning theory imply the generalization error of learning a selector $f \in \mathcal{F}$ can therefore be as large as $\tilde{\Omega}\left(H\sqrt{\left(\bar{d} + \kappa\right)/N}\right)$, which matches Equation (9.6) up to logarithmic factors.

## 9.3 Application of theory to algorithm selectors

We now instantiate Theorem 9.2.3 for several commonly-used classes of algorithm selectors. In each of the case studies, there is a feature mapping $\phi : \mathcal{Z} \to \mathbb{R}^m$ that assigns feature vectors $\phi(z) \in \mathbb{R}^m$ to problem instances $z \in \mathcal{Z}$.

### 9.3.1 Linear performance models

We begin by providing guarantees for algorithm selectors that use a linear performance model. These have been used extensively in computational research [Xu et al., 2008, 2010]. To define this type of selector, let $\boldsymbol{\rho} = (\rho_1, \ldots \rho_\kappa)$ be a set of $\kappa$ distinct parameter settings. For each $i \in [\kappa]$, define a vector $\boldsymbol{w}_i \in \mathbb{R}^m$ and let

$$W = \begin{pmatrix} | & \cdots & | \\ \boldsymbol{w}_1 & \ddots & \boldsymbol{w}_\kappa \\ | & \cdots & | \end{pmatrix}$$

be a matrix containing all $\kappa$ weight vectors. The dot product $\boldsymbol{w}_i \cdot \phi(z)$ is meant to estimate the performance of the algorithm parameterized by $\rho_i$ on instance $z$. We define the algorithm selector $f_{\boldsymbol{\rho}, W}(z) = \rho_i$ where $i = \text{argmax}_{j \in [\kappa]}\left\{\boldsymbol{w}_j \cdot \phi(z)\right\}$, which selects the parameter

setting with best predicted performance. We define the class of algorithm selectors $\mathcal{F}_L = \left\{f_{\boldsymbol{\rho},W} : W \in \mathbb{R}^{m\times\kappa}, \boldsymbol{\rho} \in \mathbb{R}^{\kappa}\right\}$. To define the class $\bar{\mathcal{F}}_L$, for each matrix $W \in \mathbb{R}^{m\times\kappa}$, let $g_W : \mathcal{Z} \to [\kappa]$ be a function where $g_W(z) = \text{argmax}_{i\in[\kappa]} \left\{\boldsymbol{w}_i \cdot \boldsymbol{\phi}(z)\right\}$. By definition, $\bar{\mathcal{F}}_L = \{g_W : W \in \mathbb{R}^{m\times\kappa}\}$, so $\bar{\mathcal{F}}_L$ is the well-studied *m-dimensional linear class* which has a Natarajan dimension of $O(m\kappa)$ [Shalev-Shwartz and Ben-David, 2014]. This fact implies the following corollary.

**Corollary 9.3.1.** *Suppose the dual functions are piecewise-constant with at most $t$ pieces. The pseudo-dimension of $\mathcal{U}_{\mathcal{F}_L} = \left\{z \mapsto u_{f(z)} : f \in \mathcal{F}_L\right\}$ is $O(\kappa m \log(\kappa m) + \kappa \log t)$.*

### 9.3.2 Regression tree performance models

We now analyze algorithm selectors that use a regression tree as the performance model. These have proven powerful in computational research [Hutter et al., 2014]. A regression tree $T$'s leaf nodes partition the feature space $\mathbb{R}^m$ into disjoint regions $R_1, \ldots, R_\ell$. In each region $R_i$, a constant value $c_i$ is used to predict the algorithm's performance on instances in the region. The internal nodes of the tree define this partition: each performs an inequality test on some feature of the input. We use the notation $h_T(z)$ to denote tree $T$'s prediction of the algorithm's performance on instance $z$. Formally, $h_T(z)$ equals the constant value corresponding to the region of the tree's partition to which $\boldsymbol{\phi}(z)$ belongs.

An algorithm selector can be defined using a regression tree performance model as follows. Let $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_\kappa)$ be a set of $\kappa$ distinct parameter settings. For each parameter setting $\rho_i$, let $T_i$ be a tree that is meant to predict the performance of the algorithm parameterized by $\rho_i$, and let $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$ be the set of all $\kappa$ trees. We define the algorithm selector $f_{\boldsymbol{\rho},\boldsymbol{T}}(z) = \rho_i$ where $i = \text{argmax}_{j\in[\kappa]} \left\{h_{T_j}(z)\right\}$. The class of algorithm selectors $\mathcal{F}_R$ consists of all functions $f_{\boldsymbol{\rho},\boldsymbol{T}}$ across all parameter vectors $\boldsymbol{\rho} \in \mathbb{R}^\kappa$ and all $\kappa$-tuples of regression trees $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$.

**Lemma 9.3.2.** *Suppose we limit ourselves to building regression trees with at most $\ell$ leaves. Then the Natarajan dimension of $\bar{\mathcal{F}}_R$ is $O(\ell\kappa \log(\ell\kappa m))$.*

*Proof.* We first sketch the proof of this lemma. For each $\kappa$-tuple of regression trees $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$, let $g_{\boldsymbol{T}} : \mathcal{Z} \to [\kappa]$ be a function where $g_{\boldsymbol{T}}(z) = \text{argmax}_{i\in[\kappa]} \{h_{T_i}(z)\}$. By definition, the set $\bar{\mathcal{F}}_R$ consists of the functions $g_{\boldsymbol{T}}$ across all $\kappa$-tuples of regression trees $\boldsymbol{T}$ with at most $\ell$ leaves. Let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of problem instances. Our goal is to bound the number of ways the functions $g_{\boldsymbol{T}}$ can label these instances. A single regression tree induces a partition of these $N$ problem instances defined by which leaf each instance is mapped to as we apply the tree's inequality tests. The key step in this proof is bounding the total number of partitions we can induce by varying the tree's inequality tests. We then generalize this intuition to bound the number of partitions $\kappa$ regression trees can induce as we vary all their parameters. Once the partition of each regression tree is fixed, the tree with the largest prediction for each problem instance depends on the relative ordering of the constants at the trees' leaves. There is a bounded number of possible relative orderings, and we aggregate all of these bounds to prove the lemma statement.

We now move on to the full proof of the lemma. In this proof, to simplify notation, we will denote the feature vector $\boldsymbol{\phi}(z)$ as $\boldsymbol{z} \in \mathbb{R}^m$. For each $\kappa$-tuple of regression trees $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$, let $g_{\boldsymbol{T}} : \mathcal{Z} \to [\kappa]$ be a function where $g_{\boldsymbol{T}}(z) = \text{argmax}_{i\in[\kappa]} \{h_{T_i}(z)\}$. By definition, the set $\bar{\mathcal{F}}_R$ consists of the functions $g_{\boldsymbol{T}}$ across all $\kappa$-tuples of regression trees $\boldsymbol{T}$ with at most $\ell$ leaves. We will assume, without loss of generality, that all trees are full.

Let $N$ be the the Natarajan dimension of $\bar{\mathcal{F}}_R$ and let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of $N$ problem instances that are multi-class shattered by $\bar{\mathcal{F}}_R$. This implies that

$$2^N \leq \left| \left\{ \begin{pmatrix} g_{\boldsymbol{T}}(z_1) \\ \vdots \\ g_{\boldsymbol{T}}(z_N) \end{pmatrix} : \boldsymbol{T} \text{ is a } \kappa\text{-tuple of regression trees} \right\} \right|. \tag{9.7}$$

In this proof, we show that the right-hand-side of this inequality is bounded by $m^{\kappa(\ell-1)}(N\ell)^{\ell\kappa}(\kappa\ell)^{\kappa\ell}$, which implies that $N = O(\ell\kappa \log(\ell\kappa m))$.

To this end, we begin by focusing on a single regression tree $T$. We analyze the number of ways the tree can partition the instances $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N$ as we vary the parameters of $T$. Each internal node of $T$ performances an inequality test on some feature of the input, so it is defined by a feature index $i \in [m]$ and a threshold $\theta \in \mathbb{R}$. Since there are $\ell$ leaves, there are $\ell - 1$ internal nodes. First, we fix the indices of all internal nodes, which leaves $\ell - 1$ real-valued thresholds to analyze. At a particular internal node $v$, let $j$ be the index of the feature on which the node performs an inequality test and let $\theta_v$ be the threshold (where the index $j$ is fixed by the threshold $\theta_v$ is not fixed). Whether or not the instance $\boldsymbol{z}_i$ would be sorted into the left or right child of the node depends on whether or not

$$z_i[j] \leq \theta_v \tag{9.8}$$

(where $z_i[j]$ is the $j^{th}$ coordinate of the vector $\boldsymbol{z}_i$). For each problem instance $\boldsymbol{z}_i$, there are therefore $\ell - 1$ hyperplanes splitting the set of thresholds $\mathbb{R}^{\ell-1}$ into regions where if we use thresholds from within any one region, the path that the instance $\boldsymbol{z}_i$ takes through the tree (from root to leaf) is constant. The same holds for all $N$ problem instances, leading to a total of $N(\ell - 1)$ hyperplanes in $\mathbb{R}^{\ell-1}$. In total, these hyperplanes split $\mathbb{R}^{\ell-1}$ into at most $(N\ell)^\ell$ regions where if we use the thresholds from within any one region, the path that each of the $N$ problem instances takes through the tree is constant [Buck, 1943]. Since this is true no matter how we fix the feature indices of each interval node, tuning all parameters of the tree $T$ (both the feature indices and the thresholds) can induce at most $m^{\ell-1}(N\ell)^\ell$ different partitions of the $N$ problem instances.

Said another way, for any tree $T$ and instance $z \in \mathcal{Z}$, let $\lambda_T(z) \in [\ell]$ be the index of the leaf that the instance $z$ is mapped to as we apply the inequality tests defined by the internal nodes of $T$. As we vary the tree $T$, the vector $(\lambda_T(z_1), \ldots, \lambda_T(z_N)) \in [\ell]^N$ will take on at most $m^{\ell-1}(N\ell)^\ell$ different values.

We now aggregate this reasoning across all $\kappa$ regression trees. For any instance $z \in \mathcal{Z}$ and any $\kappa$-tuple of regression trees $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$, let $\Lambda_{\boldsymbol{T}}(z) \in [\ell]^\kappa$ be a vector where for each $j \in [\kappa]$, the $j^{th}$ component of $\Lambda_{\boldsymbol{T}}(z)$ is the index of the leaf that the instance $z$ is mapped to as we apply the inequality tests defined by the tree $T_j$. In other words, $\Lambda_{\boldsymbol{T}}(z) = (\lambda_{T_1}(z), \ldots, \lambda_{T_\kappa}(z))$. As we vary $\boldsymbol{T}$, the matrix

$$\begin{pmatrix} \Lambda_{\boldsymbol{T}}(z_1) & \ldots & \Lambda_{\boldsymbol{T}}(z_N) \end{pmatrix} = \begin{pmatrix} \lambda_{T_1}(z_1) & \cdots & \lambda_{T_1}(z_N) \\ \vdots & \ddots & \vdots \\ \lambda_{T_\kappa}(z_1) & \cdots & \lambda_{T_\kappa}(z_N) \end{pmatrix}$$

will take on at most $m^{(\ell-1)\kappa}(N\ell)^{\ell\kappa}$ different values. After all, the first row of the matrix can take on at most $m^{\ell-1}(N\ell)^\ell$ different values as we vary the tree $T_1$, the second row can take on at most $m^{\ell-1}(N\ell)^\ell$ different values as we vary $T_2$, and so on.

Now, consider the set of all $\kappa$-tuples of regression trees $\boldsymbol{T}$ where $(\Lambda_{\boldsymbol{T}}(z_1), \ldots, \Lambda_{\boldsymbol{T}}(z_N))$ is constant. Across all such $\boldsymbol{T} = (T_1, \ldots, T_\kappa)$, we know exactly which leaf each instance $z_i$ maps to for all $\kappa$ trees. For each each instance $z_i$, the tree with the largest label—or in other words, the value of the multi-class function $g_{\boldsymbol{T}}(z_i)$—only depends on the relative order of the leaves' predictions. Since there is a total of $\kappa\ell$ leaves, there are at most $(\kappa\ell)^{\kappa\ell}$ such orderings. Combining this bound with the bound from the previous paragraph, we have that

$$\left| \left\{ \begin{pmatrix} g_{\boldsymbol{T}}(z_1) \\ \vdots \\ g_{\boldsymbol{T}}(z_N) \end{pmatrix} : \boldsymbol{T} \text{ is a } \kappa\text{-tuple of regression trees} \right\} \right| \leq m^{(\ell-1)\kappa}(N\ell)^{\ell\kappa}(\kappa\ell)^{\kappa\ell}.$$

From Equation (9.7), we have that $2^N \leq m^{(\ell-1)\kappa}(N\ell)^{\ell\kappa}(\kappa\ell)^{\kappa\ell}$, so $N = O(\ell\kappa\log(\ell\kappa m))$. □

**Corollary 9.3.3.** *Suppose the dual functions are piecewise-constant with at most $t$ pieces and we limit ourselves to building regression trees with at most $\ell$ leaves. Then* $\text{Pdim}\left(\mathcal{U}_{\mathcal{F}_R}\right) = O(\ell\kappa\log(\ell\kappa m) + \kappa\log t)$.

This pseudo-dimension bound reflects the end-to-end nature of our analysis, since the guarantee bounds the generalization error of both selecting the portfolio and training the regression tree performance model. This is why the bound grows with both the size of the portfolio ($\kappa$) and the complexity of the regression trees ($\ell$ and $m$).

### 9.3.3 Clustering-based algorithm selectors

We now provide guarantees for clustering-based algorithm selectors, which have also been used in prior research [Kadioglu et al., 2010]. This type of selector clusters the feature vectors $\phi(z_1), \ldots, \phi(z_N) \in \mathbb{R}^m$ and chooses a good parameter setting for each cluster. On a new instance $z$, the selector determines which cluster center is closest to $\phi(z)$ and runs the algorithm using the parameter setting assigned to that cluster. More formally, let $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_\kappa)$ be a set of parameter settings and let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\kappa \in \mathbb{R}^m$ be a set of vectors. We define the matrix

$$X = \begin{pmatrix} | & \cdots & | \\ \boldsymbol{x}_1 & \ddots & \boldsymbol{x}_\kappa \\ | & \cdots & | \end{pmatrix},$$

where each column $\boldsymbol{x}_i$ is meant to represent a cluster center. We define the algorithm selector $f_{\boldsymbol{\rho},X}(z) = \rho_i$ where $i = \text{argmin}_{j\in[\kappa]}\left\{\left\|\boldsymbol{x}_j - \phi(z)\right\|_p\right\}$, for some $\ell_p$-norm with $p \geq 1$. The class of algorithm selectors is $\mathcal{F}_C = \left\{f_{\boldsymbol{\rho},X} : \boldsymbol{\rho} \in \mathbb{R}^\kappa, X \in \mathbb{R}^{m\times\kappa}\right\}$.

**Lemma 9.3.4.** *For any $p \in [1, \infty)$, the Natarajan dimension of $\bar{\mathcal{F}}_C$ is $O(m\kappa\log(m\kappa p))$.*

*Proof.* We begin with a proof sketch. For each matrix $X$, let $g_X : \mathcal{Z} \to [\kappa]$ be defined such that

$$g_X(z) = \text{argmin}_{i\in[\kappa]}\left\{\|\boldsymbol{x}_i - \phi(z)\|_p^p\right\}.$$

By definition, $\bar{\mathcal{F}}_C = \{g_X : X \in \mathbb{R}^{m\times\kappa}\}$. Let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of problem instances. Our goal is to bound the number of ways the functions $g_X$ can label these instances as we vary $X \in \mathbb{R}^{m\times\kappa}$. We do so by analyzing, for each instance $z_i$, the boundaries in $\mathbb{R}^{m\times\kappa}$ where if we

shift $X$ from one side of the boundary to the other, the column in $X$ closest to $\phi(z_i)$ changes. We show that these boundaries are defined by multi-dimensional polynomials. We bound the total number of regions these boundaries induce in $\mathbb{R}^{m\times\kappa}$, which implies a bound on the Natarajan dimension of $\bar{\mathcal{F}}_C$.

We now move the full proof of the lemma. In this proof, to simplify notation, we will denote the feature vector $\phi(z)$ as $\boldsymbol{z} \in \mathbb{R}^m$. For each matrix $X \in \mathbb{R}^{m\times\kappa}$, let $g_X : \mathcal{Z} \to [\kappa]$ be a function where

$$g_X(z) = \text{argmin}_{i\in[\kappa]} \left\{ \|\boldsymbol{x}_i - \boldsymbol{z}\|_p^p \right\}.$$

By definition, $\bar{\mathcal{F}}_C = \{g_X : X \in \mathbb{R}^{m\times\kappa}\}$. Let $N$ be the Natarajan dimension of $\bar{\mathcal{F}}_C$ and let $z_1, \ldots, z_N \in \mathcal{Z}$ be a set of $N$ problem instances that are multi-class shattered by $\bar{\mathcal{F}}_C$. This implies that

$$2^N \leq \left| \left\{ \begin{pmatrix} g_X(z_1) \\ \vdots \\ g_X(z_N) \end{pmatrix} : X \in \mathbb{R}^{m\times\kappa} \right\} \right|. \tag{9.9}$$

In this proof, we analyze the partition of the parameter space $\mathbb{R}^{m\times\kappa}$ into regions where in any one region $R \subseteq \mathbb{R}^{m\times\kappa}$, across all matrices $X \in R$, the vector $(g_X(z_1), \ldots, g_X(z_N))$ is constant.

We begin by subdividing $\mathbb{R}^{m\times\kappa}$ into regions $P_1, \ldots, P_T \subseteq \mathbb{R}^{m\times\kappa}$ where in any one region $P$, across all $X \in P$, either the $\ell^{th}$ component of $\boldsymbol{z}_q$ is smaller than the $\ell^{th}$ component of $\boldsymbol{x}_j$, i.e. $z_q[\ell] \leq x_j[\ell]$, or vice versa (but not both) for all $q \in [N]$, $j \in [\kappa]$, and $\ell \in [m]$. This is partition is defined by $N\kappa m$ hyperplanes in $\mathbb{R}^{m\kappa}$, so there are $T \leq (N\kappa m + 1)^{m\kappa}$ such regions [Buck, 1943].

Next, fix one of these $T$ regions $P \subseteq \mathbb{R}^{m\times\kappa}$. Without loss of generality, assume that the $\ell^{th}$ component of $\boldsymbol{z}_q$ is smaller than the $\ell^{th}$ component of $\boldsymbol{x}_j$, i.e. $z_q[\ell] \leq x_j[\ell]$ for all $q \in [N]$, $j \in [\kappa]$, and $\ell \in [m]$. For any two labels $i, j \in [\kappa]$ and any $q \in [N]$, whether or not

$$\|\boldsymbol{x}_i - \boldsymbol{z}_q\|_p^p \geq \|\boldsymbol{x}_j - \boldsymbol{z}_q\|_p^p \tag{9.10}$$

directly depends on the sign of the polynomial

$$h_{q,i,j}(X) := \sum_{\ell=1}^{m} \left(x_i[\ell] - z_q[\ell]\right)^p - \left(x_j[\ell] - z_q[\ell]\right)^p.$$

We know there are at most $\left(N\kappa^2 p\right)^{m\kappa}$ regions partitioning $P$ so that in any one region $R$, across all $X \in R$, either $h_{q,i,j}(X) \leq 0$ or $h_{q,i,j}(X) > 0$ (but not both) for all $q \in [N]$ and $i, j \in [\kappa]$ [Anthony and Bartlett, 2009]. For any such region $R$, across all $X \in R$, all pairwise comparisons as in Equation (9.10) are fixed, so the vector $(g_X(z_1), \ldots, g_X(z_N))$ is constant. In total, there are at most $(N\kappa m + 1)^{m\kappa} \left(N\kappa^2 p\right)^{m\kappa} \leq (2N^2\kappa^3 p\ell)^{m\kappa}$ regions, which implies that

$$\left| \left\{ \begin{pmatrix} g_X(z_1) \\ \vdots \\ g_X(z_N) \end{pmatrix} : X \in \mathbb{R}^{m\times\kappa} \right\} \right| \leq (2N^2\kappa^3 p\ell)^{m\kappa}.$$

Combining this inequality with Equation (9.9), we have that $2^N \leq (2N^2\kappa^3 p\ell)^{m\kappa}$, so $N = O(mk\log(m\kappa p))$. □

Lemma 9.3.4 and Theorem 9.2.3 imply the following bound.

**Corollary 9.3.5.** *Suppose the dual functions are piecewise-constant with at most $t$ pieces. Then the pseudo-dimension of $\mathcal{U}_{\mathcal{F}_C}$ is $\tilde{O}(m\kappa + \kappa\log t)$.*

## 9.4 Learning procedure with guarantees

In this section, we use the results from the previous section to provide guarantees for the high-level learning procedure outlined in Section 9.1:

1. Draw a training set of problem instances $\mathcal{S} \sim \mathcal{D}^N$.
2. Use the training set $\mathcal{S}$ to select a set of $\kappa$ or fewer parameter settings $\hat{\mathcal{P}} \subseteq \mathbb{R}$.
3. Use $\mathcal{S}$ to learn an algorithm selector $\hat{f} \in \mathcal{F}$ that maps problem instances $z \in \mathcal{Z}$ to parameter settings $\hat{f}(z) \in \hat{\mathcal{P}}$.

Our guarantees depend on the quality of the portfolio $\hat{\mathcal{P}}$ and selector $\hat{f}$, as formalized by the following definition.

**Definition 9.4.1.** Given a training set $\mathcal{S} \subseteq \mathcal{Z}^N$ and parameters $\alpha \in (0,1]$, $\beta \in [0,1]$, and $\epsilon \in [0,1]$, we say the portfolio $\hat{\mathcal{P}}$ and the algorithm selector $\hat{f}$ are $(\alpha, \beta, \epsilon)$-optimal if:

1. The portfolio $\hat{\mathcal{P}}$ is nearly optimal over the training set in the sense that

$$\frac{1}{N} \sum_{z \in \mathcal{S}} \max_{\rho \in \hat{\mathcal{P}}} u_\rho(z) \geq \alpha \max_{\mathcal{P} \subset \mathbb{R}: |\mathcal{P}| \leq \kappa} \frac{1}{N} \sum_{z \in \mathcal{S}} \max_{\rho \in \mathcal{P}} u_\rho(z) - \beta.$$

   (The maximization means that performance is measured with respect to an oracle that selects an optimal algorithm parameter $\rho$ from the portfolio for each instance.)

2. The algorithm selector $\hat{f}$ returns high-performing parameter settings from the set $\hat{\mathcal{P}}$ in the sense that

$$\frac{1}{N} \sum_{z \in \mathcal{S}} u_{\hat{f}(z)}(z) \geq \frac{1}{N} \sum_{z \in \mathcal{S}} \max_{\rho \in \hat{\mathcal{P}}} u_\rho(z) - \epsilon. \tag{9.11}$$

For example, when algorithmic performance as a function of the parameters is piecewise constant, there are only a finite number of meaningfully different parameter values to choose among, one per piece. Then, since $\sum_{z \in \mathcal{S}} \max_{\rho \in \hat{\mathcal{P}}} u_\rho(z)$ is a submodular function of the portfolio $\hat{\mathcal{P}}$, we can use a greedy algorithm to select the portfolio $\hat{\mathcal{P}}$, and we obtain $\alpha = 1 - \frac{1}{e}$ and $\beta = 0$. Alternatively, an integer programming technique could be used to select the optimal portfolio from the finite set of candidate parameter values, in which case we would obtain $\alpha = 1$ and $\beta = 0$. Moreover, the value $\epsilon$ can be calculated directly from the training set.

The following theorem bounds the difference between the expected performance of the chosen selector $\hat{f}$ and an oracle that selects an optimal selector and an optimal portfolio.

**Theorem 9.4.2.** *Suppose that each dual function $u_z^*$ is piecewise constant with at most $t$ pieces. Given a training set $\mathcal{S} \subseteq \mathcal{Z}$ of size $N$, suppose we learn an $(\alpha, \beta, \epsilon)$-optimal portfolio $\hat{\mathcal{P}} \subset \mathbb{R}$ and algorithm selector $\hat{f} : \mathcal{Z} \to \hat{\mathcal{P}}$ in $\mathcal{F}$. With probability $1 - \delta$ over the draw of the training set $\mathcal{S} \sim \mathcal{D}^N$,*

$$\mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ u_{\hat{f}(z)}(z) \right] \geq \alpha \max_{\mathcal{P}: |\mathcal{P}| \leq \kappa} \mathbb{E} \left[ \max_{\rho \in \mathcal{P}} u_\rho(z) \right] - \epsilon - \beta - \tilde{O}\left( H \sqrt{\frac{\bar{d} + \kappa}{N}} \right),$$

*where $\bar{d}$ is the Natarajan dimension of $\bar{\mathcal{F}}$.*

*Proof.* First, let

$$\mathcal{P}^* = \text{argmax}_{\mathcal{P}\subset\mathbb{R}:|\mathcal{P}|\leq\kappa} \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}} u_\rho(z)\right].$$

A Hoeffding bound implies that with probability $1-\delta$,

$$\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}^*} u_\rho(z)\right] \leq \frac{1}{N}\sum_{z\in\mathcal{S}} \max_{\rho\in\mathcal{P}^*} u_\rho(z) + \tilde{O}\left(H\sqrt{\frac{1}{N}}\right).$$

Combining this inequality with Definition 9.4.1, we have that

$$\begin{aligned}\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}^*} u_\rho(z)\right] &\leq \frac{1}{\alpha}\left(\frac{1}{N}\sum_{z\in\mathcal{S}} \max_{\rho\in\hat{\mathcal{P}}} u_\rho(z) + \beta\right) + O\left(H\sqrt{\frac{1}{N}\log\frac{1}{\delta}}\right)\\ &\leq \frac{1}{\alpha}\left(\frac{1}{N}\sum_{z\in\mathcal{S}} u_{\hat{f}(z)}(z) + \epsilon + \beta\right) + O\left(H\sqrt{\frac{1}{N}\log\frac{1}{\delta}}\right).\end{aligned}$$

From Theorem 9.2.3, we know that with probability $1-\delta$,

$$\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\max_{\rho\in\mathcal{P}^*} u_\rho(z)\right] \leq \frac{1}{\alpha}\left(\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[u_{\hat{f}(z)}(z)\right] + \tilde{O}\left(H\sqrt{\frac{\bar{d}+\kappa}{N}}\right) + \epsilon + \beta\right).$$

Therefore, the theorem statement holds. □

By a parallel argument, we can obtain symmetric guarantees when our goal is to minimize rather than maximize a performance measure.

## 9.5 Experiments

We provide experiments that illustrate the tradeoff we investigated from a theoretical perspective in the previous sections: as we increase the portfolio size, we can hope to include a well-suited parameter setting for any problem instance, but it becomes increasingly difficult to avoid overfitting. We illustrate this in the context of integer programming algorithm configuration. We configure CPLEX, one of the most widely used commercial solvers. CPLEX uses the *branch-and-cut (B&C)* algorithm (branch-and-bound with cutting planes, primal heuristics, preprocessing, etc.) to solve integer programs (IPs). We tune a parameter $\rho \in [0,1]$ of CPLEX that controls its *variable selection policy*[2]. Specifically, $\rho$ parameterizes the *linear scoring rule* introduced in Section 4.2. As in Chapters 4 and 8, our goal is to find parameter settings leading to small trees, so we define $u_\rho(z)$ to be the size of the tree B&C builds. We aim to learn a portfolio $\hat{\mathcal{P}}$ and selector $\hat{f}$ resulting in small expected tree size $\mathbb{E}\left[u_{\hat{f}(z)}(z)\right]$.

[2]We override the default variable selection of CPLEX 12.8.0.0 using the C API. All experiments were run on a 64-core machine with 512 GB of RAM, a m4.16xlarge Amazon AWS instance, and a cluster of m4.xlarge Amazon AWS instances.

**Distribution over IPs.** As in Chapters 4 and 8, we analyze a distribution over IPs formulating the combinatorial auction winner determination problem under the OR-bidding language [Sandholm, 2002], which we generate using the Combinatorial Auction Test Suite [Leyton-Brown et al., 2000]. We use the "arbitrary" generator with 200 bids and 100 goods, resulting in IPs with around 200 variables, and the "regions" generator with 400 bids and 200 goods, resulting in IPs with around 400 variables. We define a heterogeneous distribution $\mathcal{D}$ as follows: with equal probability, we draw an instance from the "arbitrary" or "regions" distribution. To assign features to these IPs, we use all the features developed in prior research by Leyton-Brown et al. [2000] and Hutter et al. [2014], resulting in 140 features.

**Experimental procedure.** We first learn a portfolio of size 10 in the following way. We draw a training set of $M = 1000$ IPs $z_1, \ldots, z_M \sim \mathcal{D}$ and solve for the dual functions $u^*_{z_1}, \ldots, u^*_{z_M}$—which measure tree size as a function of the parameter $\rho$—using the algorithm described in Section 4.3.3. These functions are piecewise-constant with at most $t$ pieces, for some $t \in \mathbb{N}$. Therefore, there are at most $Mt$ parameter settings leading to different algorithmic performance over the training set. Let $\bar{\mathcal{P}}$ be this set of parameter settings. We use a greedy algorithm to select 10 parameter settings from $\bar{\mathcal{P}}$. First, we find a parameter setting $\rho_1$ which minimizes average tree size over the training set: $\rho_1 \in \operatorname{argmin} \sum_{i=1}^{M} u_{z_i^*}(\rho)$. Then, we find a parameter setting $\rho_2$ that minimizes average tree size when the better of $\rho_1$ or $\rho_2$ is used: $\rho_2 \in \operatorname{argmin} \sum_{i=1}^{M} \min\left\{u_{z_i^*}(\rho), u_{z_i^*}(\rho_1)\right\}$. We continue greedily until we have a portfolio $\hat{\mathcal{P}} = \{\rho_1, \ldots, \rho_{10}\}$.

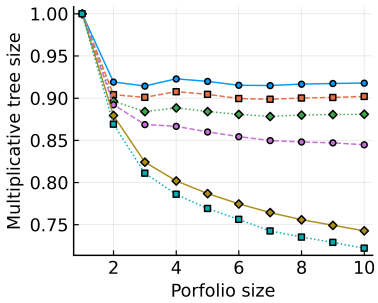
We then use a regression forest to select among parameter settings in the portfolio $\hat{\mathcal{P}}$. Prior research [Hutter et al., 2014] has illustrated that regression forests can be strong predictors of B&C runtime. Here, we use them to predict B&C tree size. A regression forest is a set $F = \{T_1, \ldots, T_M\}$ of regression trees (which we reviewed in Section 9.3.2). On an input IP $z$, the regression forest's prediction, denoted $h_F(z)$, is the average of the trees' predictions: $h_F(z) = \frac{1}{M} \sum_{i=1}^{M} h_{T_i}(z)$. We learn regression forests $F_1, \ldots, F_{10}$ for each of the 10 parameter settings in the portfolio $\hat{\mathcal{P}}$. We then define the algorithm selector $\hat{f}(z) = \rho_i$ where $i = \operatorname{argmin} \{h_{F_1}(z), \ldots, h_{F_{10}}(z)\}$.

To learn the regression forest, we draw a training set $z_1, \ldots, z_N \sim \mathcal{D}$ of IPs (with $N$ specified below). For each parameter setting $\rho_i \in \hat{\mathcal{P}}$ and IP $z_j$, we compute $u_{\rho_i}(z_j)$, the size of the tree B&C builds using the parameter setting $\rho_i$. We then train the regression forest $F_i$ corresponding to the parameter setting $\rho_i$ using the labeled training set $\{(z_1, u_{z_1}(\rho_i)), \ldots, (z_N, u_{z_N}(\rho_i))\}$. We use Python's scikit-learn regression forest implementation [Pedregosa et al., 2011] with the default parameter settings.

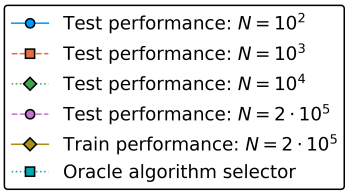
In Figure 9.1a, we plot the performance of the regression forests as we increase the sizes of both the training set and the portfolio. We denote the training set size as $N$, which ranges from 100 to 200,000. For a given choice of $N$, we first train the 10 regression forests $F_1, \ldots, F_{10}$ using the method described above. We then evaluate performance as a function of the portfolio size. Specifically, for each portfolio size $\kappa \in [10]$, we define an algorithm selector $\hat{f}_\kappa(z) = \rho_i$ where $i = \operatorname{argmin} \{h_{F_1}(z), \ldots, h_{F_\kappa}(z)\}$. We draw $N_t = 10^4$ test instances $\mathcal{S}_t \sim \mathcal{D}^{N_t}$ and evaluate the performance of $\hat{f}_\kappa$ on the test set. We denote the average test performance as

$$\hat{v}_\kappa = \frac{1}{N_t} \sum_{z \in \mathcal{S}_t} u_{\hat{f}_\kappa(z)}(z). \tag{9.12}$$

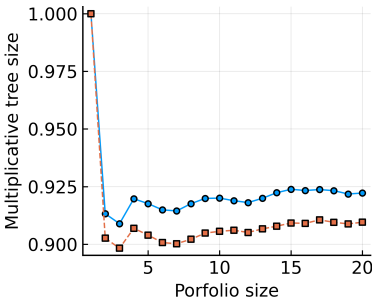
In Figure 9.1a, we plot the multiplicative performance improvement we obtain as we increase




(a) Plot with portfolio sizes 1 through 10.




(b) Legend for Figures 9.1a and 9.1c.




(c) Plot with portfolio sizes 1 through 20.

Figure 9.1: In Figures 9.1a and 9.1c, we plot the multiplicative tree size improvement we obtain as we increase both the portfolio size along the horizontal axis and the size of the training set, denoted $N$. Fixing a training set size and letting $\hat{v}_\kappa$ be the average tree size we obtain over the test set using a portfolio of size $\kappa$ (see Equation (9.12)), we plot $\hat{v}_\kappa / \hat{v}_1$. In Figure 9.1a, the portfolio size ranges from 1 to 10 and the training set size $N$ ranges from 100 to 200,000. In Figure 9.1c, the portfolio size ranges from 1 to 20 and the training set size ranges from 100 to 1000. In Figure 9.1a, we also plot a similar curve for the test performance of the oracle algorithm selector, as well as the training performance of the learned algorithm selector when $N = 2 \cdot 10^5$.

$\kappa$. Specifically, we plot $\hat{v}_\kappa / \hat{v}_1$. These are the blue solid ($N = 10^2$), orange dashed ($N = 10^3$), green dotted ($N = 10^4$), and purple dashed ($N = 2 \cdot 10^5$) lines. By the iterative fashion we constructed the portfolio, $\hat{v}_1$ is the performance of the best single parameter setting for the particular distribution, so $\hat{v}_1$ is already highly optimized.

We plot a similar curve for the test performance of the oracle algorithm selector which always selects the optimal parameter setting from the portfolio. Specifically, for each portfolio size $\kappa \in [10]$, let $f_\kappa^*$ be the oracle algorithm selector $f_\kappa^*(z) = \operatorname{argmin}_{\rho_1,\ldots,\rho_\kappa} u_{\rho_i}(z)$. Given a test set $\mathcal{S}_t \sim \mathcal{D}^{N_t}$, we define the average test performance of $f_\kappa^*$ as

$$v_\kappa^* = \frac{1}{N_t} \sum_{z \in \mathcal{S}_t} u_{f_\kappa^*(z)}(z) .$$

The blue dotted line equals $v_\kappa^* / v_1^*$ as a function of $\kappa$.

Finally, when the training set is of size $N = 2 \cdot 10^5$, we provide a similar curve for the training performance of the learned algorithm selectors $\hat{f}_\kappa$. Letting $z_1, \ldots, z_N$ be the training set, we denote the average training performance as

$$\tilde{v}_\kappa = \frac{1}{N} \sum_{i=1}^{N} u_{f_\kappa^*(z_i)}(z_i) .$$

The yellow solid line equals $\tilde{v}_\kappa / \tilde{v}_1$ as a function of the portfolio size $\kappa$.

In Figure 9.1c, we plot $\hat{v}_\kappa / \hat{v}_1$ as a function of the portfolio size $\kappa$ for larger portfolio sizes ranging from 1 to 20. We greedily extend the portfolio $\hat{\mathcal{P}}$ to include an additional 20 parameter settings. We then train 20 regression forests using freshly drawn training sets of size 100 and 1000. This plot illustrates the fact that as we increase the portfolio size, overfitting causes test performance to worsen.

**Discussion.** Focusing first on test performance using the largest training set size $N = 2 \cdot 10^5$, we see that test performance continues to improve as we increase the portfolio size, though training and test performance steadily diverge. This illustrates the tradeoff we investigated from a theoretical perspective in this chapter: as we increase the portfolio size, we can hope to include a well-suited parameter setting for every instance, but the generalization error will worsen. Figure 9.1c shows that for a given training set size, there is a portfolio size after which test performance actually starts to get strictly worse, as our theory predicts. In other words, we observe overfitting: the learned algorithm selector has strong average performance over the training set but poor test performance.

# Chapter 10

# 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 5, 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 example, 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 [Krishna, 2002]. Similarly, electricity generators in the U.K. use discriminatory auctions to sell their output [Krishna, 2002]. Sponsored search auctions are typically implemented using variants of the generalized second-price auction [Edelman et al., 2007, Varian, 2007]. Many major display ad exchanges including Google and Index Exchange have transitioned to first-price auctions, driven by ad buyers who believe it offers a higher degree of transparency [Despotakis et al., 2019, Harada, 2018, Parkin, 2018, Sluis, 2019]. Finally, nearly all fielded combinatorial auctions are manipulable. Essentially all combinatorial sourcing auctions are implemented using the first-price mechanism [Sandholm, 2013]. Combinatorial spectrum auctions are conducted using a variety of manipulable mechanisms. Even the "incentive auction" [Leyton-Brown et al., 2017] 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 [Nguyen and Sandholm, 2015].

A variety of reasons have been suggested to help explain why manipulable mechanisms are used in practice. To name a few,

1. Oftentimes, the rules are easier to explain, as exemplified by the exceptionally simple first-price auction.

2. Incentive compatibility can cease to hold when determining one's own valuation is costly (for example, due to computation or information gathering effort), even for the classic Vickrey auction [Sandholm, 2000]. This is also true of the Vickrey-Clarke-Groves (VCG) mechanism, which requires bidders to submit bids for every bundle, a task that generally requires a prohibitive amount of valuation computation or information acquisition [Conen and Sandholm, 2001, Parkes, 1999, Sandholm, 1993, Sandholm and Boutilier, 2006].

3. Single-shot incentive compatible mechanisms are generally not incentive compatible when the bid-taker uses bids from one auction to adjust the parameters of later auctions [Acquisti and Varian, 2005, Amin et al., 2013, Sandholm, 2013].

4. Incentive compatible mechanisms may leak the agents' private information [Rothkopf et al., 1990].

5. Incentive compatibility typically ceases to hold if agents are not risk neutral (i.e., the utility functions are not quasi-linear).

Due in part to the ubiquity of manipulable mechanisms, a growing body of additional research has explored mechanisms that are not incentive compatible [Archer et al., 2004, Azevedo and Budish, 2018, Conitzer and Sandholm, 2007, Dekel et al., 2010, Dütting et al., 2015, 2019, Feng et al., 2018, Golowich et al., 2018, Kothari et al., 2003, Lubin and Parkes, 2012, Mennle and Seuken, 2014]. A popular and widely-studied relaxation of incentive compatibility is $\gamma$-*incentive compatibility* [Archer et al., 2004, Azevedo and Budish, 2018, Dekel et al., 2010, Dütting et al., 2015, Kothari et al., 2003, Lubin and Parkes, 2012, Mennle and Seuken, 2014], which requires that no agent can improve his utility by more than $\gamma$ when he misreports his type.

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

## 10.1 Our contributions

Much of the literature on $\gamma$-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 [Likhodedov and Sandholm, 2004, 2005, Sandholm and Likhodedov, 2015]. 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

[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.

| Mechanisms | Upper bound on estimation error |
|---|---|
| First-price single-item auction | $\tilde{O}\left(\frac{\kappa^{-1}+n}{\sqrt{N}}\right)$ |
| First-price combinatorial auction over $\ell$ 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 10.1: Our estimation error upper bounds. The value $\kappa$ is defined such that $[0, \kappa]$ contains the range of the density functions defining the agents' type distribution. There are $n$ buyers and $N$ samples.

subset of the type space. We bound the difference between our incentive compatibility estimate and the true incentive compatibility approximation factor $\gamma$.

Accurately estimating the approximation factor $\gamma$ provides the mechanism designer with a data-dependent metric for choosing among manipulable mechanisms, which can be used in conjunction with other metrics such as expected revenue. When the distribution over agents' types is unknown, as we assume it is, one cannot use a distribution-dependent, analytic approach to deriving the approximation factor $\gamma$. However, a distribution-independent bound on the approximation factor that holds even for a worst-case distribution over types may be uninformative when the underlying distribution is not worst case. The estimate of the distribution's incentive compatibility approximation factor that we provide allows a mechanism designer to make a more informed decision when selecting a manipulable mechanism to field. As a concrete example, deep learning is increasingly being used to learn mechanisms from within classes of manipulable mechanisms [Dütting et al., 2019, Feng et al., 2018, Golowich et al., 2018, Shen et al., 2019]. Our guarantees can be used to better understand how susceptible the resulting mechanism is to strategic manipulation.

We apply our estimation technique to a variety of auction classes, summarized in Table 10.1. 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 [Harada, 2018, Parkin, 2018]. 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* [Brandt et al., 2007, Morgan et al., 2003, Sharma and Sandholm, 2010, Tang and Sandholm, 2012b], 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 [Edelman

et al., 2007], 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 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 [Krishna, 2002]. 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* [Conitzer and Sandholm, 2007], a subfield of *automated mechanism design* [Conitzer and Sandholm, 2002, Sandholm, 2003], 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 $\gamma$ defined such that no agent can misreport his type in order to improve his expected utility by more than $\gamma$, no matter his true type. We propose estimating $\gamma$ 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 $\theta$ and $\hat{\theta}$ where an agent with type $\theta$ 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 $\gamma$?

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 $\gamma$. To do so, we rely on the notion of *dispersion* [Balcan et al., 2018b]. 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 $\boldsymbol{\theta}$. 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 10.4.1, 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. [2018b]: 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* [Pollard, 1984]), then our estimate $\hat{\gamma}$ quickly converges to the true approximation factor $\gamma$ as the number of samples grows.

## 10.2 Additional related research

### 10.2.1 Prior and contemporaneous research

**Strategy-proofness in the large.** Azevedo and Budish [2018] 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* $\gamma$-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 [2013] 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 [Dütting et al., 2019, Feng et al., 2018, Golowich et al., 2018, Shen et al., 2019], 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. [2019] 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 10.3.1), no matter her true type, where the expectation is over the other agents' types. In contrast, Dütting et al. [2019] 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. [2019] analyze are disjoint: Dütting et al. [2019] 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, focusing on incentive compatible mechanism classes (cited in Section 5.1). In this model, the mechanism designer receives samples from the type distribution which she uses to find a mechanism with high average empirical revenue over the samples. These papers provided generalization guarantees that bound the generalization error of this approach, which is the difference between a mechanism's average revenue over the samples and its expected revenue.

By contrast, in this chapter, the estimation error of our IC approximation factor has two sources: 1) our discretization of the type space and 2) the fact that we do not know the type distribution, but only have samples from it. To bound the first source of error, we cannot rely on generalization bounds, which is why we rely on the notion of *dispersion*, as discussed in Section 10.1. Dispersion was originally introduced in a very different context: online and private optimization of piecewise-Lipschitz functions [Balcan et al., 2018b]. We demonstrate its utility in this context as well.

In a related direction, Chawla et al. [2014, 2016, 2017] study counterfactual revenue estimation. Given two auctions, they provide techniques for estimating one of the auction's equilib-

rium 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. [2018] 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.

### 10.2.2 Subsequent research

**Sample complexity of learning interim utilities.** In closely-related follow-up research, Fu and Lin [2020] bound the sample complexity of learning agents' interim utilities when the agents follow *any* monotone bidding strategy, complete with nearly-matching lower bounds. Their guarantees apply to the first-price and all-pay auctions.

**New incentive compatibility metrics.** Deng et al. [2020] propose a new metric to measure how far from incentive compatible a manipulable mechanism is. It applies to single-parameter mechanism design problems in both static and dynamic settings. They argue that their metric can often be calculated more efficiently than traditional incentive compatibility approximation factors.

**Certifying the approximate incentive compatibility of auction networks.** Drawing inspiration from the neural network verification and adversarial learning literature, Curry et al. [2020] propose a technique for providing a certifiable upper bound on the utility an agent can gain by misreporting their bid under the RegretNet architecture of Dütting et al. [2019].

## 10.3 Preliminaries and notation

There are $n$ agents who each have a *type*. We denote agent $i$'s type as $\theta_i$, which is an element of a (potentially infinite) set $\Theta_i$. 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 $\hat{\boldsymbol{\theta}} = \left(\hat{\theta}_1, \ldots, \hat{\theta}_n\right)$. 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 $\theta_i$, 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 $\mathcal{D}$.

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

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

$[-1, 1]$. 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 $\gamma$-*incentive compatible* if each agent can increase her utility by an additive factor of at most $\gamma$ by misreporting her type [Archer et al., 2004, Azevedo and Budish, 2018, Conitzer and Sandholm, 2007, Dekel et al., 2010, Dütting et al., 2015, 2019, Feng et al., 2018, Golowich et al., 2018, Kothari et al., 2003, Lubin and Parkes, 2012, Mennle and Seuken, 2014]. In this chapter, we concentrate on *ex-interim* approximate incentive compatibility [Azevedo and Budish, 2018, Lubin and Parkes, 2012].

**Definition 10.3.1.** A mechanism $M$ is ex-interim $\gamma$-*incentive compatible* if for each $i \in [n]$ and all $\theta_i, \hat{\theta}_i \in \Theta_i$, agent $i$ with type $\theta_i$ can increase her expected utility by an additive factor of at most $\gamma$ 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 [Balcan et al., 2019], 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.

## 10.4 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 [Azevedo and Budish, 2018, Babaioff et al., 2017, Cai and Daskalakis, 2017a, Cai et al., 2016, Goldner and Karlin, 2016, Hart and Nisan, 2012, Lubin and Parkes, 2012, Yao, 2014] that the agents' types are independently distributed. In other words, for each agent $i \in [n]$, there exists a distribution $\phi_i$ over their type space $\Theta_i$ 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* $\theta_i, \hat{\theta}_i \in \Theta_i$, $\mathbb{E}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}}\left[u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right)\right] \leq \hat{\gamma}_M$.

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 $\theta_i$ 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:

$$\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\}. \tag{10.1}$$

[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 $\theta_i$ 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 $\theta_i$ 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 $\boldsymbol{\theta}_{-i}, \hat{\boldsymbol{\theta}}_{-i} \in \times_{j \neq i} \Theta_j$, $u_{i,M}\left(\left(\theta_i, \boldsymbol{\theta}_{-i}\right), \left(\theta_i, \hat{\boldsymbol{\theta}}_{-i}\right)\right) \geq u_{i,M}\left(\left(\theta_i, \boldsymbol{\theta}_{-i}\right), \left(\hat{\theta}_i, \hat{\boldsymbol{\theta}}_{-i}\right)\right)$.

Unfortunately, in full generality, there might not be a finite-time procedure to solve this optimization problem, so in Section 10.4.1, 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 $M \in \mathcal{M}$,

$$\begin{aligned}&\max_{\theta_i,\hat{\theta}_i\in\Theta_i}\left\{\mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}_{-i}}\left[u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}\right)-u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}\right)\right]\right\}\\ &\leq \max_{\theta_i,\hat{\theta}_i\in\Theta_i}\left\{\frac{1}{N}\sum_{j=1}^{N}u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right)-u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\}+\epsilon_{\mathcal{M}}(N,\delta),\end{aligned} \tag{10.2}$$

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 (10.2) 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 $\theta_i$ 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 10.4.1.** *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 $d_i = \operatorname{Pdim}\left(\mathcal{F}_{i,\mathcal{M}}\right)$.*

*Proof.* Fix an arbitrary agent $i \in [n]$. By Theorem 2.1.3, we know that with probability at least $1-\delta/n$ over the draw of $\mathcal{S}_{-i}$, for all mechanisms $M \in \mathcal{M}$ and all $\theta_i, \hat{\theta}_i \in \Theta_i$,

$$\mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}_{-i}}\left[u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}\right)\right] - \frac{1}{N}\sum_{j=1}^{N}u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right) \leq \sqrt{\frac{2d_i}{N}\ln\frac{eN}{d_i}} + \sqrt{\frac{1}{2N}\ln\frac{2n}{\delta}}$$

and

$$\frac{1}{N}\sum_{j=1}^{N}u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right) - \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}_{-i}}\left[u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}\right)\right] \leq \sqrt{\frac{2d_i}{N}\ln\frac{eN}{d_i}} + \sqrt{\frac{1}{2N}\ln\frac{2n}{\delta}}.$$

Therefore,

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

Since the above inequality holds for all $i\in[n]$ with probability $1-\delta/n$, a union bound implies that with probability $1-\delta$, for all $i\in[n]$ and all mechanism $M\in\mathcal{M}$,

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

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 $d_i=\mathrm{Pdim}\left(\mathcal{F}_{i,\mathcal{M}}\right)$. □

### 10.4.1 Incentive compatibility guarantees via finite covers

In the previous section, we presented an empirical estimate of the *ex-interim* incentive compatibility approximation factor (Equation (10.1)) and we showed that it quickly converges to the true approximation factor. However, there may not be a finite-time procedure for computing Equation (10.1) in its full generality, restated below:

$$
\max_{\theta_i,\hat{\theta}_i\in\Theta_i}\left\{\frac{1}{N}\sum_{j=1}^{N}u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right)-u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\}. \tag{10.3}
$$

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 $\mathcal{G}\subset\Theta_i\times\Theta_i$, and approximate Equation (10.3) 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 $\mathcal{G}$, averaged over all profiles in $\mathcal{S}_{-i}$. In other words, we estimate Equation (10.3) as:

$$
\max_{\left(\theta_i,\hat{\theta}_i\right)\in\mathcal{G}}\left\{\frac{1}{N}\sum_{j=1}^{N}u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right)-u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\}. \tag{10.4}
$$

This raises two natural questions: how do we select the cover $\mathcal{G}$ and how close are the optimal solutions to Equations (10.3) and (10.4)? We provide two simple, intuitive approaches to selecting the cover $\mathcal{G}$. The first is to run a greedy procedure (see Section 10.4.1) 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 10.4.1).

**Algorithm 7** 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 $\mathcal{G} \leftarrow \emptyset$.

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 $\mathcal{G}$.

**Output:** The cover $\mathcal{G} \subseteq \Theta_i \times \Theta_i$.

### Covering via a greedy procedure

In this section, we show how to construct the cover $\mathcal{G}$ of $\Theta_i \times \Theta_i$ greedily, based off a classic learning-theoretic algorithm. We then show that when we use the cover $\mathcal{G}$ to estimate the incentive compatibility approximation factor (via Equation (10.4)), the estimate quickly converges to the true approximation factor. This greedy procedure is summarized by Algorithm 7. 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 7:

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

Note that the solution to Equation (10.3) 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 $\mathcal{G} \subseteq \Theta_i \times \Theta_i$ 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 $\mathcal{G}$. Classic results from learning theory [Anthony and Bartlett, 2009] 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 (10.4) when the cover is constructed using Algorithm 7.

**Theorem 10.4.2.** *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 $\mathcal{G}(\mathcal{S}_{-i}, M, \epsilon)$ be the cover returned by Algorithm 7. 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 $i \in [n]$,*

$$\max_{\theta_i, \hat{\theta}_i \in \Theta_i} \left\{ \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}} \left[ u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\theta_i, \theta_i, \boldsymbol{\theta}_{-i}\right) \right] \right\}$$

$$\leq \max_{\left(\theta_i,\hat{\theta}_i\right)\in\mathcal{G}(\mathcal{S}_{-i},M,\epsilon)} \left\{ \frac{1}{N}\sum_{j=1}^{N} u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\} + \epsilon + \tilde{O}\left(\sqrt{\frac{d_i}{N}}\right), \tag{10.5}$$

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

*Proof.* Let $\theta_i^*$ and $\hat{\theta}_i^*$ be the types in $\Theta_i$ that maximize

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

By definition of the set $\mathcal{G}\left(\mathcal{S}_{-i},M,\epsilon\right)$, we know there exists a pair $\left(\theta_i',\hat{\theta}_i'\right) \in \mathcal{G}\left(\mathcal{S}_{-i},M,\epsilon\right)$ such that

$$\begin{aligned}
&\max_{\theta_i,\hat{\theta}_i\in\Theta_i} \left\{ \frac{1}{N}\sum_{j=1}^{N} u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\} \\
=\ & \frac{1}{N}\sum_{j=1}^{N} u_{i,M}\left(\theta_i^*,\hat{\theta}_i^*,\boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i^*,\theta_i^*,\boldsymbol{\theta}_{-i}^{(j)}\right) \\
\leq\ & \frac{1}{N}\sum_{j=1}^{N} u_{i,M}\left(\theta_i',\hat{\theta}_i',\boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i',\theta_i',\boldsymbol{\theta}_{-i}^{(j)}\right) + \epsilon \\
\leq\ & \max_{\theta_i,\hat{\theta}_i\in\mathcal{G}(\mathcal{S}_{-i},M,\epsilon)} \left\{ \frac{1}{N}\sum_{j=1}^{N} u_{i,M}\left(\theta_i,\hat{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\theta_i,\theta_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\} + \epsilon.
\end{aligned}$$

Therefore, by this inequality and Theorem 10.4.1, we know that with probability at least $1-\delta$,

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

The bound on $\left|\mathcal{G}\left(\mathcal{S}_{-i},M,\epsilon\right)\right|$ follows from Lemma 10.4.3. □

**Lemma 10.4.3.** *For any $\mathcal{G}$ returned by Algorithm 7, $|\mathcal{G}| \leq (8eN/(\epsilon d_i))^{2d_i}$, where $d_i = \text{Pdim}(\mathcal{F}_{i,\mathcal{M}})$.*

*Proof.* Let $\mathcal{N}_1(\epsilon, U)$ be the $\epsilon$-covering number of $U$ and let $\mathcal{P}_1(\epsilon, U)$ be the $\epsilon$-packing number of $U$. In other words, $\mathcal{N}_1(\epsilon, U)$ is the size of the smallest set $V' \subseteq \mathbb{R}^N$ such that for all $v \in U$, there is a vector $v' \in V'$ such that $\|v - v'\|_1 \leq \epsilon$, and $\mathcal{P}_1(\epsilon, U)$ is the size of the largest set $P \subseteq U$ such that for all $v, v' \in P$, if $v \neq v'$, then $\|v - v'\|_1 \geq \epsilon$. As Anthony and Bartlett [2009] prove, $\mathcal{P}(\epsilon, U) \leq \mathcal{N}(\epsilon/2, U)$.

By construction, for every pair of vectors $v$ and $v'$ in the set $V$ defined in Algorithm 7, we know that $\|v - v'\|_1 \geq \epsilon$. Therefore, $|V| \leq \mathcal{P}(\epsilon, U) \leq \mathcal{N}(\epsilon/2, U)$. In the following claim, we prove that $\mathcal{N}(\epsilon/2, U) \leq (8eN/(\epsilon d_i))^{2d_i}$, which proves the lemma.

**Claim 10.4.4.** $\mathcal{N}(\epsilon/2, U) \leq (8eN/(\epsilon d_i))^{2d_i}$.

*Proof.* Let $U'$ be the set of vectors

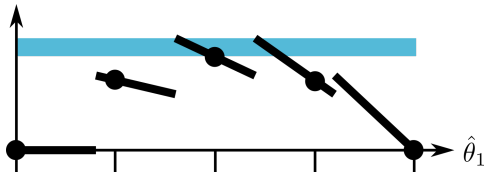
$$U' = \left\{ \frac{1}{N} \begin{pmatrix} u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \\ \vdots \\ u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(N)}\right) \end{pmatrix} : \theta_i, \hat{\theta}_i \in \Theta_i \right\}.$$

Note that $U \subseteq \{v - v' : v, v' \in U'\}$. We claim that $\mathcal{N}(\epsilon/2, U) \leq \mathcal{N}(\epsilon/4, U')^2 \leq (8eN/(\epsilon d_i))^{2d_i}$, where the second inequality follows from a theorem by Anthony and Bartlett [2009]. To see why the first inequality holds, suppose $C'$ is an $\frac{\epsilon}{4}$-cover of $U'$ with minimal size. Let $C = \{c - c' : c, c' \in C'\}$. Then for all $v \in U$, we know that $v = v' - v''$ for some $v', v'' \in U'$. Moreover, we know that there are vectors $c', c'' \in C'$ such that $\|v' - c'\|_1 \leq \epsilon/4$ and $\|v'' - c''\|_1 \leq \epsilon/4$. Therefore, $\|v - (c' - c'')\|_1 = \|(v' - v'') - (c' - c'')\|_1 \leq \epsilon/2$. Since $c' - c'' \in C$, we have that $C$ is a cover of $U$. Therefore, $\mathcal{N}(\epsilon/2, U) \leq |C| \leq |C'|^2 = \mathcal{N}(\epsilon/4, U')^2 \leq (8eN/(\epsilon d_i))^{2d_i}$. □

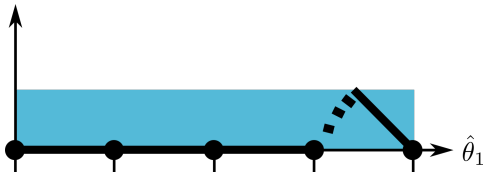
Therefore, the lemma holds. □

### Covering via a uniform grid

The greedy approach in Section 10.4.1 is extremely versatile: no matter the type space $\Theta_i$, when we use the resulting cover to estimate the incentive compatibility approximation factor (via Equation (10.4)), the estimate quickly converges to the true approximation factor. However, implementing the greedy procedure (Algorithm 7) might be computationally challenging because at each round, it is necessary to check if $U \setminus (\bigcup_{v \in V} B_1(v, \epsilon))$ 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 $\mathcal{G}$: 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 10.4.3. 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* $\mathcal{G}_w$ over $[0,1]^m$, by which we mean a finite set of vectors in $[0,1]^m$ such that for all $p \in [0,1]^m$, there exists a vector $p' \in \mathcal{G}_w$ such that $\|p - p'\|_1 \leq w$. For example, if $m = 1$, we could define $\mathcal{G}_w = \left\{0, \frac{1}{\lfloor 1/w \rfloor}, \frac{2}{\lfloor 1/w \rfloor}, \ldots, 1\right\}$. We will estimate the expected incentive compatibility approximation factor using Equation (10.4) with $\mathcal{G} = \mathcal{G}_w \times \mathcal{G}_w$. 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.




(a) Average of dispersed functions.




(b) Average of functions that are not dispersed.

Figure 10.1: Illustration of Example 10.4.6. The lines in Figure 10.1a 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 10.1a 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 10.1b, we illustrate the same concepts but for a different set of utility functions that are not as dispersed as those in Figure 10.1a. As illustrated by the blue regions, the approximation over the grid is better for the dispersed functions than the non-dispersed functions.

**Definition 10.4.5** (Balcan et al. [2018b])**.** 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 $\ell_1$-norm over a partition $\mathcal{P}_i$ of $\mathbb{R}^d$. We say that $\mathcal{P}_i$ splits 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 10.4.6.

**Example 10.4.6.** 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 10.1a 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 10.1b 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 10.1, we evaluate each function on the grid $\mathcal{G} = \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\}$. In Figure 10.1a, the maximum over $\mathcal{G}$ better approximates the the maximum over $[0,1]$ compared to Figure 10.1b. In other words,

$$\max_{\hat{\theta}_1 \in [0,1]} \sum_{\theta_2 \in \mathcal{S}_2} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right) - \max_{\hat{\theta}_1 \in \mathcal{G}} \sum_{\theta_2 \in \mathcal{S}_2} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right) \tag{10.6}$$

$$< \max_{\hat{\theta}_1 \in [0,1]} \sum_{\theta_2 \in \mathcal{S}_2'} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right) - \max_{\hat{\theta}_1 \in \mathcal{G}} \sum_{\theta_2 \in \mathcal{S}_2'} u_{1,M}\left(1, \hat{\theta}_1, \theta_2\right). \tag{10.7}$$

Intuitively, this is because the functions we average over in Figure 10.1a are more dispersed than the functions we average over in Figure 10.1b. The differences described by Equations (10.6) and (10.7) are represented by the shaded regions in Figures 10.1a and 10.1b, 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 10.4.7.** *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 $i \in [n]$,*

$$\begin{aligned}
&\max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in [0,1]^m} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} \\
&\leq \max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in \mathcal{G}_{w_i}} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + 4L_i w_i + \frac{8k_i}{N}. \qquad (10.8)
\end{aligned}$$

*Proof.* By definition of dispersion, the following conditions hold with probability $1-\delta$:

**Condition 1.** For all mechanisms $M \in \mathcal{M}$, agents $i \in [n]$, types $\boldsymbol{\theta}_i \in [0,1]^m$, and pairs of reported types $\hat{\boldsymbol{\theta}}_i, \hat{\boldsymbol{\theta}}_i' \in [0,1]^m$, if $\left\|\hat{\boldsymbol{\theta}}_i - \hat{\boldsymbol{\theta}}_i'\right\|_1 \leq w_i$, then

$$\left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i', \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \leq L_i w_i + \frac{2k_i}{N}. \qquad (10.9)$$

**Condition 2.** For all mechanisms $M \in \mathcal{M}$, agents $i \in [n]$, reported types $\hat{\boldsymbol{\theta}}_i \in [0,1]^m$, and pairs of types $\boldsymbol{\theta}_i, \boldsymbol{\theta}_i' \in [0,1]^m$, if $\left\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_i'\right\|_1 \leq w_i$, then

$$\left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i', \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \leq L_i w_i + \frac{2k_i}{N}. \qquad (10.10)$$

We claim that Inequality (10.8) holds so long as Conditions 1 and 2 hold. To see why, suppose they do both hold, and fix an arbitrary agent $i \in [n]$, mechanism $M \in \mathcal{M}$, and pair of types $\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in [0,1]^m$. Consider the average amount agent $i$ with type $\boldsymbol{\theta}_i$ can improve his utility by misreporting his type as $\hat{\boldsymbol{\theta}}_i$:

$$\frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right). \qquad (10.11)$$

By definition of the grid $\mathcal{G}_{w_i}$, we know there are points $\boldsymbol{p}, \hat{\boldsymbol{p}} \in \mathcal{G}_{w_i}$ such that $\left\|\boldsymbol{\theta}_i - \boldsymbol{p}\right\|_1 \leq w_i$ and $\left\|\hat{\boldsymbol{\theta}}_i - \hat{\boldsymbol{p}}\right\|_1 \leq w_i$. Based on Conditions 1 and 2, in the following claim, we show that if we "snap" $\boldsymbol{\theta}_i$ to $\boldsymbol{p}$ and $\hat{\boldsymbol{\theta}}_i$ to $\hat{\boldsymbol{p}}$, we will not increase Equation (10.11) by more than an additive factor of $O\left(L_i w_i + k_i/N\right)$.

**Claim 10.4.8.** *If Conditions 1 and 2 hold, then for all $M \in \mathcal{M}$ and $i \in [n]$,*

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

*Proof of Claim 10.4.8.* Fix an agent $i \in [n]$ and let $\boldsymbol{\theta}_i$ and $\hat{\boldsymbol{\theta}}_i$ be two fixed, arbitrary vectors in $[0,1]^m$. By definition of $\mathcal{G}_w$, there must be a point $\boldsymbol{p} \in \mathcal{G}_w$ such that $\|\boldsymbol{\theta}_i - \boldsymbol{p}\|_1 \leq w_i$. By Equation (10.9),

$$\left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \leq L_i w_i + \frac{2k_i}{N}$$

and by Equation (10.10),

$$\left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{p}, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \leq L_i w_i + \frac{2k_i}{N}.$$

Similarly, there must be a point $\hat{\boldsymbol{p}} \in \mathcal{G}_w$ such that $\left\|\hat{\boldsymbol{p}} - \hat{\boldsymbol{\theta}}_i\right\|_1 \leq w_i$. By Equation (10.9),

$$\left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \leq L_i w_i + \frac{2k_i}{N}$$

and by Equation (10.10),

$$\left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{p}, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \leq L_i w_i + \frac{2k_i}{N}.$$

Therefore,

$$\begin{aligned}
& \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \\
\leq & \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \\
+ & \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{p}, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \\
+ & \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{p}, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{p}, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \\
+ & \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{p}, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| \\
+ & \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right|
\end{aligned}$$

$$\leq \left| \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{p}, \boldsymbol{p}, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{p}, \hat{\boldsymbol{p}}, \boldsymbol{\theta}_{-i}^{(j)}\right) \right| + 4L_i w_i + \frac{8k_i}{N}.$$

Since $\boldsymbol{p}, \hat{\boldsymbol{p}} \in \mathcal{G}_{w_i}$, the claim holds. □

Since $\boldsymbol{p}$ and $\hat{\boldsymbol{p}}$ are elements of $\mathcal{G}_{w_i}$, Inequality (10.8) holds so long as Conditions 1 and 2 hold. Since Conditions 1 and 2 hold with probability $1 - \delta$, the lemma statement holds. □

In Section 10.4.3, we prove that under mild assumptions, for a wide range of mechanisms, Conditions 1 and 2 in Lemma 10.4.7 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 10.4.1 and Lemma 10.4.7 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 10.4.9.** *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 $i \in [n]$,*

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

*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 $d_i = \operatorname{Pdim}\left(\mathcal{F}_{i,\mathcal{M}}\right)$.*

*Proof.* Theorem 10.4.1 guarantees that with probability at most $\delta$, the extent to which any agent $i$ can improve his utility by misreporting his type, averaged over all profiles in $\mathcal{S}_{-i}$, does not approximate the true incentive compatibility approximation factor, as summarized below:

**Bad event 1.** For some mechanism $M \in \mathcal{M}$ and agent $i \in [n]$,

$$\begin{aligned} &\max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in \Theta_i} \left\{ \mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i} \sim \mathcal{D}_{-i}} \left[ u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}\right) \right] \right\} \\ > &\max_{\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i \in [0,1]^m} \left\{ \frac{1}{N} \sum_{j=1}^{N} u_{i,M}\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) - u_{i,M}\left(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i, \boldsymbol{\theta}_{-i}^{(j)}\right) \right\} + \epsilon_{\mathcal{M}}(N, \delta), \end{aligned} \tag{10.12}$$

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 $d_i = \text{Pdim}\left(\mathcal{F}_{i,\mathcal{M}}\right)$.

By Lemma 10.4.7, we also know that with probability at most $\delta$, we cannot approximate Equation (10.12) by discretizing the agent's type space, as summarized by the following bad event:

**Bad event 2.** For some mechanism $M \in \mathcal{M}$ and agent $i \in [n]$,

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

By a union bound, the probability that either Bad Event 1 or Bad Event 2 occurs is at most $2\delta$. Therefore, the probability that neither occurs is at least $1-2\delta$. If neither occurs, then for every mechanism $M \in \mathcal{M}$ and every agent $i \in [n]$,

$$\begin{aligned}&\max_{\boldsymbol{\theta}_i,\hat{\boldsymbol{\theta}}_i\in[0,1]^m}\left\{\mathop{\mathbb{E}}_{\boldsymbol{\theta}_{-i}\sim\mathcal{D}_{-i}}\left[u_{i,M}\left(\boldsymbol{\theta}_i,\hat{\boldsymbol{\theta}}_i,\boldsymbol{\theta}_{-i}\right)-u_{i,M}\left(\boldsymbol{\theta}_i,\boldsymbol{\theta}_i,\boldsymbol{\theta}_{-i}\right)\right]\right\}\\ \le&\max_{\boldsymbol{\theta}_i,\hat{\boldsymbol{\theta}}_i\in\mathcal{G}_{w_i}}\left\{\frac{1}{N}\sum_{j=1}^{N}u_{i,M}\left(\boldsymbol{\theta}_i,\hat{\boldsymbol{\theta}}_i,\boldsymbol{\theta}_{-i}^{(j)}\right)-u_{i,M}\left(\boldsymbol{\theta}_i,\boldsymbol{\theta}_i,\boldsymbol{\theta}_{-i}^{(j)}\right)\right\}+\epsilon,\end{aligned}$$

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

We conclude with one final comparison of the greedy approach in Section 10.4.1 and the uniform grid approach in this section. In Section 10.4.1, 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 Lipschitz continuity.

### 10.4.2 Delineable utility functions

To prove some of our pseudo-dimension guarantees, we use the notion of *delineability* introduced in Section 5.1. There, we used it in the context of profit-maximization, so here we adapt it to our setting.

Given a mechanism $M$ and agent $i \in [n]$, let $\mathcal{F}_{i,M} = \left\{u_{i,M,\theta_i,\hat{\theta}_i} \,\middle|\, \theta_i,\hat{\theta}_i \in \Theta_i\right\}$, where $u_{i,M,\theta_i,\hat{\theta}_i}$ is the function introduced in Section 10.4. Moreover, for a fixed type profile $\boldsymbol{\theta}_{-i} \in \times_{j\neq i}\Theta_j$, let $u_{\boldsymbol{\theta}_{-i}} : \Theta_i^2 \to [-1,1]$ be a function that maps a pair of types $\theta_i,\hat{\theta}_i \in \Theta_i$ to $u_{i,M,\theta_i,\hat{\theta}_i}\left(\boldsymbol{\theta}_{-i}\right)$.

**Definition 10.4.10** ($(m,t)$-delineable)**.** Given a mechanism $M$ and agent $i \in [n]$, we say the function class $\mathcal{F}_{i,M}$ is $(m,t)$*-delineable* if:

1. Each agent's type space is a subset of $[0,1]^m$; and
2. For any $\boldsymbol{\theta}_{-i} \in [0,1]^{m(n-1)}$, there is a set $\mathcal{H}$ of $t$ hyperplanes such that for any connected component $\mathcal{C}$ of $[0,1]^{2m} \setminus \mathcal{H}$, the function $u_{\boldsymbol{\theta}_{-i}}\left(\boldsymbol{\theta}_i,\hat{\boldsymbol{\theta}}_i\right)$ is linear over $\mathcal{C}$.

The following theorem is a parallel of Theorem 5.1.4 from Section 5.1, though we have adapted it to our setting.

**Theorem 10.4.11.** *If $\mathcal{F}_{i,M}$ is $(m,t)$-delineable, the pseudo dimension of $\mathcal{F}_{i,M}$ is $O(m \log(mt))$.*

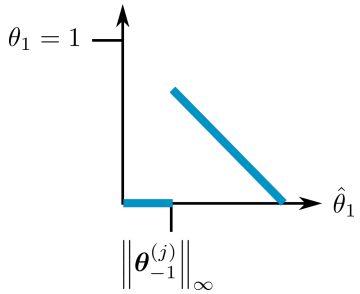
### 10.4.3 Dispersion and pseudo-dimension guarantees

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 10.1.

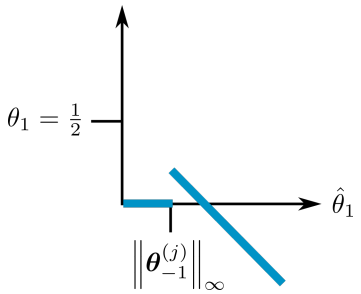
Given an agent $i \in [n]$, a true type $\boldsymbol{\theta}_i \in [0,1]^m$, and a set of samples $\boldsymbol{\theta}_{-i}^{(1)}, \ldots, \boldsymbol{\theta}_{-i}^{(N)} \sim \mathcal{D}_{-i}$, we often find that the discontinuities of each function $u_{i,M}\left(\boldsymbol{\theta}_i, \cdot, \boldsymbol{\theta}_{-i}^{(j)}\right)$ 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 *$\kappa$-bounded density functions*. A density function $\phi : \mathbb{R} \to \mathbb{R}$ is $\kappa$-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. [2018b]: 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 10.4.12). 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 10.4.13—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 10.4.15). As a result, all sequences from this second family are immediately dispersed.




(a) The function $u_{1,M}\left(1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$.




(b) The function $u_{1,M}\left(\frac{1}{2}, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$.

Figure 10.2: Agent 1's utility function under a first-price single-item auction when the other agents' values and bids are defined by $\boldsymbol{\theta}_{-1}^{(j)} \in [0,1]^{n-1}$. The utility is $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right) = \mathbf{1}_{\left\{\hat{\theta}_1 > \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty\right\}}\left(\theta_1 - \hat{\theta}_1\right)$.

**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 in Section 10.4.3. 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 10.4.12.** *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 $\left\{\left\|\boldsymbol{\theta}_{-i}^{(j)}\right\|_\infty\right\}_{j \in [N]}$.*

*Proof.* Suppose $i = 1$ and choose an arbitrary sample $j \in [N]$. For any value $\theta_1 \in [0,1]$ and bid $\hat{\theta}_1 \in [0,1]$, we know that $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right) = \mathbf{1}_{\left\{\hat{\theta}_1 > \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty\right\}}\left(\theta_1 - \hat{\theta}_1\right)$, as illustrated in Figure 10.2. Therefore, no matter the value of $\theta_1$, the function $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is piecewise 1-Lipschitz with a discontinuity at $\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$. □

We now use Lemma 10.4.12 to prove our first dispersion guarantee.

**Theorem 10.4.13.** *Suppose each agent's type has a $\kappa$-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.*

*Proof.* Consider an arbitrary bidder, and without loss of generality, suppose that bidder is bidder 1. Next, choose an arbitrary sample $\boldsymbol{\theta}_{-1}^{(j)} = \left(\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right)$. For any value $\theta_1 \in [0,1]$

and bid $\hat{\theta}_1 \in [0,1]$, we know that $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right) = \mathbf{1}_{\left\{\hat{\theta}_1 > \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty\right\}}\left(\theta_1 - \hat{\theta}_1\right)$, as illustrated in Figure 10.2. Therefore, if $\hat{\theta}_1 \leq \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$, then $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is a constant function of $\hat{\theta}_1$, whereas if $\hat{\theta}_1 > \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$, then $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is a linear function of $\hat{\theta}_1$ with a slope of $-1$. Therefore, for all $\theta_i \in [0,1]$, the function $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is piecewise 1-Lipschitz with a discontinuity at $\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$.

We now prove that with probability $1 - \frac{\delta}{n}$, for all $\theta_1 \in [0,1]$, the functions

$$u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(1)}\right), \ldots, u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(N)}\right)$$

are $(w,k)$-dispersed. Since for any $\theta_1 \in [0,1]$ the function $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ will only have a discontinuity at a point in the set $\left\{\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right\}$, it is enough to prove that with probability $1 - \frac{\delta}{n}$, at most $k$ points in the set $\mathcal{B} = \bigcup_{j=1}^{N}\left\{\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right\}$ fall within any interval of width $w$. The theorem statement then holds by a union bound over all $n$ bidders.

**Claim 10.4.14.** *With probability $1 - \frac{\delta}{n}$, at most $k$ points in the set $\bigcup_{j=1}^{N}\left\{\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right\}$ fall within any interval of width $w$.*

*Proof of Claim 10.4.14.* For $i \in \{2, \ldots, n\}$, let $\mathcal{B}_i = \left\{\theta_i^{(j)}\right\}_{j \in [N]}$. The claim follows from Lemma 10.5.1 in Section 10.5. □

□

**Theorem 10.4.15.** *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.*

*Proof.* Since the reported types $\left(\hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ are fixed, the allocation is fixed. Thus, $u_{i,M}\left(\cdot, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ is either a constant function if $\hat{\theta}_i \leq \|\boldsymbol{\theta}_{-i}\|_\infty$ or a linear function if $\hat{\theta}_i > \|\boldsymbol{\theta}_{-i}\|_\infty$. □

Next, we prove the following pseudo-dimension bound.

**Theorem 10.4.16.** *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is 2.*

*Proof.* First, we prove that $\text{Pdim}\left(\mathcal{F}_{i,M}\right) \leq 2$.

**Claim 10.4.17.** *The pseudo-dimension of $\mathcal{F}_{i,M}$ is at most 2.*

*Proof of Claim 10.4.17.* For a contradiction, suppose there exists a set $\mathcal{S}_{-i} = \left\{\boldsymbol{\theta}_{-i}^{(1)}, \boldsymbol{\theta}_{-i}^{(2)}, \boldsymbol{\theta}_{-i}^{(3)}\right\}$ that is shattered by $\mathcal{F}_{i,M}$. Without loss of generality, assume that $\left\|\boldsymbol{\theta}_{-i}^{(1)}\right\|_\infty < \left\|\boldsymbol{\theta}_{-i}^{(2)}\right\|_\infty < \left\|\boldsymbol{\theta}_{-i}^{(3)}\right\|_\infty$. Since $\mathcal{S}_{-i}$ is shatterable, there exists three values $z^{(1)}, z^{(2)}, z^{(3)} \in \mathbb{R}$ that witness the shattering of $\mathcal{S}_{-i}$ by $\mathcal{F}_{i,M}$. We split the proof into two cases: one where $z^{(3)} > 0$ and one where $z^{(3)} \leq 0$.

| $\theta_i$ | $\hat{\theta}_i$ | $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right)$ | $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(2)}\right)$ |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| | 7/8 | 1/8 | 1/8 |
| | 3/4 | 1/4 | 1/4 |
| | 1/2 | 1/2 | 0 |

Table 10.2: Shattering of the example from Claim 10.4.18.

**Case 1:** $z^{(3)} > 0$**.** Since $\mathcal{S}_{-i}$ is shatterable, there is a value $\theta_i \in [0,1]$ and bid $\hat{\theta}_i \in [0,1]$ such that $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \geq z^{(1)}$, $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(2)}\right) < z^{(2)}$, and $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(3)}\right) \geq z^{(3)}$. Recall that for any $\boldsymbol{\theta}_{-i} \in [0,1]^{n-1}$, $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) = \mathbf{1}_{\left\{\hat{\theta}_i > \|\boldsymbol{\theta}_{-i}\|_\infty\right\}}\left(\theta_i - \hat{\theta}_i\right)$. Since $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(3)}\right) \geq z^{(3)} > 0$, it must be that $\hat{\theta}_i > \left\|\boldsymbol{\theta}_{-i}^{(3)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(2)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(1)}\right\|_\infty$. Therefore, $\theta_i - \hat{\theta}_i \geq z^{(1)}$ and $\theta_i - \hat{\theta}_i < z^{(2)}$, which means that $z^{(1)} < z^{(2)}$.

Next, there must also be a value $\theta_i' \in [0,1]$ and bid $\hat{\theta}_i' \in [0,1]$ such that $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(1)}\right) < z^{(1)}$, $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(2)}\right) \geq z^{(2)}$, and $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(3)}\right) \geq z^{(3)}$. Again, since $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(3)}\right) \geq z^{(3)} > 0$, it must be that $\hat{\theta}_i' > \left\|\boldsymbol{\theta}_{-i}^{(3)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(2)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(1)}\right\|_\infty$. Therefore, $\theta_i' - \hat{\theta}_i' < z^{(1)}$ and $\theta_i' - \hat{\theta}_i' \geq z^{(2)}$, which means that $z^{(2)} > z^{(1)}$, which is a contradiction.

**Case 2:** $z^{(3)} \leq 0$**.** Since $\mathcal{S}_{-i}$ is shatterable, there is a value $\theta_i \in [0,1]$ and bid $\hat{\theta}_i \in [0,1]$ such that $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(1)}\right) \geq z^{(1)}$, $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(2)}\right) < z^{(2)}$, and $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(3)}\right) < z^{(3)}$. Since $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}^{(3)}\right) < z^{(3)} \leq 0$, it must be that $\hat{\theta}_i > \left\|\boldsymbol{\theta}_{-i}^{(3)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(2)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(1)}\right\|_\infty$. Therefore, $\theta_i - \hat{\theta}_i \geq z^{(1)}$ and $\theta_i - \hat{\theta}_i < z^{(2)}$, which means that $z^{(1)} < z^{(2)}$.

Next, there must also be a value $\theta_i' \in [0,1]$ and bid $\hat{\theta}_i' \in [0,1]$ such that $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(1)}\right) < z^{(1)}$, $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(2)}\right) \geq z^{(2)}$, and $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(3)}\right) < z^{(3)}$. Again, since $u_{i,M}\left(\theta_i', \hat{\theta}_i', \boldsymbol{\theta}_{-i}^{(3)}\right) < z^{(3)} \leq 0$, it must be that $\hat{\theta}_i' > \left\|\boldsymbol{\theta}_{-i}^{(3)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(2)}\right\|_\infty > \left\|\boldsymbol{\theta}_{-i}^{(1)}\right\|_\infty$. Therefore, $\theta_i' - \hat{\theta}_i' < z^{(1)}$ and $\theta_i' - \hat{\theta}_i' \geq z^{(2)}$, which means that $z^{(2)} > z^{(1)}$, which is a contradiction.

Since we arrive at a contradiction in both cases, we conclude that $\text{Pdim}\left(\mathcal{F}_{i,M}\right) \leq 2$. □

We now prove that $\text{Pdim}\left(\mathcal{F}_{i,M}\right) \geq 2$.

**Claim 10.4.18.** *The pseudo-dimension of $\mathcal{F}_{i,M}$ is at least 2.*

*Proof of Claim 10.4.18.* To prove this claim, we exhibit a set of size 2 that is shattered by $\mathcal{F}_{i,M}$. Let $\boldsymbol{\theta}_{-i}^{(1)}$ and $\boldsymbol{\theta}_{-i}^{(2)}$ be two sets of values such that $\left\|\boldsymbol{\theta}_{-i}^{(1)}\right\|_\infty = 1/3$ and $\left\|\boldsymbol{\theta}_{-i}^{(2)}\right\|_\infty = 2/3$. Table 10.2 illustrates that $z^{(1)} = 3/16$ and $z^{(2)} = 1/16$ witness the shattering of $\left\{\boldsymbol{\theta}_{-i}^{(1)}, \boldsymbol{\theta}_{-i}^{(2)}\right\}$ by $\mathcal{F}_{i,M}$. □

Together, Claims 10.4.17 and 10.4.18 prove that $\text{Pdim}\left(\mathcal{F}_{i,M}\right) = 2$. □

**First-price combinatorial auction**

Under this auction, there are $\ell$ 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 10.4.19.** *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 $\kappa$-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.*

*Proof.* Fix an arbitrary agent, and without loss of generality, suppose that agent is agent 1. Next, fix an arbitrary sample $\boldsymbol{\theta}_{-1}^{(j)}$ and pair of allocations $(b_1, \ldots, b_n)$ and $(b_1', \ldots, b_n')$. We know that $(b_1', \ldots, b_n')$ will *not* be the allocation of the first-price combinatorial auction so long as agent 1's reported type $\hat{\boldsymbol{\theta}}_1 \in [0,1]^{2^\ell}$ is chosen such that

$$\hat{\theta}_1(b_1) + \sum_{i=2}^n \theta_i^{(j)}(b_i) > \hat{\theta}_1(b_1') + \sum_{i=2}^n \theta_i^{(j)}(b_i').$$

This means that across all $\boldsymbol{\theta}_1 \in [0,1]^{2^\ell}$, there is fixed a set $\Psi_j$ of $\binom{(n+1)^\ell}{2}$ hyperplanes (one per pair of allocations) such that for any connected component $C$ of $[0,1]^{2^\ell} \setminus \Psi_j$, the allocation of the first-price combinatorial auction is invariant across all $\hat{\boldsymbol{\theta}}_1 \in C$. Namely,

$$\Psi_j = \left\{\hat{\theta}_1(b_1) - \hat{\theta}_1(b_1') = \sum_{i=2}^n \theta_i^{(j)}(b_i') - \theta_i^{(j)}(b_i) : (b_1, \ldots, b_n) \text{ and } \left(b_1', \ldots, b_n'\right) \text{ are allocations}\right\}.$$

So long as the allocation is fixed, agent 1's utility is 1-Lipschitz in $\hat{\boldsymbol{\theta}}_1$.

For each pair of allocations $\boldsymbol{b} = (b_1, \ldots, b_n)$ and $\boldsymbol{b}' = (b_1', \ldots, b_n')$, let

$$\mathcal{B}_{\boldsymbol{b},\boldsymbol{b}'} = \left\{\hat{\theta}_1(b_1) - \hat{\theta}_1(b_1') = \sum_{i=2}^n \theta_i^{(j)}(b_i') - \theta_i^{(j)}(b_i) : j \in [N]\right\}.$$

Within each set, the hyperplanes are parallel with probability 1. By Lemmas 10.5.3 and 10.5.4 in Section 10.5, their offsets are independently drawn from distributions with $\kappa$-bounded density functions. Therefore, by Lemma 10.5.6 in Section 10.5, with probability $1 - \frac{\delta}{n}$, for all $\boldsymbol{\theta}_1 \in [0,1]^{2^\ell}$, the functions $u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(1)}\right), \ldots, u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(N)}\right)$ are $(w,k)$-dispersed with $w = O\left(\frac{1}{\kappa\sqrt{N}}\right)$ and $k = O\left((n+1)^{2\ell}\sqrt{N \log \frac{n(n+1)^{2\ell}}{\delta}}\right)$. The theorem statement holds by a union bound over all $n$ agents. □

**Theorem 10.4.20.** *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.*

*Proof.* So long as all bids are fixed, the allocation is fixed, so $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz. □

Next, we prove the following pseudo-dimension bound.

**Theorem 10.4.21.** *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is $O\left(\ell 2^\ell \log n\right)$.*

*Proof.* As we saw in the proof of Theorem 10.4.19, for any type profile $\boldsymbol{\theta}_{-i} \in [0,1]^{(n-1)2^\ell}$, there is a set $\mathcal{H}$ of $(n+1)^{2^\ell}$ hyperplanes such that for any connected component $C$ of $[0,1]^{2^\ell} \setminus \mathcal{H}$, the auction's allocation given bids $\left(\hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is invariant across all $\hat{\boldsymbol{\theta}}_i \in C$. So long as the allocation is fixed, $u_{i,M}\left(\cdot, \cdot, \boldsymbol{\theta}_{-i}\right)$ is a linear function of $\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i\right)$. Therefore, the pseudo-dimension bound follows from Theorem 10.4.11, which relates the class's pseudo-dimension to the number of hyperplanes splitting the type space into regions where the utility function is linear. □

### 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 10.4.22.** *Suppose each agent's type has a $\kappa$-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.*

*Proof.* Let $w = O\left(\frac{1}{r\kappa\sqrt{N}}\right)$ and $k = O\left(n^{3/2}\sqrt{N \log \frac{nr}{\delta}}\right)$. Consider an arbitrary agent, and without loss of generality, suppose that agent is agent 1. Next, choose an arbitrary sample $\boldsymbol{\theta}_{-1}^{(j)} = \left(\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right)$ and mechanism $M \in \mathcal{M}_r$ with agent weights $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n) \in \left\{\frac{1}{r}, \frac{2}{r}, \ldots, 1\right\}^n$. Let $\omega_{i_1}\theta_{i_1}^{(j)} \leq \cdots \leq \omega_{i_{n-1}}\theta_{i_{n-1}}^{(j)}$ be the weighted types of all agents except agent 1. Consider the $n$ intervals delineated by these $n-1$ weighted values: $I_1 = \left[0, \omega_{i_1}\theta_{i_1}^{(j)}\right), \ldots, I_n = \left[\omega_{i_{n-1}}\theta_{i_{n-1}}^{(j)}, 1\right]$ with

$$I_\tau = \left[\omega_{i_{\tau-1}}\theta_{i_{\tau-1}}^{(j)}, \omega_{i_\tau}\theta_{i_\tau}^{(j)}\right).$$

Suppose we vary $\hat{\theta}_1$ in such a way that $\omega_1\hat{\theta}_1$ remains within a single interval $I_\tau$. Said another way, consider varying $\hat{\theta}_1$ over the interval

$$\left[\frac{\omega_{i_{\tau-1}}\theta^{(j)}_{i_{\tau-1}}}{\omega_1}, \frac{\omega_{i_\tau}\theta^{(j)}_{i_\tau}}{\omega_1}\right).$$

The allocation will be constant, which means agent 1's utility will be a constant function of $\hat{\theta}_1$. Therefore, no matter the value of $\theta_1$, $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}^{(1)}_{-1}\right), \ldots, u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}^{(N)}_{-1}\right)$ are piecewise constant with discontinuities in the set

$$\mathcal{B}_{1,\boldsymbol{\omega}} = \left\{\frac{\omega_{i'}\theta^{(j)}_{i'}}{\omega_1} : i' \in \{2, \ldots, n\}, j \in [N]\right\}.$$

This means that in order to prove the theorem, it is enough to show that with probability $1-\delta$, for all $\boldsymbol{\omega} \in \left\{\frac{1}{r}, \frac{2}{r}, \ldots, 1\right\}^n$ and all agents $i \in [n]$, at most $k$ of the values in each set

$$\mathcal{B}_{i,\boldsymbol{\omega}} = \left\{\frac{\omega_{i'}\theta^{(j)}_{i'}}{\omega_i} : i' \in [n] \setminus \{i\}, j \in [N]\right\}$$

fall within any interval of length $w$. This follows from the following claim and a union bound.

**Claim 10.4.23.** *For a fixed $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_n) \in \left\{\frac{1}{r}, \frac{2}{r}, \ldots, 1\right\}^n$ and agent $i \in [n]$, with probability $1 - \frac{\delta}{nr^n}$, at most $k$ points from $\mathcal{B}_{i,\boldsymbol{\omega}}$ fall within any interval of width $w$.*

*Proof of Claim 10.4.23.* For each $i' \in [n] \setminus \{i\}$, let

$$\mathcal{B}_{i,i',\boldsymbol{\omega}} = \left\{\frac{\omega_{i'}\theta^{(j)}_{i'}}{\omega_i} : j \in [N]\right\}.$$

Since $\frac{\omega_{i'}}{\omega_i} \geq \frac{1}{r}$, Lemma 10.5.2 in Section 10.5 implies that the points in $\mathcal{B}_{i,i',\boldsymbol{\omega}}$ are independent draws from a $r\kappa$-bounded distribution. The claim then follows from Lemma 10.5.1 in Section 10.5. □

□

**Theorem 10.4.24.** *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.*

*Proof.* Since the reported types $\left(\hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ are fixed, the allocation is fixed. Let $\pi(s)$ be the agent who is allocated slot $s$. The function $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ is either a constant function of $\theta_i$ if agent $i$ is not allocated a slot or a linear function of $\theta_i$ with a slope of $\alpha_{\pi^{-1}(i),i}$ otherwise. □

We will use the following lemma in this section's pseudo-dimension bound.

**Lemma 10.4.25** (Anthony and Bartlett [2009])**.** *Suppose $f_1, \ldots, f_p$ are polynomials of degree at most $d$ in $v \leq p$ variables. Then*

$$\left|\left\{\begin{pmatrix}\operatorname{sign}\left(f_1(\boldsymbol{x})\right)\\ \vdots \\ \operatorname{sign}\left(f_p(\boldsymbol{x})\right)\end{pmatrix} : \boldsymbol{x} \in \mathbb{R}^v\right\}\right| \leq 2\left(\frac{2epd}{v}\right)^v.$$

We now provide the following pseudo-dimension guarantee.

**Theorem 10.4.26.** *For any agent $i \in [n]$ and $r \in \mathbb{Z}_{\geq 1}$, $\operatorname{Pdim}\left(\mathcal{F}_{i,\mathcal{M}_r}\right) = O\left(n \log n\right)$.*

*Proof.* Fix an arbitrary agent, and without loss of generality, suppose that agent is agent 1. Fix a vector $\boldsymbol{\theta}_{-1} = (\theta_2, \ldots, \theta_n) \in [0,1]^{n-1}$. We denote the utility of agent 1 when the agents' types are $(\theta_1, \boldsymbol{\theta}_{-1})$ and reported types are $\left(\hat{\theta}_1, \boldsymbol{\theta}_{-1}\right)$ as $u_{\boldsymbol{\theta}_{-1}}\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right)$, which maps $\mathbb{R}^{n+2}$ to $[-1,1]$.

We now show that we can split $\mathbb{R}^{n+2}$ into regions $R$ where the allocation of the generalized second-price auction is fixed across all $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right) \in R$, given the fixed set of reported types $\boldsymbol{\theta}_{-1} = (\theta_2, \ldots, \theta_n)$. Let $\bar{\pi}$ be a permutation of the $n$ agents, and let $R_{\bar{\pi}} \subseteq [0,1]^{n+2}$ be the set of all vectors $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right)$ where the ordering of the elements $\left\{\omega_1\hat{\theta}_1, \omega_2\theta_2, \ldots, \omega_n\theta_n\right\}$ matches $\bar{\pi}$. Specifically, for all $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right) \in R_{\bar{\pi}}$, if $i' = \operatorname{argmax}\left\{\omega_1\hat{\theta}_1, \omega_2\theta_2, \ldots, \omega_n\theta_n\right\}$, then $\bar{\pi}(1) = i'$ and if $i'' = \operatorname{argmin}\left\{\omega_1\hat{\theta}_1, \omega_2\theta_2, \ldots, \omega_n\theta_n\right\}$, then $\bar{\pi}(n) = i''$. By definition of the generalized second-price auction, this means that the agent who receives the first slot is agent $\bar{\pi}(1)$ and the agent who receives the $m^{th}$ is agent $\bar{\pi}(m)$. Therefore, for all $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right) \in R_{\bar{\pi}}$, the allocation of the generalized second-price auction given agent weights $(\omega_1, \ldots, \omega_n)$ and reported types $\left(\hat{\theta}_1, \boldsymbol{\theta}_{-1}\right)$ is invariant. Next, consider the transformation $\phi : \left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right) \mapsto \left(\theta_1, \frac{\omega_2}{\omega_1}, \ldots, \frac{\omega_n}{\omega_1}\right)$. Since the vector $\boldsymbol{\theta}_{-1}$ is fixed, the function $u_{\boldsymbol{\theta}_{-1}}\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right)$ is a fixed linear function of $\phi\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right)$ across all $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right) \in R_{\bar{\pi}}$.

Next, let $\mathcal{S}_{-1} = \left\{\boldsymbol{\theta}_{-1}^{(1)}, \ldots, \boldsymbol{\theta}_{-1}^{(N)}\right\}$ be a set of vectors that is shatterable by the set of functions $\mathcal{F}_{1,\mathcal{M}_r}$, where $\boldsymbol{\theta}_{-1}^{(j)} = \left(\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right)$. This means there is a set of values $z^{(1)}, \ldots, z^{(N)} \in \mathbb{R}$ that witnesses the shattering of $\mathcal{S}_{-1}$ by $\mathcal{F}_{1,\mathcal{M}_r}$. Therefore, for every $T \subseteq [N]$, there exists a pair of values $\theta_T, \hat{\theta}_T$ and a mechanism $M_T \in \mathcal{M}_r$ such that $u_{1,M_T,\theta_T,\hat{\theta}_T}\left(\boldsymbol{\theta}_{-1}^{(j)}\right) \leq z^{(j)}$ if and only if $j \in T$. Let $(\omega_{1,T}, \ldots, \omega_{n,T})$ be the set of agent weights corresponding to each such mechanism $M_T$. We define the set $\mathcal{R}$ to be the set of $2^N$ vectors $\mathcal{R} = \left\{\left(\theta_T, \hat{\theta}_T, \omega_{1,T}, \ldots, \omega_{n,T}\right) : T \subseteq [N]\right\} \subset \mathbb{R}^{n+2}$.

For each $j \in [N]$, let $\mathcal{H}^{(j)}$ be the set of $\binom{n}{2} + n - 1$ hypersurfaces

$$\mathcal{H}^{(j)} = \left\{\omega_1\hat{\theta}_1 - \omega_i\theta_i^{(j)} = 0 : i \in \{2, \ldots, n\}\right\} \cup \left\{\omega_i\theta_i - \omega_{i'}\theta_{i'}^{(j)} = 0 : i, i' \in \{2, \ldots, n\}\right\}.$$

As we saw, as we range $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right)$ over a single connected component of $\mathbb{R}^{n+2} \setminus \mathcal{H}^{(j)}$, the allocation of the generalized second-price auction with agent weights $(\omega_1, \ldots, \omega_n)$ and reported types $\left(\hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is invariant. If we define $\mathcal{H} = \bigcup_{j=1}^{N} \mathcal{H}^{(j)}$, then as we range $\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right)$ over a single connected component of $\mathbb{R}^{n+2} \setminus \mathcal{H}$, the allocations are fixed across all $j \in [N]$. By Lemma 10.4.25, the number of connected components is at most

$$2\left(\frac{4eN\left(\binom{n}{2} + n - 1\right)}{n+2}\right)^{n+2} = 2\left(2eN(n-1)\right)^{n+2}.$$

Next, fix a connected component $C$ of $\mathbb{R}^{n+2} \setminus \mathcal{H}$. We know that for each $j \in [N]$, there is a constant $\alpha_j \in [0,1]$ and a bidder $i_j \neq 1$ such that $u_{\boldsymbol{\theta}_{-1}^{(j)}}\left(\theta_1, \hat{\theta}_1, \omega_1, \ldots, \omega_n\right) = \alpha_j\left(\theta_1 - \frac{\omega_{i_j}\theta_{i_j}}{\omega_1}\right)$ (where $\alpha_j = 0$ if agent $i$ does not receive any slot). Consider the $N$ hypersurfaces $\mathcal{H}_C = \left\{\alpha_j\left(\theta_1 - \frac{\omega_{i_j}t_{i_j}^{(j)}}{\omega_1}\right) = z^{(j)} : j \in [N]\right\}$. We know that at most one vector in $\mathcal{R}$ can come from each

connected component of $C \setminus \mathcal{H}_C$, of which there are at most $2\left(\frac{4eN}{n}\right)^n$. In total, this means that $2^N = |\mathcal{R}| \le 4\left(2eN(n-1)\right)^{n+2}\left(\frac{4eN}{n}\right)^n$, so by Lemma 2.1.4, $N = O(n \log n)$. □

**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 $\theta_i[1] \ge \theta_i[2] \ge \cdots \ge \theta_i[m]$. 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.

**Theorem 10.4.27.** *Suppose that each agent's value for each marginal unit has a $\kappa$-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.*

*Proof.* Fix an arbitrary agent, and without loss of generality, suppose that agent is agent 1. Let $\theta'_1 \le \theta'_2 \le \cdots \le \theta'_{Nm(n-1)}$ be the sorted values of $\left\{\theta_i^{(j)}[\mu] : i \in \{2, \ldots, n\}, j \in [N], \mu \in [m]\right\}$, and define $\theta'_0 = 0$ and $\theta'_{Nm(n-1)+1} = 1$. So long as agent 1's bids fall between these sorted bids, the allocation will be fixed across all samples. More formally, for each $\mu \in [m]$, so long as $\theta'_i < \hat{\theta}_1[\mu] < \theta'_{i+1}$ for some $i \in \{0, \ldots, Nm(n-1)\}$, the resulting allocation will be fixed across all $N$ samples. Now, for a given sample $\boldsymbol{\theta}_{-1}^{(j)}$, consider a region $R \subseteq [0,1]^m$ where the allocation of the discriminatory auction given bids $\left(\hat{\boldsymbol{\theta}}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is invariant across all $\hat{\boldsymbol{\theta}}_1 \in R$. In particular, let $m_R$ be the number of units allocated to agent 1. For a fixed $\boldsymbol{\theta}_1 \in [0,1]^m$, agent 1's utility will be linear in $\hat{\boldsymbol{\theta}}_1$, because across all $\hat{\boldsymbol{\theta}}_1 \in R$, $u_{i,M}\left(\boldsymbol{\theta}_1, \hat{\boldsymbol{\theta}}_1, \boldsymbol{\theta}_{-1}^{(j)}\right) = \sum_{\mu=1}^{m_R} \theta_1[\mu] - \hat{\theta}_1[\mu]$. Therefore, the functions $u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(1)}\right), \ldots, u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(N)}\right)$ are piecewise 1-Lipschitz.

Note that across all $\boldsymbol{\theta}_1 \in [0,1]^m$, the partition $\mathcal{P}_j$ splitting $u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ into Lipschitz portions is invariant. In particular, no matter the value of $\boldsymbol{\theta}_1$, the partition $\mathcal{P}_j$ is delineated by the set of $m^2(n-1)$ hyperplanes $\mathcal{H}_j = \left\{\theta_i^{(j)}[\mu] - \hat{\theta}_1[\mu'] = 0 : i \in \{2, \ldots, n\}, \mu, \mu' \in [m]\right\}$. For each $i \in \{2, \ldots, n\}$ and pair $\mu, \mu' \in [m]$, let $\mathcal{B}_{i,\mu,\mu'}$ be the set of hyperplanes

$$\mathcal{B}_{i,\mu,\mu'} = \left\{\theta_i^{(j)}[\mu] - \hat{\theta}_1[\mu'] = 0 : j \in [N]\right\}.$$

Within each set, the hyperplanes are parallel with probability 1 and their offsets are independently drawn from $\kappa$-bounded distributions. Therefore, by Lemma 10.5.6 in Section 10.5, with probability $1 - \frac{\delta}{n}$, for all $\boldsymbol{\theta}_1 \in [0,1]^m$, the functions $u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(1)}\right), \ldots, u_{1,M}\left(\boldsymbol{\theta}_1, \cdot, \boldsymbol{\theta}_{-1}^{(N)}\right)$ are $\left(O\left(\frac{1}{\kappa\sqrt{N}}\right), \tilde{O}\left(nm^2\sqrt{N}\right)\right)$-dispersed. The theorem holds by a union bound over the agents. □

**Theorem 10.4.28.** *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.*

*Proof.* So long as all bids are fixed, the allocation is fixed, so $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz. □

Next, we prove the following pseudo-dimension bound.

**Theorem 10.4.29.** *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is $O\left(m \log(nm)\right)$.*

*Proof.* As we saw in the proof of Theorem 10.4.27, for any $\boldsymbol{\theta}_{-i} \in [0,1]^{(n-1)m}$, there is a set $\mathcal{H}$ of $O\left(m^2 n\right)$ hyperplanes such that for any connected component $C$ of $[0,1]^m \setminus \mathcal{H}$, the allocation is fixed across all $\hat{\boldsymbol{\theta}}_i \in C$. So long as the allocation is fixed, $u_{i,M}\left(\cdot, \cdot, \boldsymbol{\theta}_{-i}\right)$ is a linear function of $\left(\boldsymbol{\theta}_i, \hat{\boldsymbol{\theta}}_i\right)$. Therefore, the pseudo-dimension bound follows directly from Theorem 10.4.11. □

**Uniform-price auction**

Under the uniform-price auction, the allocation rule is the same as in the discriminatory auction (Section 10.4.3). However, all $m$ units are sold at a "market-clearing" price, meaning that the total amount demanded is equal to the total amount supplied. We adopt the convention [Krishna, 2002] that the market-clearing price equals the highest losing bid. Let $\boldsymbol{c}_{-i} \in \mathbb{R}^m$ be the vector $\hat{\boldsymbol{\theta}}_{-i}$ of competing bids facing agent $i$, sorted in decreasing order and limited to the top $m$ bids. This means that $c_{-i}[1] = \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty$ is the highest of the other bids, $c_{-i}[2]$ is the second-highest, and so on. Agent $i$ will win exactly one unit if and only if $\hat{\theta}_i[1] > c_{-i}[m]$ and $\hat{\theta}_i[2] < c_{-i}[m-1]$; that is, his bid must be higher than the lowest competing bid but not the second-lowest. Similarly, agent $i$ will win exactly two units if and only if $\hat{\theta}_i[2] > c_{-i}[m-1]$ and $\hat{\theta}_i[3] < c_{-i}[m-2]$. More generally, agent $i$ will win exactly $m_i > 0$ units if and only if $\hat{\theta}_i\left[m_i\right] > c_{-i}\left[m - m_i + 1\right]$ and $\hat{\theta}_i\left[m_i + 1\right] < c_{-i}\left[m - m_i\right]$. The market-clearing price, which equals the highest losing bid, is $p = \max\left\{\hat{\theta}_i\left[m_i + 1\right], c_{-i}\left[m - m_i + 1\right]\right\}$. In a uniform-price auction, agent $i$ pays $m_i p$.

**Theorem 10.4.30.** *Suppose that each agent's value for each marginal unit has a $\kappa$-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 $\boldsymbol{\theta}_i \in [0,1]^m$, 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 $\left(O\left(\frac{1}{\kappa\sqrt{N}}\right), O\left(nm^2\sqrt{N \log \frac{n}{\delta}}\right)\right)$-dispersed.*

*Proof.* This theorem follows by the exact same reasoning as Theorem 10.4.27. □

**Theorem 10.4.31.** *For all agents $i \in [n]$, all $\hat{\boldsymbol{\theta}}_i \in [0,1]^m$, and all $\boldsymbol{\theta}_{-i} \in [0,1]^{(n-1)m}$, the function $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz.*

*Proof.* So long as all bids are fixed, the allocation is fixed, so $u_{i,M}\left(\cdot, \hat{\boldsymbol{\theta}}_i, \boldsymbol{\theta}_{-i}\right)$ is 1-Lipschitz function as a function of the true type $\boldsymbol{\theta}_i \in [0,1]^m$. □

Next, we prove the following pseudo-dimension bound.

**Theorem 10.4.32.** *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is $O\left(m \log(nm)\right)$.*

*Proof.* This theorem follows by the exact same reasoning as Theorem 10.4.29. □

**Second-price auction with spiteful agents**

We conclude by studying *spiteful agents* [Brandt et al., 2007, Morgan et al., 2003, Sharma and Sandholm, 2010, Tang and Sandholm, 2012b], 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

$$u_{i,M}\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}\right) = \alpha_i \mathbf{1}_{\left\{\hat{\theta}_i > \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right\}} \left(\theta_i - \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right) - (1-\alpha_i) \sum_{i' \neq i} \mathbf{1}_{\left\{\hat{\theta}_{i'} > \left\|\hat{\boldsymbol{\theta}}_{-i'}\right\|_\infty\right\}} \left(\theta_{i'} - \left\|\hat{\boldsymbol{\theta}}_{-i'}\right\|_\infty\right),$$

where $\mathbf{1}_{\left\{\hat{\theta}_i > \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right\}} \left(\theta_i - \left\|\hat{\boldsymbol{\theta}}_{-i}\right\|_\infty\right)$ is the agent's surplus and $\sum_{i' \neq i} \mathbf{1}_{\left\{\hat{\theta}_{i'} > \left\|\hat{\boldsymbol{\theta}}_{-i'}\right\|_\infty\right\}} \left(\theta_{i'} - \left\|\hat{\boldsymbol{\theta}}_{-i'}\right\|_\infty\right)$ is the surplus of the other agents. The closer $\alpha_i$ is to zero, the more spiteful bidder $i$ is because his utility depends less on his own surplus and more on the other agents' surplus.

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

**Lemma 10.4.33.** *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 $\left\{\left\|\boldsymbol{\theta}_{-i}^{(j)}\right\|_\infty\right\}_{j \in [N]}$.*

*Proof.* Consider an arbitrary bidder, and without loss of generality, suppose that bidder is bidder 1. Next, choose a sample $\boldsymbol{\theta}_{-1}^{(j)} = \left(\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right)$. Letting $j^* = \operatorname{argmax}\left\{\theta_2^{(j)}, \ldots, \theta_n^{(j)}\right\}$ and letting $s^{(j)}$ be the second-largest component of $\boldsymbol{\theta}_{-1}^{(j)}$, we know that either agent 1 will win and pay $\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$ or agent $j^*$ will win and pay the maximum of $\hat{\theta}_1$ and $s^{(j)}$. Therefore, for any value $\theta_1 \in [0,1]$ and bid $\hat{\theta}_1 \in [0,1]$, we know that

$$u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right) = \begin{cases} (\alpha_1 - 1)\left(\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty - s^{(j)}\right) & \text{if } \hat{\theta}_1 \leq s^{(j)} \\ (\alpha_1 - 1)\left(\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty - \hat{\theta}_1\right) & \text{if } s^{(j)} < \hat{\theta}_1 \leq \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty \\ \alpha_1\left(\theta_1 - \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty\right) & \text{if } \hat{\theta}_1 > \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty. \end{cases} \tag{10.13}$$

This means that if $\hat{\theta}_1 > \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$, then $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is a constant function of $\hat{\theta}_1$, whereas if $\hat{\theta}_1 \leq \left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$, then $u_{1,M}\left(\theta_1, \hat{\theta}_1, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is a continuous function of $\hat{\theta}_1$ with a slope of either 0 (if $\hat{\theta}_1 < s^{(j)}$) or $\alpha_1 - 1$ (if $\hat{\theta}_1 \geq s^{(j)}$). We illustrate this function in Figure 10.3. Since $|\alpha_1 - 1| \leq 1$, this means that for all $\theta_i \in [0,1]$, the function $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ is piecewise 1-Lipschitz with a discontinuity at $\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty$. □

Lemma 10.4.33 implies the following dispersion guarantee.

**Theorem 10.4.34.** *Suppose each agent's type has a $\kappa$-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.*

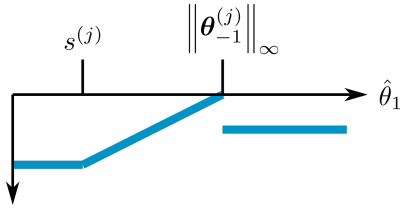



Figure 10.3: Graph of $u_{1,M}\left(\theta_1, \cdot, \boldsymbol{\theta}_{-1}^{(j)}\right)$ (Equation (10.13) ) for a spiteful agent under a second-price auction with $\alpha_1 = \theta_1 = \frac{1}{2}$ and $\left\|\boldsymbol{\theta}_{-1}^{(j)}\right\|_\infty = \frac{3}{4}$. We use the notation $s^{(j)}$ to denote the second-largest component of $\boldsymbol{\theta}_{-1}^{(j)}$, and in this figure, $s^{(j)} = \frac{1}{4}$.

*Proof.* This follows from Lemma 10.4.33 exactly as Theorem 10.4.13 follows from Lemma 10.4.12. □

**Theorem 10.4.35.** *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.*

*Proof.* Since the reported types $\left(\hat{\theta}_i, \boldsymbol{\theta}_{-i}\right)$ are fixed, the allocation is fixed. If $\hat{\theta}_i > \|\boldsymbol{\theta}_{-i}\|_\infty$, then $u_{i,M}\left(\theta_i, \hat{\theta}_i, \boldsymbol{\theta}_{-i}\right) = \alpha_i\left(\theta_i - \|\boldsymbol{\theta}_{-i}\|_\infty\right)$, which is an $\alpha_i$-Lipschitz function of $\theta_i$. Otherwise, it is a constant function of $\theta_i$. □

We conclude with the following pseudo-dimension bound.

**Theorem 10.4.36.** *For any agent $i \in [n]$, the pseudo-dimension of the class $\mathcal{F}_{i,M}$ is $O(1)$.*

*Proof.* For any vector $\boldsymbol{\theta}_{-i} \in [0,1]^{(n-1)}$, let $s$ be the second-highest component. Letting $\mathcal{H} = \{\hat{\theta}_i = \|\boldsymbol{\theta}_{-i}\|_\infty, \hat{\theta}_i = s\}$ we know that for any interval $I$ of $[0,1] \setminus \mathcal{H}$, the allocation is fixed across all $\hat{\theta}_i \in C$. So long as the allocation is fixed, $u_{i,M}\left(\cdot, \cdot, \boldsymbol{\theta}_{-i}\right)$ is a linear function of $\left(\theta_i, \hat{\theta}_i\right)$. Therefore, the pseudo-dimension bound follows directly from Theorem 10.4.11. □

## 10.5 Dispersion lemmas

We include a few lemmas about *dispersion* (Definition 10.4.5) from work by Balcan et al. [2018b] as well as a few variations we prove ourselves for our specific applications.

**Lemma 10.5.1** (Balcan et al. [2018b])**.** *Let $\mathcal{B} = \{\beta_1, \ldots, \beta_r\} \subset \mathbb{R}$ be a collection of samples where each $\beta_i$ is drawn from a distribution with a $\kappa$-bounded density function. For any $\delta \geq 0$, the following statements hold with probability at least $1 - \delta$:*

1. *If the $\beta_i$ are independent, then every interval of width $w$ contains at most $k = O(rw\kappa + \sqrt{r\log(1/\delta)})$ samples. In particular, for any $\alpha \geq 1/2$ we can take $w = 1/(\kappa r^{1-\alpha})$ and $k = O(r^\alpha\sqrt{\log(1/\delta)})$.*

2. *If the samples can be partitioned into $P$ buckets $\mathcal{B}_1, \ldots, \mathcal{B}_P$ such that each $\mathcal{B}_i$ contains independent samples and $|\mathcal{B}_i| \leq M$, then every interval of width $w$ contains at most $k = O(PMw\kappa + \sqrt{M\log(P/\delta)}$ samples. In particular, for any $\alpha \geq 1/2$ we can take $w = 1/(\kappa M^{1-\alpha})$ and $k = O(PM^\alpha\sqrt{\log(P/\delta)})$.*

**Lemma 10.5.2** (Balcan et al. [2018b])**.** *Suppose $X$ is a random variable with $\kappa$-bounded density function and suppose $c \neq 0$ is a constant. Then $cX$ has a $\frac{\kappa}{|c|}$-bounded density function.*

**Lemma 10.5.3.** *Suppose $X$ and $Y$ are random variables taking values in $[0,1]$ and suppose that their joint distribution has a $\kappa$-bounded density function. Then the distribution of $X+Z$ has a $\kappa$-bounded density function.*

*Proof.* Let $Z = X + Y$. We will perform change of variables using the function $g(x,y) = (x, x+y)$. Let $g^{-1}(x,z) = h(x,z) = (x, z-x)$. Then

$$J_h(x,z) = \det \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} = 1.$$

Therefore, $f_{X,Z}(x,z) = f_{X,Y}(x, z-x)$. This means that $f_Z(z) = \int_0^1 f_{X,Y}(x, z-x)\,dx \leq \kappa$, so the theorem statement holds. □

**Lemma 10.5.4.** *Suppose $X$ and $Y$ are random variables taking values in $[0,1]$ and suppose that their joint distribution has a $\kappa$-bounded density function. Then the distribution of $X-Z$ has a $\kappa$-bounded density function.*

*Proof.* Let $Z = X - Y$. We will perform change of variables using the function $g(x,y) = (x, x-y)$. Let $g^{-1}(x,z) = h(x,z) = (x, x-z)$. Then

$$J_h(x,z) = \det \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix} = -1.$$

Therefore, $f_{X,Z}(x,z) = f_{X,Y}(x, x-z)$. This means that $f_Z(z) = \int_0^1 f_{X,Z}(x,z)\,dx = \int_0^1 f_{X,Y}(x, x-z)\,dx \leq \kappa$, so the theorem statement holds. □

**Definition 10.5.5** (Hyperplane delineation)**.** Let $\Psi$ be a set of hyperplanes in $\mathbb{R}^m$ and let $\mathcal{P}$ be a partition of a set $\mathbb{R}^m$. Let $K_1, \ldots, K_q$ be the connected components of $\mathbb{R}^m \setminus \Psi$. Suppose every set in $\mathcal{P}$ is the union of some collection of sets $K_{i_1}, \ldots, K_{i_j}$ together with their limit points. Then we say that the set $\Psi$ *delineates* $\mathcal{P}$.

**Lemma 10.5.6** (Balcan et al. [2018b])**.** *Let $u_1, \ldots, u_N$ be a set of piecewise $L$-Lipschitz functions mapping $\mathbb{R}^m$ to $\mathbb{R}$, drawn i.i.d. from a distribution $\mathcal{D}$. For each $j \in [N]$, let $\mathcal{P}_j$ be the partition of $[0,1]^m$ such that over any $R \in \mathcal{P}_j$, $u_j$ is $L$-Lipschitz. Suppose the hyperplane sets $\Psi_1, \ldots, \Psi_N$ delineate the partitions $\mathcal{P}_1, \ldots, \mathcal{P}_N$. Moreover, suppose the multi-set union of $\Psi_1, \ldots, \Psi_N$ can be partitioned into $P$ multi-sets $\mathcal{B}_1, \ldots, \mathcal{B}_P$ such that for each multi-set $\mathcal{B}_i$:*

1. *The hyperplanes in $\mathcal{B}_i$ are parallel with probability 1 over the draw of $u_1, \ldots, u_N$.*
2. *The offsets of the hyperplanes in $\mathcal{B}_i$ are independently drawn from $\kappa$-bounded distributions.*

*With probability at least $1-\delta$ over the draw of $u_1, \ldots, u_N$, the functions are*

$$\left( \frac{1}{2\kappa\sqrt{\max |\mathcal{B}_i|}}, O\left( P\sqrt{\max |\mathcal{B}_i| \ln \frac{P}{\delta}} \right) \right) \text{-dispersed.}$$

# Part II

# Efficient procedures for algorithm configuration and beyond

# Chapter 11

# Frugal training with generalization guarantees

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. These sample complexity bounds apply no matter how the learning algorithm operates, and these papers do not include learning algorithms that extend beyond exhaustive search.

The second line of research provides algorithms for finding nearly-optimal configurations from a finite set [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019]. These algorithms can also be used when the parameter space is infinite: for any $\gamma \in (0, 1)$, first sample $\tilde{\Omega}(1/\gamma)$ configurations, and then run the algorithm over this finite set. The authors guarantee that the output configuration will be within the top $\gamma$-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 4, 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 11.4.

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 [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019], which we prove will then return a nearly optimal parameter from the infinite set. Our approach applies to any configuration problem where the algorithm's performance as a function of its parameters—the dual function—is piecewise constant. We proved that this structure holds in the previous chapters of this thesis in the contexts of integer programming and computational biology algorithm configuration, and other papers have shown that it holds in the context of clustering as well [Balcan et al., 2017, 2018c, 2020a].

We now describe our algorithm at a high level. Let $\ell$ 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 $\infty$. 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 $\ell$, 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 11.0.1** (Informal)**.** *The following guarantees hold:*

1. *The set of output parameters contains a nearly-optimal parameter with high probability.*
2. *Given accuracy parameters $\epsilon$ and $\delta$, 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 [Balcan et al., 2020e]. Our current results were published in AAAI 2020.

### Subsequent research

Subsequent research by Weisz et al. [2020] extends the configuration procedure by Weisz et al. [2019], enabling it to quickly discard suboptimal configurations without running the risk of discarding nearly-optimal configurations. It can thus operate with a better choice of $\gamma$ compared to prior research [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019]. As in the previous approaches, if there is only a small region of high-performing parameters, however, a uniform sample of configurations will likely miss all good parameter settings.

## 11.1 Problem definition

The algorithm configuration model we adopt is a generalization of the model from prior research by Kleinberg et al. [2017, 2019] and Weisz et al. [2018, 2019]. There is a set $\mathcal{Z}$ of problem instances and an unknown distribution $\mathcal{D}$ over $\mathcal{Z}$. Each algorithm is parameterized by a vector

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

$\boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^d$. 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, $\tau$ might measure the maximum running time we allow the algorithm. There is a utility function $U : \mathcal{P} \times \mathcal{Z} \times \mathbb{Z}_{\geq 0} \to \{0, 1\}$, 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 $\tau$. 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 $\tau$, and otherwise, $\ell_{\boldsymbol{\rho}}(z) = \operatorname{argmin} \{\tau : U(\boldsymbol{\rho}, z, \tau) = 1\}$. In Section 11.1.1, 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

$$\mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[\ell_{\hat{\boldsymbol{\rho}}}(z)\right] \leq (1 + \epsilon) \inf_{\boldsymbol{\rho} \in \mathcal{P}} \left\{ \mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[\ell_{\boldsymbol{\rho}}(z)\right] \right\}. \tag{11.1}$$

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 11.1.1.** [Weisz et al. [2019]] Let $\mathcal{P} = \{1, 2\}$ be a set of two configurations. Suppose that the loss of the first configuration is 2 for all problem instances: $\ell_1(z) = 2$ for all $z \in \mathcal{Z}$. Meanwhile, suppose that $\ell_2(z) = \infty$ with probability $\delta$ 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 $\delta$ 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 (11.1).

The obstacle that this example exposes is that some configurations might have an enormous loss on a few rare problem instances. To deal with this impossibility result, Weisz et al. [2018, 2019], building off of work by Kleinberg et al. [2017, 2019], propose a relaxed notion of approximate optimality. To describe this relaxation, we introduce the following notation. Given $\delta \in (0, 1)$ and a parameter vector $\boldsymbol{\rho} \in \mathcal{P}$, let $t_\delta(\boldsymbol{\rho})$ be the largest cutoff $\tau \in \mathbb{Z}_{\geq 0}$ such that the probability $\ell_{\boldsymbol{\rho}}(z)$ is greater than $\tau$ is at least $\delta$. 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 "$\delta$-tail." We illustrate the definition of $t_\delta(\boldsymbol{\rho})$ in Figure 11.1. We now define the relaxed notion of approximate optimality by Weisz et al. [2018].

**Definition 11.1.2** ($(\epsilon, \delta, \mathcal{P})$-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 $\delta$-capped expected loss is within a $(1 + \epsilon)$-factor of the optimal $\delta/2$-capped expected loss.[2] To condense notation, we write

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

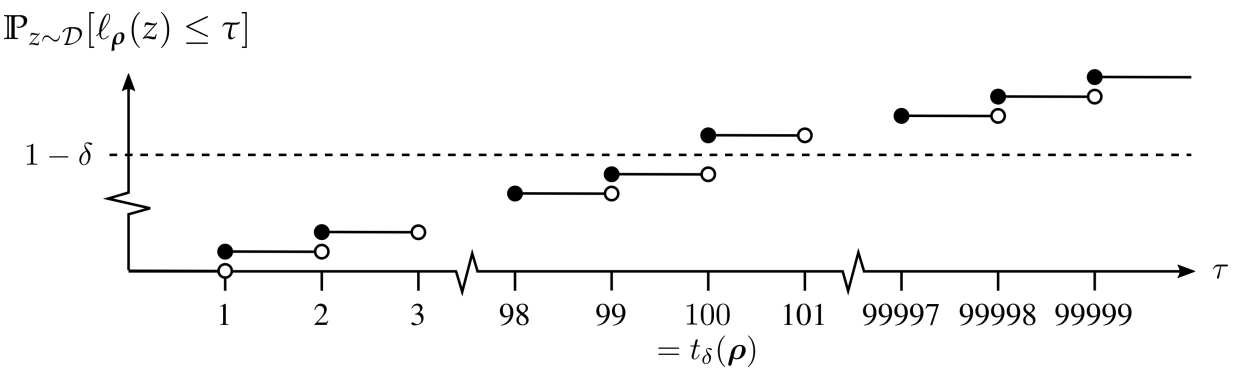



Figure 11.1: Fix a parameter vector $\boldsymbol{\rho}$. The figure is a hypothetical illustration of the cumulative density function of $\ell_{\boldsymbol{\rho}}(z)$ when $z$ is sampled from $\mathcal{D}$. For each value $\tau$ along the $x$-axis, the solid line equals $\Pr_{z\sim\mathcal{D}}\left[\ell_{\boldsymbol{\rho}}(z) \leq \tau\right]$. 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 $t_\delta(\boldsymbol{\rho}) = 100$.

$OPT_{c\delta} := \inf_{\boldsymbol{\rho}\in\mathcal{P}}\left\{\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell_{\boldsymbol{\rho}}(z), t_{c\delta}(\boldsymbol{\rho})\right\}\right]\right\}$. 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. [2019] provide one such algorithm, CapsAndRuns.

### 11.1.1 Example applications

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

**Tree search.** Tree search algorithms, which we have studied in Chapters 4, 8, and 9, are the most widely-used tools for solving combinatorial problems. Given parameters $\boldsymbol{\rho}$ and a problem instance $z$, we might define the budget $\tau$ 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 $\tau$. 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. [2017] provide sample complexity guarantees for clustering algorithm configuration. Each problem instance is a set of datapoints and there is a distribution over clustering problem instances. They analyze several infinite classes of clustering algorithms. Each of these algorithms begins with a linkage-based step and concludes with a dynamic programming step. The linkage-based routine constructs a hierarchical tree of clusters. At the beginning of the pro-

cess, each datapoint is in a cluster of its own. The algorithm sequentially merges the clusters into larger clusters until all elements are in the same cluster. There are many ways to build this tree: merge the clusters that are closest in terms of their two closest points (*single-linkage*), their two farthest points (*complete-linkage*), or on average over all pairs of points (*average-linkage*). These linkage procedures are commonly used in practice [Awasthi et al., 2017, Saeed et al., 2003, White et al., 2010] and come with theoretical guarantees. Balcan et al. [2017] study an infinite parameterization, *ρ-linkage*, that interpolates between single-, average-, and complete-linkage. After building the cluster tree, the dynamic programming step returns the pruning of this tree that minimizes a fixed objective function, such as the $k$-means, $k$-median, or $k$-center objectives.

Building the full hierarchy is expensive: the best-known algorithm's runtime is $O(n^2 \log n)$, where $n$ is the number of datapoints [Manning et al., 2010]. It is not always necessary, however, to build the entire tree: the algorithm can preemptively terminate the linkage step after $\tau$ merges, then use dynamic programming to recover the best pruning of the cluster forest. We refer to this variation as *$\tau$-capped $\rho$-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 $\tau$-capped $\rho$-linkage returns, given the instance $z$ as input. We assume there is a threshold $\theta_z$ 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 $\theta_z$ might equal the optimal $k$-means objective value (obtained only for the training instances via an expensive computation) plus an error term.

## 11.2 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 [Weisz et al., 2019], returns a parameter that's nearly optimal over the infinite set $\mathcal{P}$. Therefore, our goal is to provide an algorithm that takes as input an infinite parameter space and returns a finite subset that contains at least one good parameter. A bit more formally, a parameter is "sufficiently good" if its $\delta/2$-capped expected loss is within a $\sqrt{1+\epsilon}$-factor of $OPT_{\delta/4}$. We say a finite parameter set $\bar{\mathcal{P}}$ is an $(\epsilon, \delta)$-*optimal subset* if it contains a good parameter.

**Definition 11.2.1** ($(\epsilon, \delta)$-optimal subset)**.** A finite set $\bar{\mathcal{P}} \subset \mathcal{P}$ is an $(\epsilon, \delta)$-*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 $(\epsilon, \delta)$-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 11.2.2.** *Let $\bar{\mathcal{P}} \subset \mathcal{P}$ be an $(\epsilon, \delta)$-optimal subset and let $\epsilon' = \sqrt{1+\epsilon} - 1$. Suppose $\hat{\boldsymbol{\rho}} \in \bar{\mathcal{P}}$ is $\left(\epsilon', \delta, \bar{\mathcal{P}}\right)$-optimal. Then $\mathbb{E}_{z \sim \mathcal{D}} \left[\min \left\{\ell_{\hat{\boldsymbol{\rho}}}(z), t_{\delta}\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \leq (1+\epsilon) \cdot OPT_{\delta/4}$.*

*Proof.* Since the parameter $\hat{\boldsymbol{\rho}}$ is $\left(\epsilon', \delta, \bar{\mathcal{P}}\right)$-optimal, we know that $\mathbb{E}_{z \sim \mathcal{D}} \left[\min \left\{\ell_{\hat{\boldsymbol{\rho}}}(z), t_{\delta}\left(\hat{\boldsymbol{\rho}}\right)\right\}\right] \leq \sqrt{1+\epsilon} \cdot \min_{\boldsymbol{\rho} \in \bar{\mathcal{P}}} \left\{\mathbb{E}_{z \sim \mathcal{D}} \left[\min \left\{\ell\left(\boldsymbol{\rho}, z\right), t_{\delta/2}(\boldsymbol{\rho})\right\}\right]\right\}$. (We use a minimum instead of an infimum because $\bar{\mathcal{P}}$ is a finite set by Definition 11.2.1.) The set $\bar{\mathcal{P}}$ is an $(\epsilon, \delta)$-optimal subset of the parameter space $\mathcal{P}$, so there exists a parameter vector $\boldsymbol{\rho}' \in \bar{\mathcal{P}}$ such that $\mathbb{E}_{z \sim \mathcal{D}} \left[\min \left\{\ell\left(\boldsymbol{\rho}', z\right), t_{\delta/2}\left(\boldsymbol{\rho}'\right)\right\}\right] \leq$

$\sqrt{1+\epsilon} \cdot OPT_{\delta/4}$. Therefore,

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

so the theorem statement holds. □

## 11.3 Main result: An algorithm for learning $(\epsilon, \delta)$-optimal subsets

We present an algorithm for learning $(\epsilon, \delta)$-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 $\boldsymbol{\rho}, \boldsymbol{\rho}' \in R$, $\min\left\{\ell_{\boldsymbol{\rho}}(z), \tau\right\} = \min\left\{\ell_{\boldsymbol{\rho}'}(z), \tau\right\}$.

To exploit this piecewise-constant structure, we require access to a function Partition that takes as input a set $\mathcal{S}$ of problem instances and an integer $\tau$ 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 $z \in \mathcal{S}$, $\min\left\{\ell_{\boldsymbol{\rho}}(z), \tau\right\} = \min\left\{\ell_{\boldsymbol{\rho}'}(z), \tau\right\} = \tau_i[z]$.

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

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

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

Results from prior research imply guidance for implementing Partition in the contexts of clustering and integer programming. For example, in the clustering application we describe in Section 11.1.1, 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. [2017] prove that for any set $\mathcal{S}$ of samples and any cap $\tau$, in the worst case, $|\textsc{Partition}(\mathcal{S}, \tau)| = O\left(|\mathcal{S}|n^8\right)$, though empirically, $|\textsc{Partition}(\mathcal{S}, \tau)|$ is often several orders of magnitude smaller [Balcan et al., 2020a]. Balcan et al. [2017, 2020a] provide guidance for implementing Partition.

**High-level description of algorithm.** We now describe our algorithm for learning $(\epsilon, \delta)$-optimal subsets. See Algorithm 8 for the pseudocode. The algorithm maintains a variable $T$, initially set to $\infty$, 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 $\ell$ it computes by 1. At the beginning

**Algorithm 8** Algorithm for learning $(\epsilon, \delta)$-optimal subsets

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

1: Set $\eta \leftarrow \min\left\{\frac{1}{8}\left(\sqrt[4]{1+\epsilon}-1\right), \frac{1}{9}\right\}$, $t \leftarrow 1$, $T \leftarrow \infty$, and $\mathcal{G} \leftarrow \emptyset$.

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

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

4: **while** $\eta\delta$ is smaller than

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

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

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

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

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

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

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

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

11: $t \leftarrow t + 1$.

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

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

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* [Bartlett and Mendelson, 2002, Koltchinskii, 2001].

Next, the algorithm evaluates the function $\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)$ 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 parameter from each set in $\mathcal{G}$.

**Algorithm analysis.** We now provide guarantees on Algorithm 8'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 11.3.1.** *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 8 terminates after $\bar{t} = O\left(\log\left(c \cdot OPT_{\delta/4}\right)\right)$ iterations.*
2. *Algorithm 8 returns an $(\epsilon, \delta)$-optimal set of parameters of size at most*

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

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

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

*Proof.* We split the proof into separate lemmas. Lemma 11.3.4 proves Part 1. Lemma 11.3.6 as well as Lemma B.0.10 in Appendix B prove Part 2. Finally, Part 3 follows from classic results in learning theory on Rademacher complexity. In particular, it follows from an inversion of the inequality in Step 4 and the fact that $2^t \leq 2^{\bar{t}} \leq c \cdot OPT_{\delta/4}$, as we prove in Lemma 11.3.4. □

Theorem 7.2.3 hinges on the assumption that the samples $\mathcal{S}_1, \ldots, \mathcal{S}_{\bar{t}}$ Algorithm 8 draws in Step 4 are sufficiently representative of the distribution $\mathcal{D}$, formalized as follows:

**Definition 11.3.2** ($\zeta$-representative run)**.** For each round $t \in [\bar{t}]$, denote the samples in $\mathcal{S}_t$ as $\mathcal{S}_t = \left\{z_i^{(t)} : i \in [|\mathcal{S}_t|]\right\}$. We say that Algorithm 8 has a *$\zeta$-representative run* if for all rounds $t \in [\bar{t}]$, all integers $b \in [|\mathcal{S}_t|]$, all caps $\tau \in \mathbb{Z}_{\geq 0}$, and all parameters $\boldsymbol{\rho} \in \mathcal{P}$, the following conditions hold:

1. The average number of instances $z_i^{(t)} \in \left\{z_1^{(t)}, \ldots, z_b^{(t)}\right\}$ with loss smaller than $\tau$ nearly matches the probability that $\ell_{\boldsymbol{\rho}}(z) \leq \tau$:

$$\left|\frac{1}{b}\sum_{i=1}^{b} \mathbf{1}_{\left\{\ell\left(\boldsymbol{\rho}, z_i^{(t)}\right) \leq \tau\right\}} - \Pr_{z\sim\mathcal{D}}\left[\ell_{\boldsymbol{\rho}}(z) \leq \tau\right]\right| \leq \gamma(t, b, \tau),$$

   and

2. The average $\tau$-capped loss of the instances $z_1^{(t)}, \ldots, z_b^{(t)}$ nearly matches the expected $\tau$-capped loss:

$$\left|\frac{1}{b}\sum_{i=1}^{b} \min\left\{\ell\left(\boldsymbol{\rho}, z_i^{(t)}\right), \tau\right\} - \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell_{\boldsymbol{\rho}}(z), \tau\right\}\right]\right| \leq \tau \cdot \gamma(t, b, \tau),$$

where

$$\gamma(t, b, \tau) = \sqrt{\frac{2d \ln\left|\textsc{Partition}\left(\left\{z_1^{(t)}, \ldots, z_b^{(t)}\right\}, \tau\right)\right|}{b}} + 2\sqrt{\frac{2}{b}\ln\frac{8\left(\tau b t\right)^2}{\zeta}}.$$

In Step 4, we ensure $\mathcal{S}_t$ is large enough that Algorithm 8 has a $\zeta$-representative run with probability $1-\zeta$.

**Lemma 11.3.3.** *With probability $1-\zeta$, Algorithm 8 has a $\zeta$-representative run.*

*Proof.* Intuitively, there are only $|\textsc{Partition}(\mathcal{S},\tau)|$ algorithms with varying $\tau$-capped losses over any set of samples $\mathcal{S}$. We can therefore invoke Massart's finite lemma [Massart, 2000], which guarantees that each set $\mathcal{S}_t$ is sufficiently large to ensure that Algorithm 8 indeed has a $\zeta$-representative run. More formally, for any $\tau \in \mathbb{Z}_{\geq 0}$, define the function classes $\mathcal{F}_\tau = \{z \mapsto \min\{\ell(\boldsymbol{\rho},z),\tau\} \mid \boldsymbol{\rho} \in \mathcal{P}\}$ and $\mathcal{H}_\tau = \left\{z \mapsto \mathbf{1}_{\{\ell(\boldsymbol{\rho},z)\leq\tau\}} \mid \boldsymbol{\rho} \in \mathcal{P}\right\}$. Let $\mathcal{S} \subseteq \Pi$ be a set of problem instances. By Massart's lemma (Lemma 2.1.8 in Chapter 2) we know that $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{F}_\tau) \leq \tau\sqrt{\frac{2d\ln|\textsc{Partition}(\mathcal{S},\tau)|}{|\mathcal{S}|}}$ and $\widehat{\mathscr{R}}_{\mathcal{S}}(\mathcal{H}_\tau) \leq \sqrt{\frac{2d\ln|\textsc{Partition}(\mathcal{S},\tau)|}{|\mathcal{S}|}}$. Therefore, the lemma holds. □

The remainder of our analysis will assume that Algorithm 8 has a $\zeta$-representative run.

**Number of iterations until termination.** We begin with a proof sketch of the first part of Theorem 7.2.3.

**Lemma 11.3.4.** *Suppose Algorithm 8 has a $\zeta$-representative run. Then $2^{\bar{t}} \leq \frac{16}{\delta}\sqrt[4]{1+\epsilon}\cdot OPT_{\delta/4}$.*

*Proof.* For each set $\mathcal{P}' \in \bar{\mathcal{G}}$, let $t_{\mathcal{P}'}$ be the round where $\mathcal{P}'$ is added to the set $\mathcal{G}$, let $\mathcal{S}_{\mathcal{P}'} = \mathcal{S}_{t_{\mathcal{P}'}}$, and let $\boldsymbol{\rho}_{\mathcal{P}'}$ be an arbitrary parameter vector in $\mathcal{P}'$. Since no set is added to $\mathcal{G}$ when $t = \bar{t}$, it must be that for all sets $\mathcal{P}' \in \bar{\mathcal{G}}$, $t_{\mathcal{P}'} \leq \bar{t}-1$. Moreover, since

$$OPT_{\delta/4} := \inf_{\boldsymbol{\rho}\in\mathcal{P}}\left\{\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho},z\right),t_{\delta/4}(\boldsymbol{\rho})\right\}\right]\right\},$$

we know that for every $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho}^*$ such that

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

Below, we prove that $2^{\bar{t}} \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,z\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]$ and thus the lemma statement holds (see Lemma B.0.9).

**Case 1: $\boldsymbol{\rho}^* \notin \bigcup_{\mathcal{P}'\in\bar{\mathcal{G}}}\mathcal{P}'$.** By Lemma 11.3.5, we know that

$$2^{\bar{t}-3}\delta \leq \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,z\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right].$$

Therefore, $2^{\bar{t}} \leq \frac{8}{\delta}\cdot\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,z\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right] \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,z\right),t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right]$.

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

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

Since Algorithm 8 had a $\zeta$-representative run, $2^{\bar{t}-4}\delta$ is upper-bounded by

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

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

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

By Step 4 of Algorithm 8, $2^{\bar{t}-4}\delta \leq \mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, z\right), \tau_{\mathcal{P}'}\right\}\right] + \tau_{\mathcal{P}'}\eta\delta$. Finally, by Corollary B.0.4, $2^{\bar{t}-4}\delta \leq (1+4\eta)\,\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, z\right), \tau_{\mathcal{P}'}\right\}\right]$. Since $\eta < \left(\sqrt[4]{1+\epsilon} - 1\right)/4$,

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

Recalling that $\tau_{\mathcal{P}'} \leq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)$ by Lemma B.0.1, we conclude that

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

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

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

Since Algorithm 8 had a $\zeta$-representative run,

$$\begin{aligned}
&\Pr_{z\sim\mathcal{D}}\left[\ell\left(\boldsymbol{\rho}^*, z\right) \leq 2^{\bar{t}-2}\right] \\
&\leq \frac{1}{\left|\mathcal{S}'\right|}\sum_{z\in\mathcal{S}'}\mathbf{1}_{\left\{\ell\left(\boldsymbol{\rho}^*, z\right)\leq 2^{\bar{t}-2}\right\}} + \sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}', 2^{\bar{t}-2}\right)\right|}{\left|\mathcal{S}'\right|}} + 2\sqrt{\frac{2}{\left|\mathcal{S}'\right|}\ln\frac{8\left(2^{\bar{t}-2}\left|\mathcal{S}'\right|\left(\bar{t}-2\right)\right)^2}{\zeta}} \\
&< 1 - \frac{3\delta}{8} + \sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}', 2^{\bar{t}-2}\right)\right|}{\left|\mathcal{S}'\right|}} + 2\sqrt{\frac{2}{\left|\mathcal{S}'\right|}\ln\frac{8\left(2^{\bar{t}-2}\left|\mathcal{S}'\right|\left(\bar{t}-2\right)\right)^2}{\zeta}}.
\end{aligned}$$

Based on Step 4 of Algorithm 8, $\Pr_{z\sim\mathcal{D}}\left[\ell\left(\boldsymbol{\rho}^*, z\right) \leq 2^{\bar{t}-2}\right] < 1 - (3/8 - \eta)\delta$. Since $\eta \leq 1/9$, $\Pr_{z\sim\mathcal{D}}\left[\ell\left(\boldsymbol{\rho}^*, z\right) \leq 2^{\bar{t}-2}\right] < 1 - \delta/4$. Therefore,

$$\Pr_{z\sim\mathcal{D}}\left[\ell\left(\boldsymbol{\rho}^*, z\right) \geq 2^{\bar{t}-2}\right] \geq \Pr_{z\sim\mathcal{D}}\left[\ell\left(\boldsymbol{\rho}^*, z\right) > 2^{\bar{t}-2}\right] \geq \delta/4.$$

Since $t_{\delta/4}(\boldsymbol{\rho}^*) = \operatorname{argmax}_{\tau\in\mathbb{Z}}\left\{\Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho}^*, z) \geq \tau] \geq \delta/4\right\}$, we have that $2^{\bar{t}-2} \leq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)$. Therefore,

$$\begin{aligned}
\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, z\right), t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right] &\geq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\Pr\left[\ell\left(\boldsymbol{\rho}^*, z\right) \geq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right] \\
&\geq 2^{\bar{t}-2}\Pr\left[\ell\left(\boldsymbol{\rho}^*, z\right) \geq t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right] \\
&\geq \delta 2^{\bar{t}-4},
\end{aligned}$$

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

In the next lemma, we prove the upper bound on $2^{\bar{t}}$ that we use in Lemma 11.3.4.

**Lemma 11.3.5.** *Suppose Algorithm 8 has a $\zeta$-representative run. For any parameter vector $\boldsymbol{\rho} \notin \bigcup_{\mathcal{P}' \in \bar{\mathcal{G}}} \mathcal{P}'$, $2^{\bar{t}} \leq \frac{8}{\delta} \cdot \mathbb{E}_{z \sim \mathcal{D}} \left[ \min \left\{ \ell_{\boldsymbol{\rho}}(z), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right]$.*

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

$$
\begin{aligned}
&\Pr_{z \sim \mathcal{D}} \left[ \ell(\boldsymbol{\rho}, z) \leq 2^{\bar{t}-1} \right] \\
\leq\ & \frac{1}{|\bar{\mathcal{S}}|} \sum_{z \in \bar{\mathcal{S}}} \mathbf{1}_{\left\{\ell(\boldsymbol{\rho}, z) \leq 2^{\bar{t}-1}\right\}} + \sqrt{\frac{2d \ln \left| \textsc{Partition}\left(\bar{\mathcal{S}}, 2^{\bar{t}-1}\right) \right|}{|\bar{\mathcal{S}}|}} + 2\sqrt{\frac{2}{|\bar{\mathcal{S}}|} \ln \frac{8\left(2^{\bar{t}-1} |\bar{\mathcal{S}}| \left(\bar{t} - 1\right)\right)^2}{\zeta}} \\
\leq\ & \frac{1}{|\bar{\mathcal{S}}|} \sum_{z \in \bar{\mathcal{S}}} \mathbf{1}_{\left\{\ell(\boldsymbol{\rho}, z) \leq 2^{\bar{t}-1}\right\}} + \eta\delta,
\end{aligned}
$$

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

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

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

$$
\begin{aligned}
\mathop{\mathbb{E}}_{z \sim \mathcal{D}} \left[ \min \left\{ \ell(\boldsymbol{\rho}, z), t_{\delta/4}(\boldsymbol{\rho}) \right\} \right] &\geq t_{\delta/4}(\boldsymbol{\rho}) \Pr_{z \sim \mathcal{D}} \left[ \ell(\boldsymbol{\rho}, z) \geq t_{\delta/4}(\boldsymbol{\rho}) \right] \\
&\geq \frac{\delta}{4} \cdot t_{\delta/4}(\boldsymbol{\rho}) \\
&\geq 2^{\bar{t}-3}\delta.
\end{aligned}
$$

□

**Optimality of Algorithm 8's output.** Next, we provide a proof sketch of the second part of Theorem 7.2.3, which guarantees that Algorithm 8 returns an $(\epsilon, \delta)$-optimal subset. For each set $\mathcal{P}' \in \bar{\mathcal{G}}$, $\tau_{\mathcal{P}'}$ denotes the value of $\tau_{\lfloor |\mathcal{S}_t|(1-3\delta/8) \rfloor}$ in Step 9 of Algorithm 8 during the iteration $t$ that $\mathcal{P}'$ is added to $\mathcal{G}$.

**Lemma 11.3.6.** *If Algorithm 8 has a $\zeta$-representative run, it returns an $(\epsilon, \delta)$-optimal subset.*

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

$$\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}',z\right),t_{\delta/2}\left(\boldsymbol{\rho}'\right)\right\}\right] \leq \sqrt{1+\epsilon}\cdot\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho},z),t_{\delta/4}(\boldsymbol{\rho})\right\}\right], \tag{11.2}$$

and thus the lemma statement holds (see Lemma B.0.8).

First, suppose $\boldsymbol{\rho}$ is contained in a set $\mathcal{P}'\in\bar{\mathcal{G}}$. By Lemmas B.0.1 and B.0.7, there exists a parameter $\boldsymbol{\rho}'\in\mathcal{P}'\cap\mathcal{P}^*$ such that

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

Since $\sqrt[4]{1+\epsilon}\leq\sqrt{1+\epsilon}$, Equation (11.2) holds in this case.

Otherwise, suppose $\boldsymbol{\rho}\notin\bigcup_{\mathcal{P}'\in\bar{\mathcal{G}}}\mathcal{P}'$. By Lemma B.0.5, we know $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho},z),t_{\delta/4}(\boldsymbol{\rho})\right\}\right]\geq\bar{T}$. Moreover, by Lemma B.0.6, there exists a set $\mathcal{P}'\in\bar{\mathcal{G}}$ and parameter vector $\boldsymbol{\rho}^*\in\mathcal{P}'$ such that $\sqrt[4]{1+\epsilon}\cdot\bar{T}\geq\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*,z\right),\tau_{\mathcal{P}'}\right\}\right]$. Finally, by Lemma B.0.7, there exists a parameter vector $\boldsymbol{\rho}'\in\mathcal{P}'\cap\mathcal{P}^*$ such that

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

Therefore, Equation (11.2) holds in this case as well. □

The second part of Theorem 7.2.3 also guarantees that the size of the set Algorithm 8 returns is bounded (see Lemma B.0.10 in Appendix B). Together, Lemmas 11.3.4 and 11.3.6, as well as Lemma B.0.10 in Appendix B, bound the number of iterations Algorithm 8 makes until it returns an $(\epsilon,\delta)$-optimal subset.

## 11.4 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 11.1.1.

**Uniformly sampling configurations.** Prior research provides algorithms for finding nearly-optimal configurations from a finite set [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019]. If the parameter space is infinite and their algorithms optimize over a uniformly-sampled set of $\tilde{\Omega}(1/\gamma)$ configurations, then the output configuration will be within the top $\gamma$-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 4.3.1 from Chapter 4, restated here for convenience.)

**Theorem 11.4.1.** *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 $\rho$ on the input IP $z$.*

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 $\delta$-capped expected loss of any parameter in the range $[0,a] \cup [b,1]$ is at least $2^{(n-6)/2}$. Therefore, if we uniformly sample a finite set of parameters and optimize over this set using an algorithm for finite parameter spaces [Kleinberg et al., 2017, 2019, Weisz et al., 2018, 2019], we must ensure that we sample at least one parameter within $(a,b)$. As $a$ and $b$ converge, however, the required number of samples shoots to infinity, as we formalize below.

**Theorem 11.4.2.** *For the B&B configuration problem in Theorem 11.4.1, with constant probability over the draw of $m = \lfloor 1/(b-a) \rfloor$ parameters $\rho_1, \ldots, \rho_m \sim$ Uniform$[0,1]$, $\{\rho_1, \ldots, \rho_m\} \cap (a,b) = \emptyset$.*

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

Meanwhile, Algorithm 8 quickly terminates, having found an optimal parameter, as we describe below.

**Theorem 11.4.3.** *For the configuration problem in Theorem 11.4.1, Algorithm 8 ends after $\tilde{O}\left(\log \frac{1}{\delta}\right)$ iterations, having drawn $\tilde{O}((\delta\eta)^{-2})$ samples (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 $(a,b)$.*

*Proof.* In the proof of Theorem 11.4.1 (Theorem 4.3.1 from Chapter 4), we showed that for any distribution $\mathcal{D}$ in this family and any subset $\mathcal{S}$ from the support of $\mathcal{D}$, $|\textsc{Partition}(\mathcal{S}, \tau)| = 3$, and the partition Partition returns is $[0,a]$, $(a,b)$, and $[b,1]$. Therefore, for the first three iterations, our algorithm will use $\tilde{O}\left((\delta\eta)^{-2}\right)$ samples. At that point, $t = 3$, and the tree-size cap is 8. Algorithm 8 will discover that for all of the samples $z \in \mathcal{S}_3$, when $\rho \in (a,b)$, $\ell(\rho, z) = 8$. Therefore, it add $(a,b)$ to $\mathcal{G}$ and it will set $T = 8$. It will continue drawing $\tilde{O}\left((\delta\eta)^{-2}\right)$ samples at each round until $2^{t-3}\delta \geq T = 8$, or in other words, until $t = O(\log(1/\delta))$. Therefore, the total number of samples it draws is $\tilde{O}\left((\delta\eta)^{-2}\right)$. The set it returns will contain a point $\rho \in (a,b)$, which is optimal. □

Similarly, Balcan et al. [2018d] exemplify clustering configuration problems—which we overview in Section 11.1.1—where the optimal parameters lie within an arbitrarily small region, and any other parameter leads to significantly worse performance. As in Theorem 11.4.2, this means a uniform sampling of the parameters will fail to find optimal parameters.

# Chapter 12

# Learning to prune: Speeding up repeated computations

Throughout this thesis, 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 [Cesa-Bianchi and Lugosi, 2006], 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 12.1.

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

Figure 12.1: 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.

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 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 $\mathscr{U}$ that represents the entire search space. We call each subset a "pruning" of the search space. For example, in the shortest paths problem, $\mathscr{U}$ 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 $f = f_{\mathscr{U}}$. We assume that for every $z$, there is a set $S^*(z) \subseteq \mathscr{U}$ 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

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

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 $i \in [T]$, 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 12.1. 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 [Alabi et al., 2019]. Our current results were published in COLT 2019.

## 12.1 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.

As we described in Section 2.2, several papers provide guarantees for online algorithm configuration without distributional assumptions from a theoretical perspective [Balcan et al., 2018b, 2020b,c, Cohen-Addad and Kanade, 2017, Gupta and Roughgarden, 2017]. 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 [2017], Cohen-Addad and Kanade [2017], and Balcan et al. [2018b, 2020b,c] 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 [Banerjee and

[1] As we will formalize, when determining $S_1, \ldots, S_T$, our algorithm must compute the smallest set $S$ such that $f_S(z_i) = f(z_i)$ 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 $i \in [T]$.

Roy, 2015] and matching [Deb et al., 2006]. 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., Awerbuch and Kleinberg, 2008, Kalai and Vempala, 2005]. In such a setting, one cannot hope to return the best $y_i$ for each $x_i$ with 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.

## 12.2 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 12.2.1** (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 $z_1, ..., z_i$.

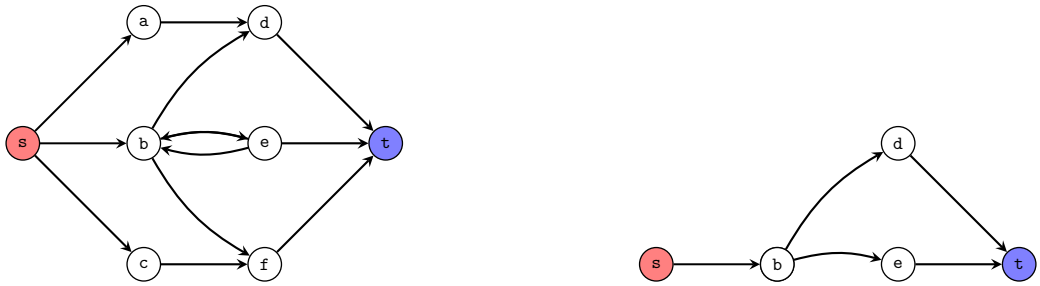
We assume each problem instance $z \in \mathcal{Z}$ has a unique correct solution (invoking tie-breaking assumptions as necessary; in Section 12.5, we discuss how to handle problems that admit multiple solutions). We denote the mapping from instances to correct solutions as $f : \mathcal{Z} \to Y$. For 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 12.5). To measure correctness, we use a *mistake bound model* [see, e.g., Littlestone, 1987].

**Definition 12.2.2** (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




(a) Original unweighted graph $G$. (b) Optimal pruning.

Figure 12.2: 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.

"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 12.2.3.** *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 $S^*(z) \subseteq S$.*

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 12.2 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 12.2.3 holds because $\mathscr{U} = E$, so $f_{\mathscr{U}}$ computes the shortest path on the entire graph. Part 2 of Assumption 12.2.3 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 12.5 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., $f(\boldsymbol{z}) = \operatorname{argmax}_{\boldsymbol{y} \in \mathbb{R}^n} \left\{ \boldsymbol{z}^T \boldsymbol{y} : A\boldsymbol{y} \leq \boldsymbol{b} \right\}$. For a subset of constraints $S \subseteq \mathscr{U}$, the function $f_S$ computes the optimal solution restricted to those constraints, i.e., $f_S(\boldsymbol{z}) = \operatorname{argmax}_{\boldsymbol{y} \in \mathbb{R}^n} \{ \boldsymbol{z}^T \boldsymbol{y} : A_S \boldsymbol{y} \leq \boldsymbol{b}_S \}$, 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 12.2.3 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 12.2.3 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 $\mathcal{Z}$ 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 12.2.3 holds: $f_{\mathscr{U}}(q, p) = f(q, p)$ for all $(q, p) \in \mathcal{Z}$, since $f_{\mathscr{U}}$ checks

every index in $[n - m + 1]$ for a match. Moreover, part 2 of Assumption 12.2.3 holds because $f_{\mathscr{U}}(z_i) = f_S(z_i)$ if and only if $S^*(z_i) = \{j_i^*\} \subseteq S$.

## 12.3 The algorithm

We now present an algorithm (Algorithm 9), 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 12.3.1, 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.

**Algorithm 9** 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 $\bar{S}_{i+1} \leftarrow \bar{S}_i$.

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 $i \in [T]$. To prove our guarantees, we use the following helpful lemma:

**Lemma 12.3.1.** *For any $z_1, \ldots, z_T \in \mathcal{Z}$, $S^* = \bigcup_{i=1}^{T} S^*(z_i)$.*

*Proof.* First, we prove that $S^* \supseteq \cup_{i=1}^{T} S^*(z_i)$. For a contradiction, suppose that for some $i \in [T]$, there exists an element $j \in S^*(z_i)$ such that $j \notin S^*$. This means that $f_{S^*}(z_i) = f(z_i)$, but $S^* \not\supseteq S^*(z_i)$, which contradicts Assumption 12.2.3: $S^*(z_i)$ is the unique smallest subset of $\mathscr{U}$ such that for any set $S \subseteq \mathcal{U}$, $f(z_i) = f_S(z_i)$ if and only if $S^*(z_i) \subseteq S$. Therefore, $S^* \supseteq \cup_{i=1}^{T} S^*(z_i)$. Next, let $C = \cup_{i=1}^{T} S^*(z_i)$. Since $S^*(z_i) \subseteq C$, Assumption 12.2.3 implies that $f(z_i) = f_C(z_i)$ for all $i \in [T]$. Based on the definition of $S^*$ and the fact that $S^* \supseteq C$, we conclude that $S^* = C = \cup_{i=1}^{T} S^*(z_i)$. □

We now provide a mistake bound for Algorithm 9.

**Theorem 12.3.2.** *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 9 has a mistake bound of*

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

*Proof.* Let $S_1, \dots, S_T$ be the sets such that on round $i$, Algorithm 9 computes the function $f_{S_i}$. Consider any element $e \in S^*$. Let $N_T(e)$ be the number of times $e \notin S_i$ but $e \in S^*(z_i)$ for some $i \in [T]$. In other words, $N_T(e) = |\{i : e \notin S_i, e \in S^*(z_i)\}|$. Every time the algorithm makes a mistake, the current set $S_i$ must not contain some $e \in S^*(z_i)$ (otherwise, $S_i \supseteq S^*(z_i)$, so the algorithm would not have made a mistake by Assumption 12.2.3). This means that every time the algorithm makes a mistake, $N_T(e)$ is incremented by 1 for at least one $e \in S^* = \cup_{i=1}^T S^*(z_i)$. Therefore,

$$M_T(\mathcal{A}^*, z_{1:T}) \leq \sum_{e \in S^*} \mathbb{E}[N_T(e)], \tag{12.1}$$

where the expectation is over the random choices of Algorithm 9.

For any element $e \in S^*$, let $i_1, \dots, i_t$ be the iterations where for all $\ell \in [t]$, $e \in S^*(z_{i_\ell})$. By definition, $N_T(e)$ will only be incremented on some subset of these rounds. Suppose $N_T(e)$ is incremented by 1 on round $i_r$. It must be that $e \notin S_{i_r}$, which means $S_{i_r} \neq \mathscr{U}$, and thus $S_{i_r} = \bar{S}_{i_r}$. Since $e \notin \bar{S}_{i_r}$, it must be that $e \notin \bar{S}_{i_\ell}$ for $\ell \leq r$ since $\bar{S}_{i_r} \supseteq \bar{S}_{i_{r-1}} \supseteq \cdots \supseteq \bar{S}_{i_1}$. Therefore, in each round $i_\ell$ with $\ell < r$, Algorithm 9 must not have computed $S^*(z_{i_\ell})$, because otherwise $e$ would have been added to the set $\bar{S}_{i_\ell+1}$. We can bound the probability of these bad events as

$$\mathbb{P}\left[S_{i_r} = \bar{S}_{i_r} \text{ and Algorithm 9 does not compute } S^*(z_{i_\ell}) \text{ for } \ell < r\right] = \prod_{\ell=1}^{r} (1 - p_{i_\ell}) \leq (1-p)^r.$$

As a result,

$$\mathbb{E}[N_T(e)] \leq \sum_{r=1}^{t} (1-p)^r \leq \sum_{r=1}^{T} (1-p)^r = \frac{(1-p)(1-(1-p)^T)}{p}. \tag{12.2}$$

The theorem statement follows by combining Equations (12.1) and (12.2). □

**Corollary 12.3.3.** *Algorithm 9 with $p_i = \frac{1}{\sqrt{i}}$ has a mistake bound of $M_T(\mathcal{A}^*, z_{1:T}) \leq |S^*|\sqrt{T}$.*

In Theorem 12.3.2, we bounded the expected number of mistakes Algorithm 9 makes. Next, we bound $\mathbb{E}\left[\frac{1}{T}\sum |S_i|\right]$, where $S_i$ is the set such that Algorithm 9 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 9.

**Theorem 12.3.4.** *For any inputs $z_1, \dots, z_T$, let $S_1, \dots, S_T$ be the sets such that on round $i$, Algorithm 9 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 9.*

*Proof.* We know that for all $i$, $S_i = \mathscr{U}$ with probability $p_i$ and $S_i = \bar{S}_i$ with probability $1 - p_i$. Therefore,

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

where the final inequality holds because $\bar{S}_i \subseteq S^*$ for all $i \in [T]$. □

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 12.3.5.** *Given a set of inputs $z_1, \ldots, z_T$, let $S_1, ..., S_T$ be the sets such that on round $i$, Algorithm 9 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 9.*

### 12.3.1 Instantiations of Algorithm 9

We now revisit and discuss instantiations of Algorithm 9 for the three applications outlined in Section 12.2: 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 12.3.3 is particularly strong when the shortest path does not vary much from day to day. Corollary 12.3.5 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.

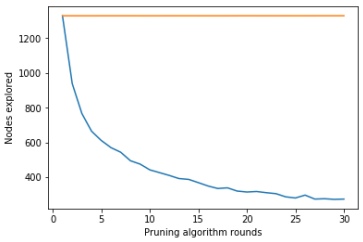
**Linear programming.** In the context of linear programming, computing the set $S^*(\boldsymbol{z}_i)$ is equivalent to computing $f(\boldsymbol{z})$ and returning the set of tight constraints. Since $S^* = \cup_{i=1}^{T} S^*(\boldsymbol{z}_i)$, the mistake bound of $|S^*|\sqrt{T}$ given by Corollary 12.3.3 is strongest when the same constraints are tight across most timesteps. Corollary 12.3.5 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 12.3.3 is particularly strong when the matching indices are similar across string pairs. Corollary 12.3.5 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.

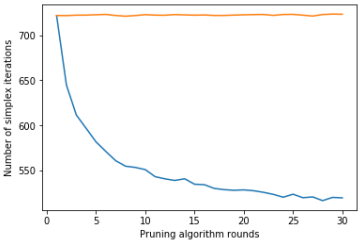
## 12.4 Experiments

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 9 explores.




(c) Top line: average number of simplex iterations the simplex algorithm makes. Bottom line: average number of simplex iterations Algorithm 9 makes.

Figure 12.3: Empirical evaluation of Algorithm 9 applied to shortest-path routing in Pittsburgh (Figures 12.3a and 12.3b) and linear programming (Figure 12.3c).

**Shortest-path routing.** We test Algorithm 9's performance on real-world street maps, which we access via Python's OSMnx package [Boeing, 2017]. 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 Appendix C, we experiment with alternative perturbation methods.

In Figures 12.3a and 12.3b, we illustrate our algorithm's performance in Pittsburgh. Figure 12.3a 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 12.3b 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. In Appendix C, we show a plot of the average pruned set size as a function of the number of rounds.

**Linear programming.** We generate linear programming instances representing the linear relaxation of the combinatorial auction winner determination problem (Example 4.2.1 from Section 4.2 in Chapter 4). We use the Combinatorial Auction Test Suite (CATS) [Leyton-Brown et al., 2000] 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 the $i^{th}$ round, we select a new objective vector $\boldsymbol{z}_i$ such that each component $z_i[j]$ is drawn independently from the normal distribution with a mean of $z[j]$ and a standard deviation of 1.

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 9 for 30 rounds ($T = 30$) with $p_i = 1/\sqrt{i}$ for all $i \in [T]$, and we repeat this 5000 times. In Figure 12.3c, 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 [Jones et al., 2001–], which implements the simplex algorithm [Dantzig, 2016]. Our algorithm returned the incorrect solution on a 0.018 fraction of the $5000 \cdot T = 150,000$ rounds. In Appendix C, we show a plot of the average pruned set size as a function of the number of rounds.

## 12.5 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.

# Part III

# Conclusions and future directions

Machine learning and optimization have the potential to transform the way we design algorithms, allowing us to fine-tune their performance to the specific application domain at hand. One powerful way of doing so is *automated algorithm configuration and selection*, where machine learning is used to tune algorithm parameters. These parameters may be made available explicitly for the user to tune, as is often the case in applied disciplines. For example, integer programming solvers expose over one hundred parameters for the user to tune. There may not be parameter settings that admit meaningful worst-case bounds, but with a deft configuration, these algorithms can quickly find solutions to computationally challenging problems. However, applied approaches to parameter tuning have rarely come with provable guarantees. Alternatively, an algorithm's parameters may be set implicitly, as is often the case in theoretical computer science: a proof may implicitly optimize over a parameterized family of algorithms in order to guarantee a worst-case approximation factor or runtime bound. Worst-case bounds, however, can be overly pessimistic in practice. This thesis helped develop the theoretical underpinnings of automated algorithm configuration and selection, placing these approaches on firm foundations.

We focused on a batch learning approach to automated algorithm configuration, where the configuration procedure's input is a training set (or "batch") of typical problem instances from the particular application domain at hand. The configuration procedure then returns a parameter configuration with strong average empirical performance over the training set, where performance is measured, for example, in terms of runtime, solution quality, or memory usage.

This batch learning approach to algorithm configuration raises two fundamental questions. First, how do we find a parameter setting with provably strong average empirical performance over the training set? Second, no matter what configuration we come up with—regardless of the specific procedure we use to optimize the parameters—can we guarantee that its average empirical performance is indicative of its future performance on problems from the same application but which are not already in our training set? This latter category of *sample complexity guarantees* apply no matter how the parameters are optimized, via an algorithmic search as in automated algorithm configuration [e.g., DeBlasio and Kececioglu, 2018, Sandholm, 2013, Xu et al., 2008, 2011], or manually as in *experimental algorithmics* [e.g., Bentley et al., 1984, Iyer et al., 2002, McGeoch, 2012].

In the first part of this thesis, we focused on providing sample complexity guarantees for a variety of algorithms from different settings: integer programming, computational biology, and economics (namely, selling and voting mechanisms). We analyzed polynomial-time approximation algorithms for integer quadratic programming in Chapter 3, where the goal is to optimize the quality of the solution that the algorithm returns. In Chapter 4, we studied branch-and-bound, the most widely-used algorithm for solving integer programs. We provided the first sample complexity guarantees for automated algorithm configuration in this setting and proved that a deft configuration of branch-and-bound's parameters can lead to an exponential improvement in the size of the tree that branch-and-bound builds. We then provided guarantees for automated *mechanism* configuration in Chapter 5, where the goal was to tune the parameters of economic mechanisms in order to maximize revenue and social welfare. In Chapter 6, we studied algorithms from computational biology for sequence alignment, RNA folding, and predicting topologically associated domains in DNA.

The next three chapters in Part I (Chapters 7-9) provided broadly-applicable sample complexity guarantees that apply to all of the above problems: integer programming, computa-

tional biology, and mechanism design. In Chapter 7, we formalized a unifying structure that links these disparate configuration problems: for any fixed problem instance, the algorithm's performance is a *piecewise-structured* function of its parameters. We proved that this structure implies extremely general guarantees for all of these problems simultaneously. We provided data-dependent sample complexity bounds in Chapter 8, whereas the guarantees in Chapters 3-7 are worst case: they hold for any worst-case distribution over problem instances, no matter how gnarly. With experiments, we showed that these data-dependent guarantees can provide significant improvements in the number of samples needed to obtain small generalization error. Chapters 3-8 focused on learning a *single* high-performing configuration, so in Chapter 9, we generalized our analysis to handle the problem of learning *portfolios* of multiple configurations. At runtime, a learned *algorithm selector* is used to choose a configuration from the portfolio to use. We provided guarantees for the problem of learning the portfolio in conjunction with an algorithm selector.

We concluded Part I with Chapter 10, where we studied the problem of estimating the extent to which a manipulable mechanism is *approximately* incentive compatible. We showed how this estimation can be done using samples from the distribution over agents' values. Increasingly, automated mechanism configuration and deep learning are being used to design manipulable mechanisms [Dütting et al., 2019, Feng et al., 2018, Golowich et al., 2018, Shen et al., 2019]. The tools from this chapter can be used to understand the extent to which agents can improve their utilities by behaving strategically under these machine-learned mechanisms.

Part I provided sample complexity bounds that apply to *any* procedure for optimizing over the parameters. In Part II, we focused on the question of *how* to optimize over the parameters, as well as other techniques for integrating machine learning into algorithm design. In Chapter 11, we provided a learning algorithm that finds provably optimal configurations from within a infinite set. This learning algorithm applies whenever the parameterized algorithm's performance is a piecewise-constant function of its parameters, a structure we observed holds across a diverse array of configuration problems in Chapter 7. Finally, in Chapter 12, we analyzed another means of using machine learning in the context of algorithm design, beyond the context of parameter configuration. We showed how to learn to prune irrelevant regions of the search space which never contain optimal solutions for the application domain at hand.

Research at the intersection of machine learning and algorithm design has the potential for both significant practical impact—for example, integration into large-scale algorithms such as commercial integer programming solvers—as well as deep theoretical analysis. Research on this topic from a theoretical perspective has only just begun, so there are many directions yet to be explored that can have a tangible, marked impact on algorithms used in practice. Below, we highlight some of the high-level future directions that the research in this thesis leaves unanswered.

**Dual function structure for faster configuration procedures.** This thesis deeply investigated statistical guarantees for automated algorithm configuration, which apply to any procedure for optimizing over the parameters. We and others have begun to answer the question of *how* to learn provably high-performing parameters (for example, the research in Chapter 11), but many open questions still remain. This thesis made a strong case for how understanding structure exhibited by the *dual* functions—measuring an algorithm's performance as a function of its parameters—can imply sample complexity guarantees for automated algorithm configuration. However, we are still only at the beginning of understanding how this dual function structure

can imply fast algorithms for learning provably high-performing configurations.

We saw that across many different domains, the dual functions are piecewise-structured—for example, they are piecewise-constant or -linear. This suggests a natural "empirical risk minimization" algorithm: uncover the analytical form of the dual functions across all of the problem instances in the training set and return the parameter setting that has optimal average empirical performance. Unfortunately, this "brute-force" approach can be computationally prohibitive, especially as the number of tunable parameters grows. It also does not take advantage of any additional structure exhibited by the dual functions, beyond their piecewise structure. What additional structure is there which we could use to devise faster learning algorithms? For example, each dual function may be volatile, but on average over samples, structure often emerges (for example, in Section 4.4 of Chapter 4, the average dual functions were often unimodal, though interestingly, not always Lipschitz continuous). Moreover, in Chapter 8, we saw that the dual functions can often be well-approximated by simple functions. Could these structures, or any others, be useful for designing configuration procedures with provable guarantees?

**Data-dependent guarantees.** Another direction which will help answer both statistical *and* algorithmic questions are *data-dependent* bounds. The majority of the guarantees in this thesis are worst case, in the sense that they hold for *any* distribution over problem instances, no matter how gnarly. In Chapter 8, we saw data-dependent guarantees based on the observation that dual functions can often be approximated by simple functions, but this is only one particular type of structure. How else can we get better data-dependent bounds? These bounds will also help us answer the algorithmic question of *how* to learn provably high-performing configurations since we will need fewer samples to verify the optimality or suboptimality of any configuration.

**Learning with limited data.** The batch learning model that we focused on in this thesis has been used to great success in practice when the learner has access to a training set of problem instances encountered in the past [e.g., Sandholm, 2013, Xu et al., 2008]. However, there may be scenarios where this batch learning approach is not the right model—for example, when a company is just starting out and they have only a single large problem instance to solve, rather than a training set. Is it possible to use a training set from a related domain to learn a good configuration? Mathematically, what does it mean for one domain of, say, integer programs to be related to another? How can we use ideas from the literature on domain adaptation, transfer learning, and transductive learning to help answer these questions in the context of combinatorial algorithm configuration? Another option is to learn on-the-fly, within the instance, which has been explored in prior research from an applied perspective [e.g., Khalil et al., 2016, Tang et al., 2020]. However, as-of-yet, we have limited understanding from a theoretical perspective of how to learn within an instance.

**Algorithm configuration meets learning-augmented algorithms.** Theoretical research on data-driven algorithm design has developed in parallel with research on learning-augmented algorithms, where worst-case algorithms are endowed with machine-learned predictors about the input distribution [e.g., Hsu et al., 2019, Lykouris and Vassilvitskii, 2018, Mitzenmacher, 2018, Purohit et al., 2018]. For example, a caching algorithm may be endowed with a machine-learned function that predicts the next time a particular element is going to appear, as in research by

Lykouris and Vassilvitskii [2018]. The goal is then to design an algorithm that improves over the worst case when the predictions are accurate, but never performs worse than the best-known worst-case algorithm, even when the predictions are terrible. What are the formal, higher-level connections between these two lines of research? Can these learning-augmented algorithm design problems ever be reframed as parameter configuration problems? Is there a general theory that connects these two domains?

**Configuration beyond algorithms and mechanisms.** This thesis focused on configuring the parameters of algorithms, the fundamental tool of computer science. However, tunable parameters abound beyond the context of algorithm design. We saw this in the realm of economic mechanism design, but it true also in other areas of engineering, in experimental design, and beyond. At their core, the ideas from this thesis are not bound to algorithm parameter optimization, so to what extent can they provide guidance in other domains? What breakthroughs can we achieve by taking a data-driven approach to optimizing other non-algorithmic parameters?

# Appendix A

# Omitted details about mechanism configuration (Chapter 5)

In this section, we connect the hyperplane structure we investigate in Section 5.1 to the structured prediction literature in machine learning [e.g., Collins, 2000], thus proving even stronger generalization bounds for item-pricing mechanisms under buyers with unit-demand and general valuations and answering an open question by Morgenstern and Roughgarden [2016]. Balcan et al. [2014] were the first to explore the connection between structured prediction and mechanism design, though in a different setting from us: they provided algorithms that make use of past data describing the purchases of a utility-maximizing agent to produce a hypothesis function that can accurately forecast the future behavior of the agent.

Morgenstern and Roughgarden [2016] used structured prediction to provide sample complexity guarantees for several "simple" mechanism classes. They observed that these classes have profit functions $u_{\boldsymbol{\rho}}(\boldsymbol{v})$ which are the composition of two simpler functions: A generalized allocation function $f_{\boldsymbol{\rho}}^{(1)} : \mathcal{X} \to \mathcal{Z}$ and a simplified profit function $f_{\boldsymbol{\rho}}^{(2)} : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ such that $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = f_{\boldsymbol{\rho}}^{(2)}\left(\boldsymbol{v}, f_{\boldsymbol{\rho}}^{(1)}(\boldsymbol{v})\right)$. For example, $\mathcal{Z}$ might be the set of allocations. In this case, we say that the set of profit functions $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ is $\left(\mathcal{F}^{(1)}, \mathcal{F}^{(2)}\right)$-decomposable, where $\mathcal{F}^{(1)} = \left\{f_{\boldsymbol{\rho}}^{(1)} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ and $\mathcal{F}^{(2)} = \left\{f_{\boldsymbol{\rho}}^{(2)} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$. See Example A.0.1 for an example of this decomposition.

**Example A.0.1** (Item-pricing mechanisms [Morgenstern and Roughgarden, 2016])**.** Let $\mathcal{U}$ be the set of profit functions corresponding to the class of anonymous item-pricing mechanisms over a single additive buyer and let $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_m)$ be a vector of prices. In this case, we can define $f_{\boldsymbol{\rho}}^{(1)} : \mathcal{X} \to \{0,1\}^m$ where the $i^{th}$ component of $f_{\boldsymbol{\rho}}^{(1)}(\boldsymbol{v})$ is 1 if and only if the buyer buys item $i$. Define $\psi(\boldsymbol{v}, \boldsymbol{\alpha}) = (v(\boldsymbol{\alpha}), -\boldsymbol{\alpha})$ and define $\boldsymbol{w}^{\boldsymbol{\rho}} = (1, \boldsymbol{\rho})$. Then the $\boldsymbol{\alpha}$ that maximizes $\langle \boldsymbol{w}^{\boldsymbol{\rho}}, \psi(\boldsymbol{v}, \boldsymbol{\alpha})\rangle$ is the $\boldsymbol{\alpha}$ that maximizes the buyer's utility, i.e., $f_{\boldsymbol{\rho}}^{(1)}(\boldsymbol{v})$, as desired. Finally, we define $f_{\boldsymbol{\rho}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \langle \boldsymbol{\alpha}, \boldsymbol{\rho}\rangle$, and we have that $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = f_{\boldsymbol{\rho}}^{(2)}\left(\boldsymbol{v}, f_{\boldsymbol{\rho}}^{(1)}(\boldsymbol{v})\right)$, as desired.

Morgenstern and Roughgarden [2016] bound $\text{Pdim}(\mathcal{U})$ using the intrinsic complexity of $\mathcal{F}^{(1)}$, which they quantified using tools from structured prediction, namely, *generalized linear functions.*

**Definition A.0.2** (*a*-dimensional linear class)**.** A set of functions $\mathcal{F} = \left\{f_{\boldsymbol{\rho}} : \mathcal{X} \to \mathcal{Z} \mid \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ is an *a-dimensional linear class* if there is a function $\psi : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}^a$ and a vector $\boldsymbol{w}^{\boldsymbol{\rho}} \in \mathbb{R}^a$ for each $\boldsymbol{\rho} \in \mathbb{R}^d$ such that $f_{\boldsymbol{\rho}}(\boldsymbol{v}) \in \operatorname{argmax}_{\boldsymbol{\alpha} \in \mathcal{Z}} \langle \boldsymbol{w}^{\boldsymbol{\rho}}, \psi(\boldsymbol{v}, \boldsymbol{\alpha})\rangle$ and $\left|\operatorname{argmax}_{\boldsymbol{\alpha} \in \mathcal{Z}} \langle \boldsymbol{w}^{\boldsymbol{\rho}}, \psi(\boldsymbol{v}, \boldsymbol{\alpha})\rangle\right| = 1$.

If $\mathcal{U}$ is $\left(\mathcal{F}^{(1)}, \mathcal{F}^{(2)}\right)$-decomposable and $\mathcal{F}^{(1)}$ is an $a$-dimensional linear class over $\mathcal{Z}$, we say that $\mathcal{U}$ is an $a$-dimensional linear class over $\mathcal{Z}$.

The bounds Morgenstern and Roughgarden [2016] provided using linear separability are loose in several settings: for anonymous and non-anonymous item-pricing mechanisms under additive buyers, their structured prediction approach gives a pseudo-dimension bound of $O\left(m^2\right)$ and $O\left(nm^2 \log m\right)$, respectively. They left as an open question whether linear separability can be used to prove tighter guarantees. Using the hyperplane structures we study in Section 5.1, we prove that the answer is "yes." We require the following refined notion of $(d, t)$-delineable classes.

**Definition A.0.3.** Suppose that $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ is $\left(\mathcal{F}^{(1)}, \mathcal{F}^{(2)}\right)$-decomposable. We say that $\mathcal{U}$ is $(d, \tau_1, \tau_2)$-*divisible* if:

1. For any $\boldsymbol{v} \in \mathcal{X}$, there is a set $\mathcal{H}$ of $\tau_1$ hyperplanes such that for any connected component $\mathcal{P}'$ of $\mathbb{R}^d \setminus \mathcal{H}$, the function $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{\rho})$ is constant over all $\boldsymbol{\rho} \in \mathcal{P}'$.
2. For any $\boldsymbol{v} \in \mathcal{X}$ and any $\boldsymbol{\alpha} \in \mathcal{Z}$, there is a set $\mathcal{H}_2$ of $\tau_2$ hyperplanes such that for any connected component $\mathcal{P}'$ of $\mathbb{R}^d \setminus \mathcal{H}_2$, the function $f_{\boldsymbol{v},\boldsymbol{\alpha}}^{(2)}(\boldsymbol{\rho})$ is linear over all $\boldsymbol{\rho} \in \mathcal{P}'$.

Note that $(d, \tau_1, \tau_2)$-divisibility implies $(d, \tau_1 + \tau_2)$-delineability. Theorem A.0.4 connects linear separability and divisibility with pseudo-dimension.

**Theorem A.0.4.** *Suppose $\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}$ is $(d, \tau_1, \tau_2)$-divisible with $\tau_1, \tau_2 \geq 1$ and an $a$-dimensional linear class over $\mathcal{Z}$. Let $\omega = \min\left\{|\mathcal{Z}|^a, d\left(a\tau_1\right)^d\right\}$. Then*

$$\operatorname{Pdim}(\mathcal{U}) = O\left((d+a)\log(d+a) + d\log\tau_2 + \log\omega\right).$$

*Proof.* To prove this theorem, we will use the following standard notation. For a class $\mathcal{F}$ of real-valued functions mapping $\mathcal{X}$ to $\mathbb{R}$, let $\mathcal{S} = \left\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}\right\}$ be a subset of $\mathcal{X}$. We define

$$\Pi_{\mathcal{F}}(\mathcal{S}) = \max_{t^{(1)},\ldots,t^{(N)} \in \mathbb{R}} \left|\left\{\begin{pmatrix} \mathbf{1}_{\left\{f\left(\boldsymbol{v}^{(1)}\right) \geq t^{(1)}\right\}} \\ \vdots \\ \mathbf{1}_{\left\{f\left(\boldsymbol{v}^{(N)}\right) \geq t^{(N)}\right\}} \end{pmatrix} : f \in \mathcal{F}\right\}\right|.$$

The pseudo-dimension of $\mathcal{F}$ is the size of the largest set $\mathcal{S}$ such that $\Pi_{\mathcal{F}}(\mathcal{S}) = 2^{|\mathcal{S}|}$. Given a set $\mathcal{S} = \left\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}\right\}$, we also use the notation $f(\mathcal{S})$ to denote the vector $\left(f(\boldsymbol{v}^{(1)}), \ldots, f(\boldsymbol{v}^{(N)})\right)$. Morgenstern and Roughgarden [2016] proved the following lemma.

**Lemma A.0.5** (Morgenstern and Roughgarden [2016])**.** *Suppose $\mathcal{U}$ is $\left(\mathcal{F}^{(1)}, \mathcal{F}^{(2)}\right)$-decomposable and an $a$-dimensional linear class. Let $\mathcal{S} = \left\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}\right\}$ be a subset of $\mathcal{X}$. Then*

$$\Pi_{\mathcal{U}}(\mathcal{S}) \leq \left|\left\{\left(\mathcal{S}', f_{\boldsymbol{\rho}}^{(1)}(\mathcal{S}')\right) : \mathcal{S}' \subseteq \mathcal{S}, |\mathcal{S}'| = a, \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \cdot \max_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(N)} \in \mathcal{Z}} \left\{\Pi_{\mathcal{F}^{(2)}}\left(\left\{\left(\boldsymbol{v}^{(1)}, \boldsymbol{\alpha}^{(1)}\right), \ldots, \left(\boldsymbol{v}^{(N)}, \boldsymbol{\alpha}^{(N)}\right)\right\}\right)\right\}.$$

Suppose $\text{Pdim}(\mathcal{U}) = N$. By definition, there exists a set $\mathcal{S} = \left\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)}\right\}$ that is shattered by $\mathcal{U}$. By Lemmas A.0.5 and A.0.6, this means that

$$2^N = \Pi_{\mathcal{U}}(\mathcal{S}) \leq N^a \omega \max_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(N)} \in \mathcal{Z}} \left\{\Pi_{\mathcal{F}^{(2)}}\left(\left\{\left(\boldsymbol{v}^{(1)}, \boldsymbol{\alpha}^{(1)}\right), \ldots, \left(\boldsymbol{v}^{(N)}, \boldsymbol{\alpha}^{(N)}\right)\right\}\right)\right\}.$$

To prove this theorem, we will show that

$$\max_{\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(N)} \in \mathcal{Z}} \left\{\Pi_{\mathcal{F}^{(2)}}\left(\left\{\left(\boldsymbol{v}^{(1)}, \boldsymbol{\alpha}^{(1)}\right), \ldots, \left(\boldsymbol{v}^{(N)}, \boldsymbol{\alpha}^{(N)}\right)\right\}\right)\right\} < d^2 \left(N^2 \tau_2\right)^d, \tag{A.1}$$

which means that $2^N < N^{2d+a} d^2 \tau_2^d \omega$, and thus $N = O\left((d+a)\log(d+a) + d\log\tau_2 + \log\omega\right)$.

To this end, let $\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(N)}$ be $N$ arbitrary elements of $\mathcal{Z}$ and let $t^{(1)}, \ldots, t^{(N)}$ be $N$ arbitrary elements of $\mathbb{R}$. Since $\mathcal{U}$ is $(d, \tau_1, \tau_2)$-divisible, we know that for each $i \in [N]$, there is a set $\mathcal{H}_2^{(i)}$ of $\tau_2$ hyperplanes such that for any connected component $\mathcal{P}'$ of $\mathbb{R}^d \setminus \mathcal{H}_2^{(i)}$, $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho})$ is linear over all $\boldsymbol{\rho} \in \mathcal{P}'$. We now consider the overlay of all $N$ partitions $\mathbb{R}^d \setminus \mathcal{H}_2^{(1)}, \ldots, \mathbb{R}^d \setminus \mathcal{H}_2^{(N)}$. Formally, this overlay is made up of the sets $\mathcal{P}_1, \ldots, \mathcal{P}_\tau$, which are the connected components of $\mathbb{R}^d \setminus \left(\bigcup_{i=1}^N \mathcal{H}_2^{(i)}\right)$. For each set $\mathcal{P}_j$ and each $i \in [N]$, $\mathcal{P}_j$ is completely contained in a single connected component of $\mathbb{R}^d \setminus \mathcal{H}_2^{(i)}$, which means that $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho})$ is linear over $\mathcal{P}_j$. Since $\left|\mathcal{H}_2^{(i)}\right| \leq \tau_2$ for all $i \in [N]$, $\tau < d(N\tau_2)^d$ [Buck, 1943].

Now, consider a single connected component $\mathcal{P}_j$ of $\mathbb{R}^d \setminus \left(\bigcup_{i=1}^N \mathcal{H}_2^{(i)}\right)$. For any sample $\boldsymbol{v}^{(i)} \in \mathcal{S}$, we know that $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho})$ is linear over $\mathcal{P}_j$. Let $\boldsymbol{a}_j^{(i)} \in \mathbb{R}^d$ and $b_j^{(i)} \in \mathbb{R}$ be the weight vector and offset such that $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho}) = \boldsymbol{a}_j^{(i)} \cdot \boldsymbol{\rho} + b_j^{(i)}$ for all $\boldsymbol{\rho} \in \mathcal{P}_j$. We know that there is a hyperplane $\boldsymbol{a}_j^{(i)} \cdot \boldsymbol{\rho} + b_j^{(i)} = t^{(i)}$ where on one side of the hyperplane, $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho}) \leq t^{(i)}$ and on the other side, $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho}) > t^{(i)}$. Let $\mathcal{H}_{\mathcal{P}_j}$ be all $N$ hyperplanes for all $N$ samples, i.e., $\mathcal{H}_{\mathcal{P}_j} = \left\{\boldsymbol{a}_j^{(i)} \cdot \boldsymbol{\rho} + b_j^{(i)} = t^{(i)} : i \in [N]\right\}$. Notice that in any connected component $\mathcal{P}'$ of $\mathcal{P}_j \setminus \mathcal{H}_{\mathcal{P}_j}$, for all $i \in [N]$, $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho})$ is either greater than $t^{(i)}$ or less than $t^{(i)}$ (but not both) for all $\boldsymbol{\rho} \in \mathcal{P}'$.

In total, the number of connected components of $\mathcal{P}_j \setminus \mathcal{H}_{\mathcal{P}_j}$ is smaller than $dN^d$. The same holds for every partition $\mathcal{P}_j$. Thus, the total number of regions where for all $i \in [N]$, $f^{(2)}_{\boldsymbol{v}^{(i)},\boldsymbol{\alpha}^{(i)}}(\boldsymbol{\rho})$ is either greater than $t^{(i)}$ or less than $t^{(i)}$ (but not both) is smaller than $dN^d \cdot d(N\tau_2)^d$. In other words,

$$\left|\left\{\begin{pmatrix} \mathbf{1}\left(f^{(2)}_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(1)}, \boldsymbol{\alpha}^{(1)}\right) \geq t^{(1)}\right) \\ \vdots \\ \mathbf{1}\left(f^{(2)}_{\boldsymbol{\rho}}\left(\boldsymbol{v}^{(N)}, \boldsymbol{\alpha}^{(N)}\right) \geq t^{(N)}\right) \end{pmatrix} : \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \leq dN^d \cdot d(N\tau_2)^d.$$

Since we chose $\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(N)}$ and $t^{(1)}, \ldots, t^{(N)}$ arbitrarily, we may conclude that Inequality (A.1) holds. □

**Lemma A.0.6.** *Suppose $\mathcal{U}$ is an $a$-dimensional linear class over $\mathcal{Z}$ and $(d, \tau_1, \tau_2)$-divisible. Then for any set $\mathcal{S} \subseteq \mathcal{X}$ of size $N$,*

$$\left|\left\{(\mathcal{S}', f^{(1)}_{\boldsymbol{\rho}}(\mathcal{S}')) : \mathcal{S}' \subseteq \mathcal{S}, |\mathcal{S}'| = a, \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \leq N^a \min\left\{|\mathcal{Z}|^a, d(a\tau_1)^d\right\}.$$

*Proof.* To begin with, there are of course at most $N^a$ ways to choose a set $\mathcal{S}' \subseteq \mathcal{S}$ of size $a$. How many ways are there to label a fixed set $\mathcal{S}' = \left\{\boldsymbol{v}^{(i_1)}, \ldots, \boldsymbol{v}^{(i_a)}\right\}$ of size $a$ using functions from $\mathcal{F}^{(1)}$? An easy upper bound is $|\mathcal{Z}|^a$. Alternatively, we can use the structure of $\mathcal{U}$ to prove that there are $d(a\tau_1)^d$ ways to label $\mathcal{S}'$. Since $\mathcal{U}$ is $(d, \tau_1, \tau_2)$-divisible, we know that for any $\boldsymbol{v}^{(i_j)} \in \mathcal{S}'$, there is a set $\mathcal{H}_1^{(i_j)}$ of $\tau_1$ hyperplanes such that for any connected component $\mathcal{P}'$ of $\mathbb{R}^d \setminus \mathcal{H}_1^{(i_j)}$, $f^{(1)}_{\boldsymbol{v}^{(i_j)}}(\boldsymbol{\rho})$ is constant over all $\boldsymbol{\rho} \in \mathcal{P}'$. We now consider the overlay of all $a$ partitions $\mathbb{R}^d \setminus \mathcal{H}_1^{(i_j)}$ for all $\boldsymbol{v}^{(i_j)} \in \mathcal{S}'$. Formally, this overlay is made up of the sets $\mathcal{P}_1, \ldots, \mathcal{P}_\tau$, which are the connected components of $\mathbb{R}^d \setminus \left(\bigcup_{\boldsymbol{v}^{(i_j)} \in \mathcal{S}'} \mathcal{H}_1^{(i_j)}\right)$. For each set $\mathcal{P}_t$ and each $\boldsymbol{v}^{(i_j)} \in \mathcal{S}'$, $\mathcal{P}_t$ is completely contained in a single connected component of $\mathbb{R}^d \setminus \mathcal{H}_1^{(i_j)}$, which means that $f^{(1)}_{\boldsymbol{v}^{(i_j)}}(\boldsymbol{\rho})$ is constant over $\mathcal{P}_t$. This means that the number of ways to label $\mathcal{S}'$ is at most $\tau$. Since $\left|\mathcal{H}_1^{(i_j)}\right| \leq \tau_1$ for all $\boldsymbol{v}^{(i_j)} \in \mathcal{S}'$, $\tau < d(a\tau_1)^d$ [Buck, 1943]. Therefore, $\left|\left\{\left(\mathcal{S}', f^{(1)}_{\boldsymbol{\rho}}(\mathcal{S}')\right) : \mathcal{S}' \subseteq \mathcal{S}, |\mathcal{S}'| = a, \boldsymbol{\rho} \in \mathbb{R}^d\right\}\right| \leq N^a \min\left\{|\mathcal{Z}|^a, d(a\tau_t)^d\right\}$, so the lemma statement holds. □

## Divisible mechanism classes

We now instantiate Theorem A.0.4.

**Lemma A.0.7.** *Let $\mathcal{U}$ and $\mathcal{U}'$ be sets of profit functions corresponding to the classes of item-pricing mechanisms with anonymous prices and non-anonymous prices. If the buyers are unit-demand, then $\mathcal{U}$ is $(m, nm^2, 1)$-divisible and $\mathcal{U}'$ is $(nm, nm^2, 1)$-divisible. Also, $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$. Therefore,*

$$\operatorname{Pdim}(\mathcal{U}) = O\left(\min\left\{m^2, m\log(nm)\right\}\right) \text{ and } \operatorname{Pdim}\left(\mathcal{U}'\right) = O\left(nm\log(nm)\right).$$

*Proof.* We begin with anonymous reserves. Let $f^{(1)}_{\boldsymbol{p}} : \mathcal{X} \to \{0,1\}^m$ be defined so that the $i^{th}$ component is 1 if and only if item $i$ is sold. For each buyer $j$, there are $\binom{m}{2}$ hyperplanes defining their preference ordering on the items: $v_j(\boldsymbol{e}_i) - p(\boldsymbol{e}_i) = v_j(\boldsymbol{e}_k) - p(\boldsymbol{e}_k)$ for all $i \neq k$. This gives a total of at most $\tau_1 = nm^2$ hyperplanes splitting $\mathbb{R}^m$ into regions where $f^{(1)}_{\boldsymbol{v}}(\boldsymbol{p})$ is constant. Next, we can write $f^{(2)}_{\boldsymbol{p}}(\boldsymbol{v}, \boldsymbol{\alpha}) = \boldsymbol{\alpha} \cdot \boldsymbol{p}$, which is always linear, so we may set $\tau_2 = 1$.

Under non-anonymous reserve prices, let $f^{(1)}_{\boldsymbol{p}} : \mathcal{X} \to \{0,1\}^{nm}$ be defined so that for every buyer $j$ and every item $i$, there is a component of $f^{(1)}_{\boldsymbol{p}}(\boldsymbol{v})$ that is 1 if and only if buyer $j$ receives item $i$. As with anonymous prices, there are $\tau_1 = nm^2$ hyperplanes splitting $\mathbb{R}^{nm}$ into regions where $f^{(1)}_{\boldsymbol{v}}(\boldsymbol{p})$ is constant. Next, we can write $f^{(2)}_{\boldsymbol{p}}(\boldsymbol{v}, \boldsymbol{\alpha}) = \boldsymbol{\alpha} \cdot \boldsymbol{p}$, which is always linear, so we may set $\tau_2 = 1$.

Morgenstern and Roughgarden [2016] proved that $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$, respectively. □

When prices are anonymous, if $n < 2^m$, Lemma A.0.7 improves on the pseudo-dimension bound of $O\left(m^2\right)$ Morgenstern and Roughgarden [2016] gave for this class, and otherwise it matches their bound. When the prices are non-anonymous our bound improves on their bound of $O\left(nm^2 \log n\right)$.

**Lemma A.0.8.** *Let $\mathcal{U}$ and $\mathcal{U}'$ be the sets of profit functions corresponding to the classes of item-pricing mechanisms with anonymous prices and non-anonymous prices, respectively. If the buyers have general values, then $\mathcal{U}$ is $\left(m, n2^{2m}, 1\right)$-divisible and $\mathcal{U}'$ is $\left(nm, n2^{2m}, 1\right)$-divisible. Also, $\mathcal{U}$ is $(m+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $\mathcal{U}'$ is $(nm+1)$-dimensionally linearly separable over $[n]^m$. Thus, $Pdim\left(\mathcal{U}\right) = O\left(m^2\right)$ and $Pdim\left(\mathcal{U}'\right) = O\left(nm\left(m + \log n\right)\right)$.*

*Proof.* We begin with anonymous reserves. Let $f_{\boldsymbol{p}}^{(1)} : \mathcal{X} \to \{0,1\}^m$ be defined so that the $i^{th}$ component is 1 if and only if item $i$ is sold. For each buyer $j$, there are $\binom{2^m}{2}$ hyperplanes defining their preference ordering on the bundles: $v_j(\boldsymbol{q}) - \sum_{i:q[i]=1} p(\boldsymbol{e}_i) = v_j(\boldsymbol{q}') - \sum_{i:q'[i]=1} p(\boldsymbol{e}_i)$ for all $\boldsymbol{q}, \boldsymbol{q}' \in \{0,1\}^m$. This gives a total of at most $\tau_1 = n2^{2m}$ hyperplanes splitting $\mathbb{R}^m$ into regions where $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is constant. Next, we can write $f_{\boldsymbol{p}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \boldsymbol{\alpha} \cdot \boldsymbol{p}$, which is always linear, so we may set $\tau_2 = 1$.

Under non-anonymous reserve prices, let $f_{\boldsymbol{p}}^{(1)} : \mathcal{X} \to \{0,1\}^{nm}$ be defined so that for every buyer $j$ and every item $i$, there is a component of $f_{\boldsymbol{p}}^{(1)}(\boldsymbol{v})$ that is 1 if and only if buyer $j$ receives item $i$. As with anonymous prices, there are $\tau_1 = n2^{2m}$ hyperplanes splitting $\mathbb{R}^{nm}$ into regions where $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is constant. Next, we can write $f_{\boldsymbol{p}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \boldsymbol{\alpha} \cdot \boldsymbol{p}$, which is always linear, so we may set $\tau_2 = 1$.

Morgenstern and Roughgarden [2016] proved that $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$, respectively. □

When there are anonymous prices, the number of hyperplanes in the partition is large, so considering the hyperplane partition does not help us. As a result, Lemma A.0.8 implies the same bound Morgenstern and Roughgarden [2016] gave. In the case of non-anonymous prices, analyzing the hyperplane partition gives a better bound than their bound of $O\left(nm^2 \log n\right)$.

In Lemma A.0.9, we use Theorem A.0.4 to prove pseudo-dimension bounds of $O\left(m \log m\right)$ and $O\left(nm \log nm\right)$ for the classes of second price auctions for additive buyers with anonymous and non-anonymous reserves, respectively. In Lemma A.0.10, we prove the same for item-pricing mechanisms. We thus answer the open question by Morgenstern and Roughgarden [2016]. These bounds match those implied by Lemmas 5.1.16 and 5.1.17.

**Lemma A.0.9.** *Let $\mathcal{U}$ and $\mathcal{U}'$ be the sets of profit functions corresponding to the classes of anonymous and non-anonymous second price item auctions. Then $\mathcal{U}$ is $(m, m, m)$-divisible and $\mathcal{U}'$ is $(nm, m, m)$-divisible. Also, $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$. Therefore, $Pdim(\mathcal{U}) = O(m \log m)$ and $Pdim(\mathcal{U}') = O(nm \log(nm))$.*

*Proof.* We begin with anonymous reserves. For a given valuation vector $\boldsymbol{v}$, let $j_i$ be the highest buyer for item $i$ and let $j_i'$ be the second highest buyer. Let $f_{\boldsymbol{p}}^{(1)} : \mathcal{X} \to \{0,1\}^m$ be defined so that the $i^{th}$ component is 1 if and only if item $i$ is sold. There are $\tau_1 = m$ hyperplanes splitting $\mathbb{R}^m$ into regions where $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is constant: the $i^{th}$ component of $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is 1 if and only if $v_{j_i}(\boldsymbol{e}_i) \geq p(\boldsymbol{e}_i)$. Next, we can write $f_{\boldsymbol{p}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \sum_{i:\alpha[i]=1} \max\left\{v_{j_i'}(\boldsymbol{e}_i), p(\boldsymbol{e}_i)\right\} - c(\boldsymbol{\alpha})$, which is linear so long as either $v_{j_i'}(\boldsymbol{e}_i) < p(\boldsymbol{e}_i)$ or $v_{j_i'}(\boldsymbol{e}_i) \geq p(\boldsymbol{e}_i)$ for all $i \in [m]$. Therefore, there are $\tau_2 = m$ hyperplanes $\mathcal{H}_2$ such that for any connected component $\mathcal{P}'$ of $\mathcal{P} \setminus \mathcal{H}_2$, $f_{\boldsymbol{v},\boldsymbol{\alpha}}^{(2)}(\boldsymbol{p})$ is linear over all $\boldsymbol{p} \in \mathcal{P}'$.

Under non-anonymous reserve prices, let $f_{\boldsymbol{p}}^{(1)} : \mathcal{X} \to \{0,1\}^{nm}$ be defined so that for every buyer $j$ and every item $i$, there is a component of $f_{\boldsymbol{p}}^{(1)}(\boldsymbol{v})$ that is 1 if and only if buyer $j$ receives

item $i$. There are $\tau_1 = m$ hyperplanes splitting $\mathbb{R}^{nm}$ into regions where $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is constant: for every item $i$, the component corresponding to buyer $j_i$ is 1 if and only if $v_{j_i}(\boldsymbol{e}_i) \geq p_j(\boldsymbol{e}_i)$. Next, we can write $f_{\boldsymbol{p}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \sum_{i:\alpha[i]=1} \max\left\{v_{j_i'}(\boldsymbol{e}_i), p_{j_i}(\boldsymbol{e}_i)\right\} - c(\boldsymbol{\alpha})$, which is linear so long as either $v_{j_i'}(\boldsymbol{e}_i) < p_{j_i}(\boldsymbol{e}_i)$ or $v_{j_i'}(\boldsymbol{e}_i) \geq p_{j_i}(\boldsymbol{e}_i)$ for all $i \in [m]$. Therefore, there are $\tau_2 = m$ hyperplanes $\mathcal{H}_2$ such that for any connected component $\mathcal{P}'$ of $\mathcal{P} \setminus \mathcal{H}_2$, $f_{\boldsymbol{v},\boldsymbol{\alpha}}^{(2)}(\boldsymbol{p})$ is linear over all $\boldsymbol{p} \in \mathcal{P}'$.

Morgenstern and Roughgarden [2016] proved that $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$, respectively. □

**Lemma A.0.10.** *Let $\mathcal{U}$ and $\mathcal{U}'$ be the sets of profit functions corresponding to the classes of item-pricing mechanisms with anonymous prices and non-anonymous prices, respectively. If the buyers are additive, then $\mathcal{U}$ is $(m, m, 1)$-divisible and $\mathcal{U}'$ is $(nm, nm, 1)$-divisible. Also, $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$. Therefore, $Pdim(\mathcal{U}) = O(m \log m)$ and $Pdim(\mathcal{U}') = O(nm \log(nm))$.*

*Proof.* We begin with anonymous reserves. For a given valuation vector $\boldsymbol{v}$, let $j_i$ be the buyer with the highest valuation for item $i$. Let $f_{\boldsymbol{p}}^{(1)} : \mathcal{X} \to \{0,1\}^m$ be defined so that the $i^{th}$ component is 1 if and only if item $i$ is sold. There are $\tau_1 = m$ hyperplanes splitting $\mathbb{R}^m$ into regions where $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is constant: the $i^{th}$ component of $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is 1 if and only if $v_{j_i}(\boldsymbol{e}_i) \geq p(\boldsymbol{e}_i)$. Next, we can write $f_{\boldsymbol{p}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \boldsymbol{\alpha} \cdot \boldsymbol{p}$, which is always linear, so we may set $\tau_2 = 1$.

Under non-anonymous reserve prices, let $f_{\boldsymbol{p}}^{(1)} : \mathcal{X} \to \{0,1\}^{nm}$ be defined so that for every buyer $j$ and every item $i$, there is a component of $f_{\boldsymbol{p}}^{(1)}(\boldsymbol{v})$ that is 1 if and only if buyer $j$ receives item $i$. There are $\tau_1 = nm$ hyperplanes splitting $\mathbb{R}^{nm}$ into regions where $f_{\boldsymbol{v}}^{(1)}(\boldsymbol{p})$ is constant: $v_j(\boldsymbol{e}_i) = p_j(\boldsymbol{e}_i)$ for all $i$ and all $j$. Next, we can write $f_{\boldsymbol{p}}^{(2)}(\boldsymbol{v}, \boldsymbol{\alpha}) = \boldsymbol{\alpha} \cdot \boldsymbol{p}$, which is always linear, so we may set $\tau_2 = 1$.

Morgenstern and Roughgarden [2016] proved that $\mathcal{U}$ and $\mathcal{U}'$ are $(m+1)$- and $(nm+1)$-dimensionally linearly separable over $\{0,1\}^m$ and $[n]^m$, respectively. □

# Appendix B

# Omitted details about frugal training with generalization guarantees (Chapter 11)

**Lemma B.0.1.** *Suppose Algorithm 8 has a $\zeta$-representative run. For any set $\mathcal{P}' \in \bar{\mathcal{G}}$ and all parameters $\boldsymbol{\rho} \in \mathcal{P}'$, $t_{\delta/2}(\boldsymbol{\rho}) \leq \tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$.*

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

**Claim B.0.2.** *Suppose Algorithm 8 has a $\zeta$-representative run. For any set $\mathcal{P}' \in \mathcal{G}$ and all parameters $\boldsymbol{\rho} \in \mathcal{P}'$, $t_{\delta/2}(\boldsymbol{\rho}) \leq \tau_{\mathcal{P}'}$.*

*Proof of Claim B.0.2.* Let $t$ be the round that $\mathcal{P}'$ is added to $\mathcal{G}$. We know that for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}'$,

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

Since Algorithm 8 had a $\zeta$-representative run, we know that

$$\begin{aligned}
&\Pr_{z \sim \mathcal{D}} \left[\ell(\boldsymbol{\rho}, z) \leq \tau_{\mathcal{P}'}\right] \\
\geq \; & \frac{1}{|\mathcal{S}_t|} \sum_{z \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, z) \leq \tau_{\mathcal{P}'}\}} - \sqrt{\frac{2d \ln |\textsc{Partition}\left(\mathcal{S}_t, \tau_{\mathcal{P}'}\right)|}{|\mathcal{S}_t|}} - \sqrt{\frac{8}{|\mathcal{S}_t|} \ln \frac{8 \left(\tau_{\mathcal{P}'} \, |\mathcal{S}_t| \, t\right)^2}{\zeta}} \\
\geq \; & \frac{1}{|\mathcal{S}_t|} \sum_{z \in \mathcal{S}_t} \mathbf{1}_{\{\ell(\boldsymbol{\rho}, z) \leq \tau_{\mathcal{P}'}\}} - \sqrt{\frac{2d \ln |\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)|}{|\mathcal{S}_t|}} - \sqrt{\frac{8}{|\mathcal{S}_t|} \ln \frac{8 \left(2^t \, |\mathcal{S}_t| \, t\right)^2}{\zeta}},
\end{aligned}$$

where the second inequality follows from the fact that $\tau_{\mathcal{P}'} \leq 2^t$ and monotonicity. Based on Step 4 of Algorithm 8, we know that

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

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

$$\Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho},z) > \tau_{\mathcal{P}'}] = \Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho},z) \ge \tau_{\mathcal{P}'} + 1] < \frac{\delta}{2}. \tag{B.2}$$

We claim that Equation (B.2) implies that $t_{\delta/2}(\boldsymbol{\rho}) \le \tau_{\mathcal{P}'}$. For a contradiction, suppose $t_{\delta/2}(\boldsymbol{\rho}) > \tau_{\mathcal{P}'}$, or in other words, $t_{\delta/2}(\boldsymbol{\rho}) \ge \tau_{\mathcal{P}'} + 1$. Since

$$t_{\delta/2}(\boldsymbol{\rho}) = \text{argmax}_{\tau\in\mathbb{Z}}\left\{\Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho},z) \ge \tau] \ge \delta/2\right\},$$

this would mean that $\delta/2 \le \Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho},z) \ge t_{\delta/2}(\boldsymbol{\rho})] \le \Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho},z) \ge \tau_{\mathcal{P}'} + 1] < \delta/2$, which is a contradiction. Therefore, the claim holds. □

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

**Claim B.0.3.** *Suppose Algorithm 8 has a $\zeta$-representative run. For any set $\mathcal{P}' \in \mathcal{G}$ and all parameters $\boldsymbol{\rho} \in \mathcal{P}'$, $\tau_{\mathcal{P}'} \le t_{\delta/4}(\boldsymbol{\rho})$.*

*Proof of Claim B.0.3.* Let $t$ be the round that $\mathcal{P}'$ is added to $\mathcal{G}$. We know that for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}'$,

$$\begin{aligned}
1 - \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\mathbf{1}_{\{\ell(\boldsymbol{\rho},z)\le\tau_{\mathcal{P}'}-1\}} &= 1 - \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\mathbf{1}_{\{\ell(\boldsymbol{\rho},z)<\tau_{\mathcal{P}'}\}} \\
&= \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\left(1 - \mathbf{1}_{\{\ell(\boldsymbol{\rho},z)<\tau_{\mathcal{P}'}\}}\right) \\
&= \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\mathbf{1}_{\{\ell(\boldsymbol{\rho},z)\ge\tau_{\mathcal{P}'}\}} \\
&\ge \frac{|\mathcal{S}_t| - \lfloor|\mathcal{S}_t|\,(1-3\delta/8)\rfloor}{|\mathcal{S}_t|} \\
&\ge \frac{3\delta}{8}.
\end{aligned}$$

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

$$\begin{aligned}
&\Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho},z) \le \tau_{\mathcal{P}'} - 1] \\
\le\ & \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\mathbf{1}_{\{\ell(\boldsymbol{\rho},z)\le\tau_{\mathcal{P}'}-1\}} + \sqrt{\frac{2d\ln|\textsc{Partition}\,(\mathcal{S}_t,\tau_{\mathcal{P}'}-1)|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left((\tau_{\mathcal{P}'}-1)\,|\mathcal{S}_t|\,t\right)^2}{\zeta}} \\
\le\ & 1 - \frac{3\delta}{8} + \sqrt{\frac{2d\ln|\textsc{Partition}\,(\mathcal{S}_t,\tau_{\mathcal{P}'}-1)|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left((\tau_{\mathcal{P}'}-1)\,|\mathcal{S}_t|\,t\right)^2}{\zeta}} \\
<\ & 1 - \frac{3\delta}{8} + \sqrt{\frac{2d\ln|\textsc{Partition}\,(\mathcal{S}_t,2^t)|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(2^t\,|\mathcal{S}_t|\,t\right)^2}{\zeta}}
\end{aligned}$$

because $\tau_{\mathcal{P}'} - 1 < \tau_{\mathcal{P}'} \leq 2^t$ and monotonicity. Based on Step 4 of Algorithm 8,

$$\Pr_{z\sim\mathcal{D}}\left[\ell(\boldsymbol{\rho}, z) \leq \tau_{\mathcal{P}'} - 1\right] < 1 - \frac{3\delta}{8} + \eta\delta.$$

Since $\eta \leq 1/9$, we have that $\Pr_{z\sim\mathcal{D}}\left[\ell(\boldsymbol{\rho}, z) \leq \tau_{\mathcal{P}'} - 1\right] < 1 - \delta/4$. Thus, $\Pr_{z\sim\mathcal{D}}\left[\ell(\boldsymbol{\rho}, z) \geq \tau_{\mathcal{P}'}\right] = \Pr_{z\sim\mathcal{D}}\left[\ell(\boldsymbol{\rho}, z) > \tau_{\mathcal{P}'} - 1\right] > \delta/4$. Since $t_{\delta/4}(\boldsymbol{\rho}) = \operatorname{argmax}_{\tau\in\mathbb{Z}}\left\{\Pr_{z\sim\mathcal{D}}[\ell(\boldsymbol{\rho}, z) \geq \tau] \geq \delta/4\right\}$, we have that $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$. □

The lemma statement follows from Claims B.0.2 and B.0.3. □

**Corollary B.0.4.** *Suppose Algorithm 8 has a $\zeta$-representative run. For every set $\mathcal{P}' \in \bar{\mathcal{G}}$ and any parameter vector $\boldsymbol{\rho} \in \mathcal{P}'$, $\tau_{\mathcal{P}'}\delta/4 \leq \mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), \tau_{\mathcal{P}'}\right\}\right]$.*

*Proof.* By Lemma B.0.1, we know that $\tau_{\mathcal{P}'} \leq t_{\delta/4}(\boldsymbol{\rho})$, so $\Pr_{z\sim\mathcal{D}}\left[\ell(\boldsymbol{\rho}, z) \geq \tau_{\mathcal{P}'}\right] \geq \frac{\delta}{4}$. Therefore, $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), \tau_{\mathcal{P}'}\right\}\right] \geq \tau_{\mathcal{P}'}\Pr_{z\sim\mathcal{D}}\left[\ell(\boldsymbol{\rho}, z) \geq \tau_{\mathcal{P}'}\right] \geq \frac{\tau_{\mathcal{P}'}\delta}{4}$. □

**Lemma B.0.5.** *Suppose Algorithm 8 has a $\zeta$-representative run. For any parameter vector $\boldsymbol{\rho} \notin \bigcup_{\mathcal{P}'\in\bar{\mathcal{G}}}\mathcal{P}'$, $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq \bar{T}$.*

*Proof.* From Lemma 11.3.5, we know that $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq 2^{\bar{t}-3}\delta$. Moreover, from Step 2 of Algorithm 8, $\bar{T} \leq 2^{\bar{t}-3}\delta$, so $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), t_{\delta/4}(\boldsymbol{\rho})\right\}\right] \geq \bar{T}$. □

**Lemma B.0.6.** *Suppose Algorithm 8 has a $\zeta$-representative run. There exists a set $\mathcal{P}' \in \bar{\mathcal{G}}$ and a parameter vector $\boldsymbol{\rho} \in \mathcal{P}'$ such that $\bar{T} \geq \frac{1}{\sqrt[4]{1+\epsilon}} \cdot \mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), \tau_{\mathcal{P}'}\right\}\right]$.*

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

$$\begin{aligned}\bar{T} &= \frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\min\left\{\ell\left(\boldsymbol{\rho}, z\right), \tau_{\mathcal{P}'}\right\} \\ &\geq \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}, z\right), \tau_{\mathcal{P}'}\right\}\right] - \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}_t, \tau_{\mathcal{P}'}\right)\right|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(\tau_{\mathcal{P}'}\left|\mathcal{S}_t\right|t\right)^2}{\zeta}}\right).\end{aligned}$$

By Step 6, we know that at least a $(1 - 3\delta/8)$-fraction of the problem instances $z \in \mathcal{S}_t$ have a loss $\ell(\boldsymbol{\rho}, z)$ that is at most $2^t$. Therefore, by definition of $\tau_{\mathcal{P}'} = \tau_{\lfloor|\mathcal{S}_t|(1-3\delta/8)\rfloor}$, it must be that $\tau_{\mathcal{P}'} \leq 2^t$. By monotonicity, this means that

$$\bar{T} \geq \mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}, z\right), \tau_{\mathcal{P}'}\right\}\right] - \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)\right|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(2^t\left|\mathcal{S}_t\right|t\right)^2}{\zeta}}\right).$$

Based on Step 4 of Algorithm 8,

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

From Corollary B.0.4, $\tau_{\mathcal{P}'}\delta/4 \leq \mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), \tau_{\mathcal{P}'}\right\}\right]$, which means that

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

Finally, the lemma statement follows from the fact that

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

□

**Lemma B.0.7.** *Suppose Algorithm 8 has a $\zeta$-representative run. For every set $\mathcal{P}' \in \bar{\mathcal{G}}$ and every pair of parameter vectors $\boldsymbol{\rho}_1, \boldsymbol{\rho}_2 \in \mathcal{P}'$, $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\}\right] \leq \sqrt[4]{1+\epsilon} \cdot \mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_2, z\right), \tau_{\mathcal{P}'}\right\}\right]$.*

*Proof.* Let $t$ be the round that the interval $\mathcal{P}'$ was added to $\mathcal{G}$. Since Algorithm 8 had a $\zeta$-representative run,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\}\right] \\
\leq\ &\frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\} + \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}_t, \tau_{\mathcal{P}'}\right)\right|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(\tau_{\mathcal{P}'}\left|\mathcal{S}_t\right| t\right)^2}{\zeta}}\right).
\end{aligned}$$

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

$$\begin{aligned}
&\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\}\right] \\
\leq\ &\frac{1}{|\mathcal{S}_t|}\sum_{z\in\mathcal{S}_t}\min\left\{\ell\left(\boldsymbol{\rho}_2, z\right), \tau_{\mathcal{P}'}\right\} + \tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}_t, \tau_{\mathcal{P}'}\right)\right|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(\tau_{\mathcal{P}'}\left|\mathcal{S}_t\right| t\right)^2}{\zeta}}\right).
\end{aligned}$$

Again, since Algorithm 8 had a $\zeta$-representative run,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\}\right] \\
\leq\ &\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_2, z\right), \tau_{\mathcal{P}'}\right\}\right] + 2\tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}_t, \tau_{\mathcal{P}'}\right)\right|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(\tau_{\mathcal{P}'}\left|\mathcal{S}_t\right| t\right)^2}{\zeta}}\right).
\end{aligned}$$

Since $\tau_{\mathcal{P}'} \leq 2^t$,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\}\right] \\
\leq\ &\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_2, z\right), \tau_{\mathcal{P}'}\right\}\right] + 2\tau_{\mathcal{P}'}\left(\sqrt{\frac{2d\ln\left|\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)\right|}{|\mathcal{S}_t|}} + 2\sqrt{\frac{2}{|\mathcal{S}_t|}\ln\frac{8\left(2^t\left|\mathcal{S}_t\right| t\right)^2}{\zeta}}\right).
\end{aligned}$$

Based on Step 4 of Algorithm 8, this means that

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

By Corollary B.0.4, $\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_1, z\right), \tau_{\mathcal{P}'}\right\}\right] \leq (1+8\eta)\,\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}_2, z\right), \tau_{\mathcal{P}'}\right\}\right]$. The lemma statement follows from the fact that $\eta \leq \left(\sqrt[4]{1+\epsilon}-1\right)/8$. □

**Lemma B.0.8.** *Let $\mathcal{P}^*$ be the set of parameters output by Algorithm 8. Suppose that for every $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho}' \in \mathcal{P}^*$ such that*

$$\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}', z), t_{\delta/2}(\boldsymbol{\rho}')\right\}\right] \leq \sqrt{1+\epsilon}\left(OPT_{\delta/4} + \gamma\right).$$

*Then* $\min_{\boldsymbol{\rho}\in\mathcal{P}^*}\left\{\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell(\boldsymbol{\rho}, z), t_{\delta/2}(\boldsymbol{\rho})\right\}\right]\right\} \leq \sqrt{1+\epsilon}\cdot OPT_{\delta/4}$.

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

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

which is a contradiction. □

**Lemma B.0.9.** *Suppose that for all $\gamma > 0$, there exists a parameter vector $\boldsymbol{\rho}^* \in \mathcal{P}$ such that $2^{\bar{t}} \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot\mathbb{E}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, z\right), t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right] \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\left(OPT_{\delta/4} + \gamma\right)$. Then $2^{\bar{t}} \leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot OPT_{\delta/4}$.*

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

$$\begin{aligned}
2^{\bar{t}} &\leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot\mathop{\mathbb{E}}_{z\sim\mathcal{D}}\left[\min\left\{\ell\left(\boldsymbol{\rho}^*, z\right), t_{\delta/4}\left(\boldsymbol{\rho}^*\right)\right\}\right] \\
&\leq \frac{16\sqrt[4]{1+\epsilon}}{\delta}\left(OPT_{\delta/4} + \frac{\gamma'\delta}{32\sqrt[4]{1+\epsilon}}\right) \\
&= \frac{16\sqrt[4]{1+\epsilon}}{\delta}\cdot OPT_{\delta/4} + \frac{\gamma'}{2} \\
&= 2^{\bar{t}} - \frac{\gamma'}{2} \\
&< 2^{\bar{t}},
\end{aligned}$$

which is a contradiction. Therefore, the lemma statement holds. □

**Lemma B.0.10.** *Suppose Algorithm 8 has a $\zeta$-representative run. The size of the set $\mathcal{P}^* \subset \mathcal{P}$ that Algorithm 8 returns is bounded by $\sum_{t=1}^{\bar{t}}\left|\textsc{Partition}\left(\mathcal{S}_t, \frac{16}{\delta}\sqrt[4]{1+\epsilon}\cdot OPT_{\delta/4}\right)\right|$.*

*Proof.* Based on Step 12 of Algorithm 8, the size of $\mathcal{P}^*$ equals the size of the set $\bar{\mathcal{G}}$. Algorithm 8 only adds sets to $\mathcal{G}$ on Step 7, and on each round $t$, the number of sets it adds

is bounded by $\left|\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)\right|$. By monotonicity, we know that $\left|\textsc{Partition}\left(\mathcal{S}_t, 2^t\right)\right| \leq \left|\textsc{Partition}\left(\mathcal{S}_t, 2^{\bar{t}}\right)\right| \leq \left|\textsc{Partition}\left(\mathcal{S}_t, \frac{16}{\delta}\sqrt[4]{1+\epsilon} \cdot OPT_{\delta/4}\right)\right|$. Therefore,

$$\left|\bar{\mathcal{G}}\right| = \left|\mathcal{P}^*\right| \leq \sum_{t=1}^{\bar{t}} \left|\textsc{Partition}\left(\mathcal{S}_t, \frac{16}{\delta}\sqrt[4]{1+\epsilon} \cdot OPT_{\delta/4}\right)\right|.$$

□

# Appendix C

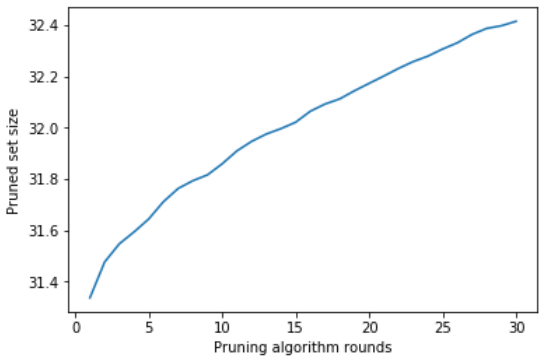
# Omitted details about learning to prune (Chapter 12)

Figure C.1 illustrates the average size of the pruned set $\bar{S}_i$ in Algorithm 9, which we ran a total of 5000 times, with $T = 30$ rounds each run. Figure C.1a corresponds to shortest-path routing and Figure C.1b corresponds to linear programming, with the same setup as described in Section 12.4.

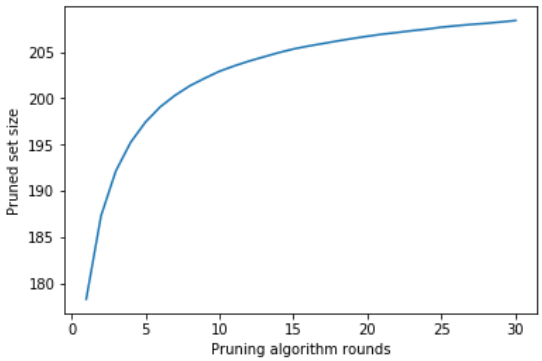
In Figure C.2, we present experiments using different perturbation methods. Figures C.2a and C.2b have the same experimental setup as in Section 12.4 except we employ a Gaussian distribution with smaller variance. We run our algorithm for thirty rounds ($T = 30$) with $p_i = 1/\sqrt{i}$. Each round, 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 $\boldsymbol{x} \in \mathbb{R}^{|E|}$ 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 $1/2$. We define a new edge-weight vector $\boldsymbol{x}_i$ such that $x_i[j] = \mathbb{I}_{\{x[j]+r_i[j]>0\}} \left(x[j] + r_i[j]\right)$. Our algorithm returned the incorrect path on a 0.034 fraction of the rounds.

In the remaining panels of Figure C.2, we use the uniform distribution rather than the Gaussian distribution. In Figures C.2c and C.2d, we run our algorithm for thirty 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 $\boldsymbol{x} \in \mathbb{R}^{|E|}$ be a vector representing all edges' weights. Let $w_i = \min\{x[i], 1/2\}$. On the $i^{th}$ round, we select a vector $\boldsymbol{r}_i \in \mathbb{R}^{|E|}$ such that each component is drawn from the uniform distribution over $[-w_i, w_i]$. We then define a new edge-weight vector $\boldsymbol{x}_i$ such that $x_i[j] = x[j] + r_i[j]$. Our algorithm returned the incorrect path on a 0.003 fraction of the $5000 \cdot T = 150,000$ rounds.

In Figures C.2e and C.2f, we use the same procedure except $w_i = \min\{x[i], 1\}$. In that setting, our algorithm returned the incorrect path on a 0.001 fraction of the $5000 \cdot T = 150,000$ rounds.

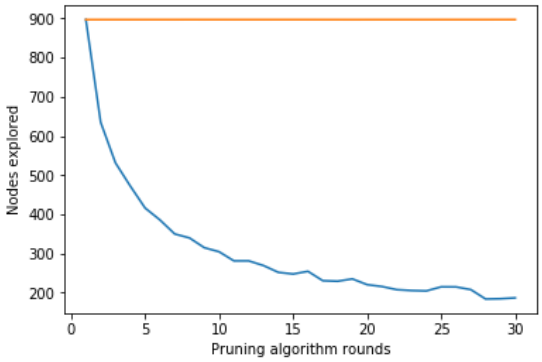



(a) Average pruned set size for shortest-path routing.

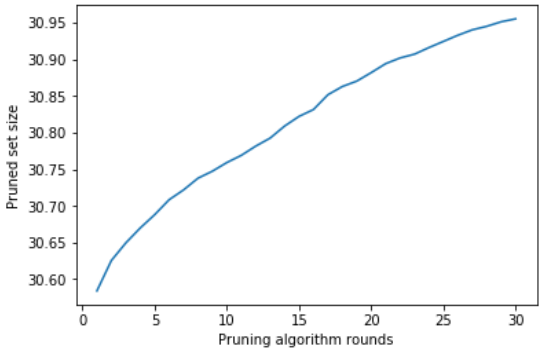



(b) Average pruned set size for linear programming.

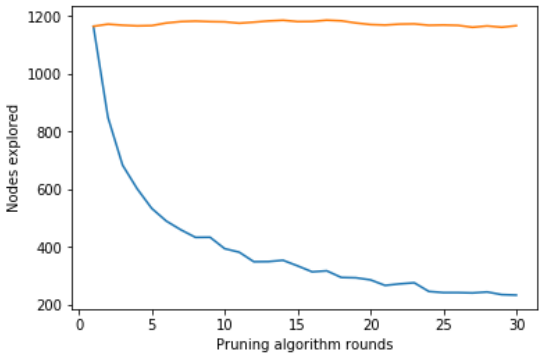
Figure C.1: Average size of the pruned set $\bar{S}_i$ in Algorithm 9.



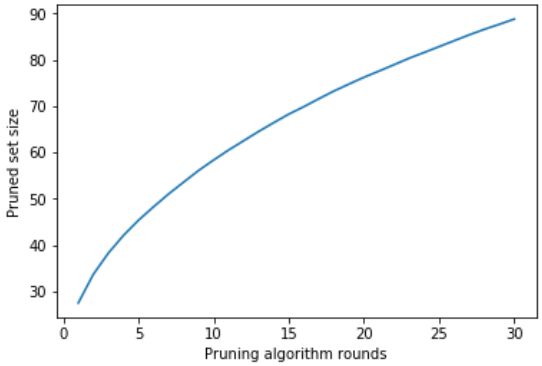

(a) Gaussian perturbation with a standard deviation of $1/2$. Top line: average number of nodes Dijkstra's algorithm explores. Bottom line: average number of nodes Algorithm 9 explores.



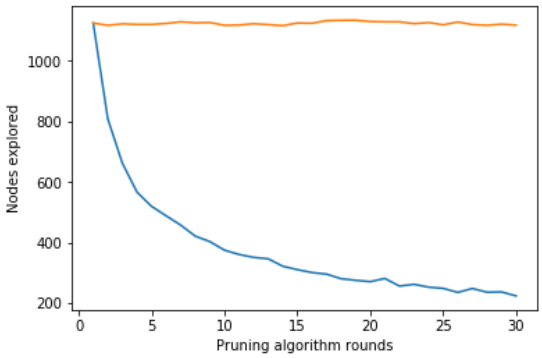

(b) Gaussian perturbation with a standard deviation of $1/2$. Average pruned set size for shortest path routing.



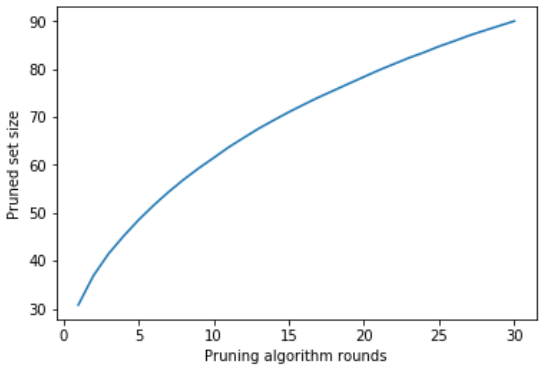

(c) Uniform perturbation with support $[-1/2, 1/2]$. Top line: average number of nodes Dijkstra's algorithm explores. Bottom line: average number of nodes Algorithm 9 explores.




(d) Uniform perturbation with support $[-1/2, 1/2]$. Average pruned set size for shortest path routing.




(e) Uniform perturbation with support $[-1, 1]$. Top line: average number of nodes Dijkstra's algorithm explores. Bottom line: average number of nodes Algorithm 9 explores.




(f) Uniform perturbation with support $[-1/2, 1/2]$. Average pruned set size for shortest path routing.

Figure C.2: Shortest path routing experiments using varying perturbation methods.

# Bibliography


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

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

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

Alessandro Acquisti and Hal R. Varian. Conditioning prices on purchase history. *Marketing Science*, 24(3):1–15, 2005.

William James Adams and Janet L. Yellen. Commodity bundling and the burden of monopoly. *Quarterly Journal of Economics*, 90(3):475–498, Aug 1976.

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.

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.

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.

Kareem Amin, Afshin Rostamizadeh, and Umar Syed. Learning prices for repeated auctions with strategic buyers. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2013.

Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 142–157. Springer-Verlag, 2009. ISBN 3642042430.

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

Victor F. Araman and René Caldentey. Dynamic pricing for nonperishable products with demand learning. *Operations Research*, 57(5):1169–1188, 2009.

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.

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

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

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

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

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.

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.

Moshe Babaioff, Yannai A Gonczarowski, and Noam Nisan. The menu-size complexity of revenue approximation. *Games and Economic Behavior*, 2021.

Yannis Bakos and Erik Brynjolfsson. Bundling information goods: Pricing, profits, and efficiency. *Management Science*, 45(12):1613–1630, Dec 1999.

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

Maria-Florina Balcan. Data-driven algorithm design. In Tim Roughgarden, editor, *Beyond Worst Case Analysis of Algorithms*. Cambridge University Press, 2020.

Maria-Florina Balcan, Avrim Blum, Jason D. Hartline, and Yishay Mansour. Mechanism design via machine learning. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 605–614, 2005.

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.

Maria-Florina Balcan, Amit Daniely, Ruta Mehta, Ruth Urner, and Vijay V Vazirani. Learning economic parameters from revealed preferences. In *International Workshop On Internet And Network Economics (WINE)*, 2014.

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.

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

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

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

Maria-Florina Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized Lloyd's families. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018c.

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*, 2018d.

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)*, 2018e.

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

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

Maria-Florina Balcan, Travis Dick, and Wesley Pegden. Semi-bandit optimization in the dispersed setting. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020b.

Maria-Florina Balcan, Travis Dick, and Dravyansh Sharma. Learning piecewise lipschitz functions in changing environments. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020c.

Maria-Florina Balcan, Siddharth Prasad, and Tuomas Sandholm. Efficient algorithms for learning revenue-maximizing two-part tariffs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2020d.

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

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

Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization

guarantees for data-driven algorithm design. *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2021a.

Maria-Florina Balcan, Siddharth Prasad, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of tree search configuration: Cutting planes and beyond. *arXiv preprint arXiv:2106.04033*, 2021b.

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. In *AAAI Conference on Artificial Intelligence*, 2021c.

Afonso S. Bandeira, Nicolas Boumal, and Vladislav Voroninski. On the low-rank approach for semidefinite programs arising in synchronization and community detection. In *Conference on Learning Theory (COLT)*, pages 361–382, 2016.

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.

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

MohammadHossein Bateni, Sina Dehghani, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Revenue maximization for selling multiple correlated items. In *Proceedings of the European Symposium on Algorithms (ESA)*, 2015.

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

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.

Michael Benisch, Norman Sadeh, and Tuomas Sandholm. Methodology for designing reasonably expressive mechanisms with application to ad auctions. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009. Earlier version in proceedings of ACM EC-08 Workshop on Advertisement Auctions.

Jon Louis Bentley, David S Johnson, Frank Thomson Leighton, Catherine C McGeoch, and Lyle A McGeoch. Some unexpected expected behavior results for bin packing. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 279–288, 1984.

Omar Besbes and Assaf Zeevi. Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms. *Operations Research*, 57(6):1407–1420, 2009.

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.

Avrim Blum, Chen Dan, and Saeed Seddighin. Learning complexity of simulated annealing. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1540–1548. PMLR, 2021.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

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

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.

William Brendel and Sinisa Todorovic. Segmentation as maximum-weight independent set. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 307–315, 2010.

Josef Broder and Paat Rusmevichientong. Dynamic pricing under a general parametric choice model. *Operations Research*, 60(4):965–980, 2012.

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.

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

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)*, 2017a.

Yang Cai and Constantinos Daskalakis. Learning multi-item auctions with (or without) samples. *arXiv preprint arXiv:1709.00228*, 2017b.

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.

Isabel Cenamor, Tomás De La Rosa, and Fernando Fernández. The IBaCoP planning system: Instance-based configured portfolios. *Journal of Artificial Intelligence Research*, 56:657–691, 2016.

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

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.

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

Shuchi Chawla, Jason D Hartline, David L Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2010.

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

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

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

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

Ed H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

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

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

Michael Collins. Discriminative reranking for natural language parsing. *International Conference on Machine Learning (ICML)*, 2000.

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.

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.

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

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.

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

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

Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 313–320, 2006.

Michael Curry, Ping-Yeh Chiang, Tom Goldstein, and John Dickerson. Certifying strategyproof auction networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

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

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

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

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

Yuan Deng, Sébastien Lahaie, Vahab Mirrokni, and Song Zuo. A data-driven metric of incentive compatibility. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 1796–1806, 2020.

Stylianos Despotakis, R Ravi, and Amin Sayedi. First-price auctions in online display advertising. *Available at SSRN 3485410*, 2019.

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.

Nikhil R Devanur, Zhiyi Huang, and Christos-Alexandros Psomas. The sample complexity of auctions with side information. *arXiv preprint arXiv:1511.02296*, 2017.

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.

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.

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.

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

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.

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.

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.

Alon Eden, Michal Feldman, Ophir Friedler, Inbal Talgam-Cohen, and S Matthew Weinberg. A simple and approximately optimal mechanism for a buyer with complements. *Operations Research*, 69(1):188–206, 2021.

Robert C Edgar. Quality measures for protein alignment benchmarks. *Nucleic acids research*, 38 (7):2145–2153, 2010.

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

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

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

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

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

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

Darya Filippova, Rob Patro, Geet Duggal, and Carl Kingsford. Identification of alternative topological domains in chromatin. *Algorithms for Molecular Biology*, 9:14, May 2014.

Martina Fischetti, Andrea Lodi, and Giulia Zarpellon. Learning MILP resolution outcomes before reaching time-limit. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, pages 275–291, 2019.

Roy Frostig, Sida Wang, Percy S Liang, and Christopher D Manning. Simple MAP inference via low-rank relaxations. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3077–3085, 2014.

Hu Fu and Tao Lin. Learning utilities and equilibria in non-truthful auctions. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

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

Vikas Garg and Adam Kalai. Supervising unsupervised learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

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

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

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.

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.

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.

Negin Golrezaei, Adel Javanmard, and Vahab Mirrokni. Dynamic incentive-aware learning: Robust pricing in contextual auctions. *Operations Research*, 2020. (forthcoming).

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

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.

Yannai A Gonczarowski and S Matthew Weinberg. The sample complexity of up-to-$\varepsilon$ multi-dimensional revenue maximization. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2018.

Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705 – 708, 1982. ISSN 0022-2836.

Theodore Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

Chenghao Guo, Zhiyi Huang, and Xinzhi Zhang. Settling the sample complexity of single-parameter revenue maximization. *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2019.

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

Dan Gusfield, Krishnan Balasubramanian, and Dalit Naor. Parametric optimization of sequence alignment. *Algorithmica*, 12(4-5):312–326, 1994.

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/.

Robert Haralick and Gordon Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

Sergiu Hart and Noam Nisan. Approximate revenue maximization with multiple items. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2012.

Sergiu Hart and Noam Nisan. Approximate revenue maximization with multiple items. *Journal of Economic Theory*, 172:313–347, 2017.

Jason Hartline and Samuel Taggart. Non-revelation mechanism design. *arXiv preprint arXiv:1608.01875*, 2016.

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)*, 2014a.

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at Facebook. In *Proceedings of the International Workshop on Data Mining for Online Advertising*, 2014b.

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.

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.

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.

Qixing Huang, Yuxin Chen, and Leonidas Guibas. Scalable semidefinite relaxation for maximum a posterior estimation. In *International Conference on Machine Learning (ICML)*, pages 64–72, 2014.

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.

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.

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

Frank Hutter, Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

IBM ILOG Inc. CPLEX 11.0 release notes, 2007.

IBM ILOG Inc. CPLEX 12.8 Parameters Reference, 2017.

Raj Iyer, David Karger, Hariharan Rahul, and Mikkel Thorup. An experimental study of polylogarithmic, fully dynamic, connectivity algorithms. *ACM Journal of Experimental Algorithmics*, 6:4–es, December 2002. ISSN 1084-6654.

Philippe Jehiel, Moritz Meyer-Ter-Vehn, and Benny Moldovanu. Mixed bundling auctions. *Journal of Economic Theory*, 134(1):494–512, 2007.

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

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].

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.

Adam Tauman Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005.

Yash Kanoria and Hamid Nazerzadeh. Incentive-compatible learning of reserve prices for repeated auctions. *Operations Research*, 2020. (forthcoming).

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.

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*, 2016.

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

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

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

Peter J Kolesar. A branch and bound algorithm for the knapsack problem. *Management science*, 13(9):723–735, 1967.

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

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.

Pravesh Kothari, Sahil Singla, Divyarthi Mohan, Ariel Schvartzman, and S Matthew Weinberg. Approximation schemes for a unit-demand buyer with independent items via symmetries. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2019.

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.

Vijay Krishna. *Auction Theory*. Academic Press, 2002.

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.

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

Sébastien Lahaie, Andrés Muñoz Medina, Balasubramanian Sivan, and Sergei Vassilvitskii. Testing incentive compatibility in display ad auctions. In *Proceedings of the International World Wide Web Conference (WWW)*, 2018.

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica*, pages 497–520, 1960.

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.

Pierre Le Bodic and George Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, pages 1–37, 2017.

Kevin Leyton-Brown. *Resource allocation in competitive multiagent systems*. PhD thesis, Stanford University, 2003.

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.

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

Kevin Leyton-Brown, Paul Milgrom, and Ilya Segal. Economics and computer science of a radio spectrum reallocation. *Proceedings of the National Academy of Sciences*, 114(28):7202–7209, 2017.

Xinye Li and Andrew Chi-Chih Yao. On revenue maximization for selling multiple independently distributed items. *Proceedings of the National Academy of Sciences*, 110(28):11232–11237, 2013.

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

Paolo Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1-2):315–326, 2000.

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.

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.

Anton Likhodedov and Tuomas Sandholm. Approximating revenue-maximizing combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, PA, 2005.

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

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

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.

Benjamin Lubin and David C Parkes. Approximate strategyproofness. *Current Science*, pages 1021–1032, 2012.

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.

Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning (ICML)*, 2018.

Alejandro M Manelli and Daniel R Vincent. Bundling as an optimal selling mechanism for a multiple-good monopolist. *Journal of Economic Theory*, 127(1):1–35, 2006.

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

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

R. Preston McAfee, John McMillan, and Michael D. Whinston. Multiproduct monopoly, commodity bundling, and correlation of values. *Quarterly Journal of Economics*, 104(2):371–383, May 1989.

Catherine C McGeoch. *A guide to experimental algorithmics*. Cambridge University Press, 2012.

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.

Debasis Mishra and Arunava Sen. Roberts' theorem with neutrality: A social welfare ordering approach. *Games and Economic Behavior*, 75(1):283–298, 2012.

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.

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.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

John Morgan, Kenneth Steiglitz, and George Reis. The spite motive and equilibrium behavior in auctions. *Contributions to Economic Analysis & Policy*, 2(1), 2003.

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.

Jamie Morgenstern and Tim Roughgarden. Learning simple auctions. In *Conference on Learning Theory (COLT)*, 2016.

Andrés Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Roger Myerson. Optimal auction design. *Mathematics of Operation Research*, 6:58–73, 1981.

Balas K Natarajan. On learning sets and functions. *Machine Learning*, 4(1):67–97, 1989.

Swaprava Nath and Tuomas Sandholm. Efficiency and budget balance in general quasi-linear domains. *Games and Economic Behavior*, 113:673 – 693, 2019.

Tri-Dung Nguyen and Tuomas Sandholm. Multi-option descending clock auction. Technical report, 2015. Mimeo.

Sergio Núñez, Daniel Borrajo, and Carlos Linares López. Automatic construction of optimal static sequential portfolios for AI planning and beyond. *Artificial Intelligence*, 226:75–101, 2015.

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.

Walter Y Oi. A Disneyland dilemma: Two-part tariffs for a Mickey Mouse monopoly. *The Quarterly Journal of Economics*, 85(1):77–96, 1971.

Lior Pachter and Bernd Sturmfels. Parametric inference for biological sequence analysis. *Proceedings of the National Academy of Sciences*, 101(46):16138–16143, 2004a. doi: 10.1073/pnas.0406011101.

Lior Pachter and Bernd Sturmfels. Tropical geometry of statistical models. *Proceedings of the National Academy of Sciences*, 101(46):16132–16137, 2004b. doi: 10.1073/pnas.0406010101.

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.

Rachel Parkin. A year in first-price. *Ad Exchanger*, 2018. URL https://adexchanger.com/the-sell-sider/a-year-in-first-price/.

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.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

David Pollard. *Convergence of Stochastic Processes*. Springer, 1984.

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.

J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Andrej Risteski and Yuanzhi Li. Approximate maximum entropy principles via Goemans-Williamson with applications to provable variational methods. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 4628–4636, 2016.

Kevin Roberts. The characterization of implementable social choice rules. In J-J Laffont, editor, *Aggregation and Revelation of Preferences*. North-Holland Publishing Company, 1979.

Michael Rothkopf, Thomas Teisberg, and Edward Kahn. Why are Vickrey auctions rare? *Journal of Political Economy*, 98(1):94–109, 1990.

Tim Roughgarden and Okke Schrijvers. Ironing in the dark. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2016.

Aviad Rubinstein and S Matthew Weinberg. Simple mechanisms for a subadditive buyer and applications to revenue monotonicity. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2015.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

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.

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

Tuomas Sandholm. 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.

Tuomas Sandholm. Issues in computational Vickrey auctions. *International Journal of Electronic Commerce*, 4(3):107–129, 2000. Early version in ICMAS-96.

Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, January 2002.

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.

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.

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.

Tuomas Sandholm and Anton Likhodedov. 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.

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.

Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1): 145–147, 1972.

Tzur Sayag, Shai Fine, and Yishay Mansour. Combining multiple heuristics. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 242–253. Springer, 2006.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

Ankit Sharma and Tuomas Sandholm. Asymmetric spite in auctions. In *AAAI Conference on Artificial Intelligence*, 2010.

Weiran Shen, Pingzhong Tang, and Song Zuo. Automated mechanism design via neural networks. In *Autonomous Agents and Multi-Agent Systems*, pages 215–223, 2019.

Sarah Sluis. Google switches to first-price auction. *Ad Exchanger*, 2019. URL https://www.adexchanger.com/online-advertising/google-switches-to-first-price-auction/.

Sagi Snir and Satish Rao. Using max cut to enhance rooted trees consistency. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):323–333, 2006.

Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.

Matthew Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1577–1584, 2009.

Matthew Streeter, Daniel Golovin, and Stephen F. Smith. Combining multiple heuristics online. In *AAAI Conference on Artificial Intelligence*, 2007.

Vasilis Syrgkanis. A sample complexity measure with applications to learning optimal auctions. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Pingzhong Tang. Reinforcement mechanism design. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Pingzhong Tang and Tuomas Sandholm. Mixed-bundling auctions with reserve prices. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2012a.

Pingzhong Tang and Tuomas Sandholm. Optimal auctions for spiteful bidders. In *AAAI Conference on Artificial Intelligence*, 2012b.

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

John Thanassoulis. Haggling over substitutes. *Journal of Economic theory*, 117(2):217–245, 2004.

Timo Tossavainen. On the zeros of finite sums of exponential functions. *Australian Mathematical Society Gazette*, 33(1):47–50, 2006.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

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.

Vladimir Vapnik and Alexey Chervonenkis. Theory of pattern recognition, 1974.

Hal R. Varian. Position auctions. *International Journal of Industrial Organization*, pages 1163–1178, 2007.

Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

William Walsh, David Parkes, Tuomas Sandholm, and Craig Boutilier. Computing reserve prices and identifying the value distribution in real-world auctions with market disruptions. In *AAAI Conference on Artificial Intelligence*, 2008. Short paper.

Jun Wang, Tony Jebara, and Shih-Fu Chang. Semi-supervised learning using greedy max-cut. *Journal of Machine Learning Research*, 14(Mar):771–800, 2013.

Michael S Waterman, Temple F Smith, and William A Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.

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

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

Gellért Weisz, András György, Wei-I Lin, Devon R Graham, Kevin Leyton-Brown, Csaba Szepesvari, and Brendan Lucier. ImpatientCapsAndRuns: Approximately optimal algorithm configuration from an infinite pool. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

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

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.

H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.

David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University press, 2011.

Robert B Wilson. *Nonlinear pricing*. Oxford University Press on Demand, 1993.

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

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

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI Conference on Artificial Intelligence*, 2010.

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

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.

Hector Yee and Bar Ifrach. Aerosolve: Machine learning for humans. *Open Source*, 2015. URL http://nerds.airbnb.com/aerosolve/.

Chihiro Yoshimura, Masanao Yamaoka, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, Ken-ichi Kawarabayashi, and Hiroyuki Mizuno. Uncertain behaviours of integrated circuits improve computational performance. *Scientific reports*, 5, 2015.

Mingjun Zhong, Nigel Goddard, and Charles Sutton. Signal aggregate constraints in additive factorial HMMs, with application to energy disaggregation. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3590–3598, 2014.