How Much Data Is Sufficient to Learn High-Performing Algorithms?

MARIA-FLORINA BALCAN, Computer Science, Carnegie Mellon University, Pittsburgh, United States
DAN DEBLASIO, Computational biology, Carnegie Mellon University, Pittsburgh, United States
TRAVIS DICK, Google Inc New York, New York, United States
CARL KINGSFORD, Computational biology, Carnegie Mellon University, Pittsburgh, United States
TUOMAS SANDHOLM, Computer science, Carnegie Mellon University, Pittsburgh, United States
ELLEN VITERCIK, Management Science & Engineering; Computer science, Stanford University, Stanford, United States

Algorithms often have tunable parameters that impact performance metrics such as runtime and solution quality. For many algorithms used in practice, no parameter settings admit meaningful worst-case bounds, so the parameters are made available for the user to tune. Alternatively, parameters may be tuned implicitly within the proof of a worst-case approximation ratio or runtime bound. Worst-case instances, however, may be rare or nonexistent in practice. A growing body of research has demonstrated that a data-driven approach to parameter tuning can lead to significant improvements in performance. This approach uses a *training set* of problem instances sampled from an unknown, application-specific distribution and returns a parameter setting with strong average performance on the training set.

We provide techniques for deriving *generalization guarantees* that bound the difference between the algorithm's average performance over the training set and its expected performance on the unknown distribution. Our results apply no matter how the parameters are tuned, be it via an automated or manual approach. The challenge is that for many types of algorithms, performance is a volatile function of the parameters: slightly perturbing the parameters can cause a large change in behavior. Prior research [e.g., 12, 16, 20, 62]

A short early version of this article appeared in 2021 at the ACM Symposium on Theory of Computing (STOC) [11].
This research is funded in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative (GBMF4554 to C.K.), the US National Institutes of Health (R01GM122935 to C.K.), the US National Science Foundation (a Graduate Research Fellowship and CCF-2338226 to E.V. and grants IIS-1901403 to M.B. and T.S., IIS-1618714, CCF-1535967, CCF-1910321, and SES-1919453 to M.B., RI-2312342, IIS-1718457, IIS-1617590, and CCF-1733556 to T.S., and DBI-1937540 to C.K.), the US Army Research Office (W911NF2210266, W911NF-17-1-0082, and W911NF2010081 to T.S.), the Vannevar Bush Faculty Fellowship to T.S., the Office of Naval Research (N00014-23-1-2876 to T.S.), the Defense Advanced Research Projects Agency under cooperative agreement HR00112020003 to M.B., an AWS Machine Learning Research Award to M.B., an Amazon Research Award to M.B., a Microsoft Research Faculty Fellowship to M.B., a Bloomberg Research Grant to M.B., a fellowship from Carnegie Mellon University's Center for Machine Learning and Health to E.V., and by the generosity of Eric and Wendy Schmidt by recommendation of the Schmidt Futures program.
Authors' Contact Information: Maria-Florina Balcan, Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States; e-mail: ninamf@cs.cmu.edu; Dan DeBlasio, Computational biology, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States; e-mail: dfdeblasio@utep.edu; Travis Dick, Google Inc New York, New York, New York, United States; e-mail: tbd@seas.upenn.edu; Carl Kingsford, Computational biology, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States; e-mail: carlk@cs.cmu.edu; Tuomas Sandholm, Computer science, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States; e-mail: sandholm@cs.cmu.edu; Ellen Vitercik, Management Science & Engineering; Computer science, Stanford University, Stanford, California, United States; e-mail: vitercik@stanford.edu.

ACM 0004-5411/2024/10-ART32
https://doi.org/10.1145/3676278

has proved generalization bounds by employing case-by-case analyses of greedy algorithms, clustering algorithms, integer programming algorithms, and selling mechanisms. We streamline these analyses with a general theorem that applies whenever an algorithm's performance is a piecewise-constant, piecewise-linear, or—more generally—*piecewise-structured* function of its parameters. Our results, which are tight up to logarithmic factors in the worst case, also imply novel bounds for configuring dynamic programming algorithms from computational biology.

CCS Concepts: • **Theory of computation → Sample complexity and generalization bounds;**

Additional Key Words and Phrases: Automated algorithm design, data-driven algorithm design, automated algorithm configuration, machine learning theory, computational biology

ACM Reference Format:
Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. 2024. How Much Data Is Sufficient to Learn High-Performing Algorithms?. *J. ACM* 71, 5, Article 32 (October 2024), 58 pages. https://doi.org/10.1145/3676278

1 Introduction

Algorithms often have tunable parameters that impact performance metrics such as runtime, solution quality, and memory usage. These parameters may be set explicitly, 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 after careful parameter tuning, 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 to guarantee a worst-case approximation factor or runtime bound. Worst-case bounds, however, can be overly pessimistic in practice. In response to these challenges, a growing body of research has demonstrated the power of using machine learning to find parameter settings that work particularly well on problems from the application domain at hand.

We present techniques for proving *generalization guarantees* for these data-driven approaches to parameter tuning. We adopt a natural learning-theoretic model introduced by Gupta and Roughgarden [62] where—as in the applied literature on this topic [e.g., 68, 71, 74, 83, 109, 109, 129, 130]—we assume there is an unknown, application-specific distribution over the algorithm's input instances. A learning procedure receives a *training set* sampled from this distribution and returns a parameter setting—or *configuration*—with strong average performance over the training set. If the training set is too small, this configuration may have poor expected performance. *Generalization guarantees* bound the difference between average performance over the training set and actual expected performance. Our guarantees apply no matter how the parameters are tuned, via an algorithmic search [e.g., 35, 109, 129, 130], or manually [e.g., 25, 73, 90].

Across many types of algorithms—for example, combinatorial algorithms, integer programs, and dynamic programs—the algorithm's performance is a volatile function of its parameters. This is a key challenge that distinguishes our results from prior research on generalization guarantees. For well-understood functions in machine learning theory, there is generally a simple connection between a function's parameters and the value of the function. Meanwhile, slightly perturbing an algorithm's parameters can cause significant changes in its behavior and performance.

To provide generalization bounds, we exploit useful structure that governs these volatile performance functions. This structure depends on the relationship between *primal* and *dual*

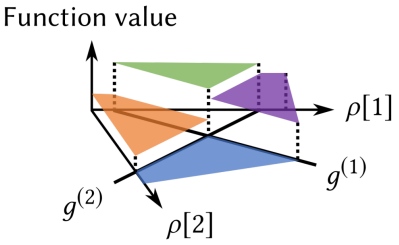


Fig. 1. A piecewise-constant function over $\mathbb{R}^2_{\geq 0}$ with linear boundary functions $g^{(1)}$ and $g^{(2)}$.

functions [5]. To derive generalization bounds, a common strategy is to calculate the *intrinsic complexity* of a function class $\mathcal{U}$ which we refer to as the *primal class*. Every function $u_{\boldsymbol{\rho}} \in \mathcal{U}$ is defined by a parameter setting $\boldsymbol{\rho} \in \mathbb{R}^d$ and $u_{\boldsymbol{\rho}}(x) \in \mathbb{R}$ measures the performance of the algorithm parameterized by $\boldsymbol{\rho}$ given the input x. We measure intrinsic complexity using the classic notion of *pseudo-dimension* [103]. This is a challenging task because the domain of every function in $\mathcal{U}$ is a set of problem instances, so there are no obvious notions of Lipschitz continuity or smoothness on which we can rely. Instead, we use structure exhibited by the *dual class* $\mathcal{U}^*$. Every *dual function* $u^*_x \in \mathcal{U}^*$ is defined by a problem instance x and measures the algorithm's performance as a function of its parameters given x as input. The dual functions have a simple, Euclidean domain $\mathbb{R}^d$ and they have ample structure which we can use to bound the pseudo-dimension of $\mathcal{U}$.

1.1 Our Contributions

Our results apply to any parameterized algorithm with dual functions that exhibit a clear-cut structural property: the duals are piecewise constant, piecewise linear, or—more broadly—piecewise *structured*. The parameter space decomposes into a small number of regions such that within each region, the algorithm's performance is "well-behaved." As an example, Figure 1 illustrates a piecewise-structured function of two parameters $\rho[1]$ and $\rho[2]$. There are two functions $g^{(1)}$ and $g^{(2)}$ that define a partition of the parameter space and four constant functions that define the function value on each subset from this partition.

More formally, the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-*piecewise decomposable* if for every problem instance, there are at most k boundary functions from a set $\mathcal{G}$ (for example, the set of linear separators) that partition the parameter space into regions such that within each region, algorithmic performance is defined by a function from a set $\mathcal{F}$ (for example, the set of constant functions). We bound the pseudo-dimension of $\mathcal{U}$ in terms of the pseudo- and VC-dimensions of the dual classes $\mathcal{F}^*$ and $\mathcal{G}^*$, denoted $\text{Pdim}(\mathcal{F}^*)$ and $\text{VCdim}(\mathcal{G}^*)$. This yields our main theorem: if $[0, H]$ is the range of the functions in $\mathcal{U}$, then with probability $1 - \delta$ over the draw of N training instances, for any parameter setting, the difference between the algorithm's average performance over the training set and its expected performance is $O(H\sqrt{\frac{1}{N}((\text{Pdim}(\mathcal{F}^*) + \text{VCdim}(\mathcal{G}^*)) \ln(k(\text{Pdim}(\mathcal{F}^*) + \text{VCdim}(\mathcal{G}^*))) + \ln \frac{1}{\delta})})$. Specifically, we prove that $\text{Pdim}(\mathcal{U}) = \tilde{O}((\text{Pdim}(\mathcal{F}^*) + \text{VCdim}(\mathcal{G}^*)) \ln k)$ and that this bound is tight up to log factors. The classes $\mathcal{F}$ and $\mathcal{G}$ are often so well structured that bounding $\text{Pdim}(\mathcal{F}^*)$ and $\text{VCdim}(\mathcal{G}^*)$ is straightforward.

This article contributes to a line of research [6, 12, 13, 16, 20, 62] that provides generalization bounds for a selection of parameterized algorithms, including greedy algorithms [62], clustering algorithms [16], and integer programming algorithms [12, 16], as well as mechanism design for revenue maximization [20, tying into a longer line of research on this topic described in Section 1.2]. These works uncover structural properties of these algorithms that then imply—in the words of

Table 1. Summary of the New Application Areas in This Article

Problem	Section
Sequence alignment	Section 4.1
RNA folding	Section 4.2
Prediction of topologically associating domains	Section 4.3

this article—that the parameterized algorithms' dual classes are piecewise decomposable. Our main theorem then immediately implies generalization bounds for these classes, thus streamlining these articles' analyses. In this article, we also derive new generalization bounds for computational biology algorithms.

Proof insights. At a high level, we prove this guarantee by counting the number of parameter settings with significantly different performance over any set $\mathcal{S}$ of problem instances. To do so, we first count the number of regions induced by the $|\mathcal{S}|k$ boundary functions that correspond to these problem instances. This step subtly depends not on the VC-dimension of the class of boundary functions $\mathcal{G}$, but rather on $\text{VCdim}(\mathcal{G}^*)$. These $|\mathcal{S}|k$ boundary functions partition the parameter space into regions where across all instances x in $\mathcal{S}$, the dual functions u_x^* are simultaneously structured. Within any one region, we use the pseudo-dimension of the dual class $\mathcal{F}^*$ to count the number of parameter settings in that region with significantly different performance. We aggregate these bounds over all regions to bound the pseudo-dimension of $\mathcal{U}$.

Parameterized dynamic programming algorithms from computational biology. Our results imply bounds for a variety of computational biology algorithms that are used in practice. We analyze parameterized sequence alignment algorithms [47, 59, 63, 101, 102] as well as RNA folding algorithms [100], which predict how an input RNA strand would naturally fold, offering insight into the molecule's function. We also provide guarantees for algorithms that predict topologically associating domains in DNA sequences [48], which shed light on how DNA wraps into three-dimensional structures that influence genome function. We provide a summary of these application areas in Table 1.

Parameterized voting mechanisms. A *mechanism* is a special type of algorithm designed to help a set of agents come to a collective decision. For example, a town's residents may want to build a public resource such as a park, pool, or skating rink, and a mechanism would help them decide which to build (as in participatory democracy [e.g., 52]). In Appendix F, we analyze *neutral affine maximizers* [91, 97, 106], a well-studied family of parameterized mechanisms. The parameters can be tuned to maximize social welfare, which is the sum of the agents' values for the mechanism's outcome. We study the standard single-shot setting where there is a distribution over agents' values that represents the mechanism designer's domain-specific knowledge. Rather than designing the mechanism using the complex analytical form of the joint distribution—as is typical in mechanism design—we study the case where the mechanism designer only needs samples from this distribution.

1.2 Additional Related Research

Starting with research by Gupta and Roughgarden [62] and followed by Balcan et al. [16], a growing body of research investigates learning-theoretic guarantees for incorporating machine learning into the process of algorithm design, known as *data-driven algorithm design* [e.g., 6, 7, 12, 13, 16, 20, 26, 50, 62]. A chapter by Balcan [8] provides a survey. We highlight a few of the articles that are most related to ours below.

1.2.1 Prior Research.

Runtime optimization with provable guarantees. Kleinberg et al. [78, 79] and Weisz et al. [126, 127] provide configuration procedures with provable guarantees when the goal is to minimize runtime. In contrast, our bounds apply to arbitrary performance metrics, such as solution quality as well as runtime. Also, their procedures are designed for the case where the set of parameter settings is finite (although they can still offer some guarantees when the parameter space is infinite by first sampling a finite set of parameter settings and then running the configuration procedure; Balcan et al. [12, 21] study what kinds of guarantees discretization approaches can and cannot provide). In contrast, our guarantees apply immediately to infinite parameter spaces. Finally, unlike our results, the guarantees from this prior research are not configuration-procedure-agnostic: they apply only to the specific procedures that are proposed.

Learning-augmented algorithms. A related line of research has designed algorithms that replace some steps of a classic worst-case algorithm with a machine-learned oracle that makes predictions about structural aspects of the input [69, 89, 92, 105]. If the prediction is accurate, the algorithm's performance (for example, its error or runtime) is superior to the original worst-case algorithm, and if the prediction is incorrect, the algorithm's performance is close to that of the best-known worst-case algorithm. Though similar, our approach to parameter tuning is different because we are not attempting to learn structural aspects of the input; rather, we provide guarantees for tuning the algorithm's parameters directly using the training set.

Online and private configuration with well-structured duals. Prior research has also demonstrated the utility of dual functions for parameter tuning in the context of *online learning* [7, 13, 23, 33, 62]—where problem instances arrive one-by-one and the goal is to minimize regret with respect to the best configuration in hindsight—and *private algorithm configuration* [13]—where the goal is to find a high-performing configuration without revealing sensitive information in the training set. These tasks are impossible in the worst case, so these articles identify a property of the dual functions under which online and private configuration are possible. Balcan et al. [13] call this property *dispersion*, which, roughly speaking, requires that the discontinuities of the dual functions are not too concentrated in any ball. Online learning guarantees imply sample complexity guarantees due to online-to-batch conversion, and Balcan et al. [13] 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 apply to a broader set of problems in the distributional learning model. In other words, the results from this article cannot be recovered using the techniques of Balcan et al. [7, 13]. However, the techniques of Balcan et al. [7, 13] apply to a broader set of problems, including private optimization, where the goal is to find a configuration without leaking sensitive information contained within the training set.

Mechanism design. Contributing to a line of research on sample complexity bounds for revenue maximization [e.g., 3, 27, 29, 34, 36, 40, 56, 57, 61, 65, 70, 93–96, 107, 117], Balcan et al. [20][1] prove that a wide variety of selling mechanisms' revenue functions are piecewise linear in their

[1]The reference by Balcan et al. [20] was an extended abstract, and the same set of results will appear in a journal publication [19].

parameters (for example, their prices). They use this structure to prove generalization guarantees. In the language of this article, Balcan et al. [20] prove that these mechanism families have piecewise-decomposable dual functions where the boundary and piece functions are linear, though Balcan et al. [20] did not frame their problem in the language of primal and dual functions.

This article advances over the article by Balcan et al. [20] in several respects. First, the case where the boundary and piece functions are linear is an especially simple case of our broader results because the dual of a class of linear functions is linear—there is essentially no distinction between the primal and dual functions. Balcan et al. [20] could therefore apply classical results about hyperplane arrangements [28] to count the number of regions induced by the boundary functions, a result that does not apply to more general boundary functions.

Crucially, since the result of Balcan et al. [20] is specific to linear functions, it would not apply to prior or subsequent research where both the piece and boundary functions are not linear [15–18]. First, as we describe in Section 5.2, prior research studied the configuration of integer quadratic programming approximation algorithms [16]. In that setting, the piece functions are inverse-quadratic, of the form $\frac{a}{\rho^2} + \frac{b}{\rho} + c$ for constants $a, b, c \in \mathbb{R}$. This prior research demonstrated that Balcan et al.'s result [20] for piecewise-linear revenue functions was not sufficient to provide a general theory for algorithm configuration.

Subsequent research has validated the utility of this article's broadly applicable results, confirming that piecewise linear functions are only a special case in algorithm configuration. For example, subsequent research has shown that the results in this article can be applied to (1) selecting cutting planes for integer programming solvers, where the boundary functions are multi-dimensional polynomial hypersurfaces [17, 18], and (2) configuring the regularization parameters of ElasticNet, where the boundary functions are semi-algebraic sets bounded by algebraic curves and the piece functions are rational polynomial functions [15]. These results take us increasingly far from the special case of piecewise-linear revenue functions. Therefore, this article aims not only at generalizing the result of Balcan et al. [20] but also to find an abstraction that simultaneously captures that piecewise-linear structure and the non-linear structure observed for other configuration problems.

Since there is no distinction between the linear primal and dual functions in the article by Balcan et al. [20], the natural but incorrect extrapolation of that article's results to non-linear functions would be that the pseudo-dimension of an $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable function class $\mathcal{U}$ will depend on the VC and pseudo-dimensions of $\mathcal{F}$ and $\mathcal{G}$, when this is not the case—rather, the pseudo-dimension of $\mathcal{U}$ depends on the intrinsic complexities of the duals $\mathcal{F}^*$ and $\mathcal{G}^*$ of these function classes. In general, the difference between $\text{VCdim}(\mathcal{G})$ and $\text{VCdim}(\mathcal{G}^*)$ can be exponential, so this is a subtle but important distinction.

1.2.2 Concurrent and Subsequent Research. Subsequently to the appearance of the original version of this article in 2019 [10], an extensive body of research has developed that studies the use of machine learning in the context of algorithm design, as we highlight below.

Learning-augmented algorithms. The literature on learning-augmented algorithms (summarized in the previous section) has continued to flourish in subsequent research [37, 38, 42, 43, 72, 76, 82, 82, 125]. Some of these articles make explicit connections to the types of parameter tuning problems we study in this article, such as research by Lavastida et al. [82], who study online flow allocation and makespan minimization problems. They formulate the machine-learned predictions as a set of parameters and study the sample complexity of learning a good parameter setting. An interesting direction for future research is to investigate which other problems from this literature can be formulated as parameter tuning problems, and whether the techniques in this article can be used to derive tighter or novel guarantees.

Sample complexity bounds for algorithm design. As indicated in Section 1.2.1, several subsequent articles have used the main results in this article to provide sample complexity bounds [2, 15, 17, 18, 108]. Additional applications have included learning heuristic functions for search [108] and algorithmic fairness [2].

Chawla et al. [30] study the *Pandora's box problem*, where there is a set of alternatives with costs drawn from an unknown distribution. A search algorithm observes the alternatives' costs one by one, eventually stopping and selecting one alternative. The authors show how to learn an algorithm that minimizes the selected alternative's expected cost, plus the number of alternatives the algorithm examines. The primary contributions of that article are (1) identifying a finite subset of algorithms that compete with the optimal algorithm, and (2) showing how to efficiently optimize over that finite subset of algorithms. Since the authors prove that they only need to optimize over a finite subset of algorithms, the sample complexity of this approach follows from a Hoeffding and union bound.

Blum et al. [26] study a data-driven approach to learning a nearly optimal cooling schedule for the simulated annealing algorithm. They provide upper and lower sample complexity bounds, with their upper bound following from a careful covering number argument. We leave as an open question whether our techniques can be combined with theirs to match their sample complexity lower bound of $\tilde{\Omega}(\sqrt[3]{m})$, where m is the cooling schedule length.

Bartlett et al. [24] provide generalization bounds for learning over a family of low-rank approximation algorithms. Their analysis is based on a refined version of the Goldberg and Jerrum framework [55] for bounding the VC dimension of a function class based on the number of arithmetic operations and conditional statements required to compute the functions in the class. Their pseudo-dimension bound grows linearly with the number of algorithm parameters and—at a high level—logarithmically with the complexity of these arithmetic operations and conditional statements. This analysis allows them to prove a tighter bound than our main theorem would imply since in their setting, the VC dimension of the set $\mathcal{G}^*$ is exponential in the number of algorithm parameters, but using their analysis, they obtain a bound that grows only linearly with the number of parameters.

Machine learning for combinatorial optimization. A growing body of applied research has developed machine learning approaches to discrete optimization, largely intending to improve classic optimization algorithms such as branch-and-bound [e.g., 41, 46, 49, 80, 104, 113, 115, 116, 118, 124, 131]. For example, Chmiela et al. [31] present data-driven approaches to scheduling heuristics in branch-and-bound, and they leave as an open question whether the techniques in this article can be used to provide provable guarantees.

2 Notation and Problem Statement

We study algorithms parameterized by a set $\mathcal{P} \subseteq \mathbb{R}^d$. As a concrete example, parameterized algorithms are often used for sequence alignment [59]. There are many features of an alignment one might wish to optimize, such as the number of matches, mismatches, or indels (defined in Section 4.1). A parameterized objective function is defined by weighting these features. As another example, hierarchical clustering algorithms often use linkage routines such as single, complete, and average linkage. Parameters can be used to interpolate between these three classic procedures [16], which can be outperformed with a careful parameter tuning [6].

We use $\mathcal{X}$ to denote the set of problem instances the algorithm takes as input. We measure the performance of the algorithm parameterized by $\boldsymbol{\rho} = (\rho[1], \ldots, \rho[d]) \in \mathbb{R}^d$ via a utility function $u_{\boldsymbol{\rho}} : \mathcal{X} \to [0, H]$, with $\mathcal{U} = \{u_{\boldsymbol{\rho}} : \boldsymbol{\rho} \in \mathcal{P}\}$ denoting the set of all such functions. We assume there is an unknown, application-specific distribution $\mathcal{D}$ over $\mathcal{X}$.

Our goal is to find a parameter vector in $\mathcal{P}$ with high performance in expectation over the distribution $\mathcal{D}$. We provide *generalization guarantees* for this problem. Given a training set of problem instances $\mathcal{S}$ sampled from $\mathcal{D}$, a generalization guarantee bounds the difference—for any choice of the parameters $\boldsymbol{\rho}$—between the average performance of the algorithm over $\mathcal{S}$ and its expected performance.

Specifically, our main technical contribution is a bound on the *pseudo-dimension* [103] of the set $\mathcal{U}$. To define pseudo-dimension, we first define the related notion of *VC dimension* [120]. Let $\mathcal{H}$ be an arbitrary set of binary functions that map an abstract domain $\mathcal{Y}$ to $\{0, 1\}$. The *VC dimension* of $\mathcal{H}$, denoted $\text{VCdim}(\mathcal{H})$, is the size of the largest set $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ that is *shattered* by $\mathcal{H}$:

$$\left|\left\{\begin{pmatrix} h(y_1) \\ \vdots \\ h(y_N) \end{pmatrix} \middle| h \in \mathcal{H}\right\}\right| = 2^N.$$

Meanwhile, pseudo-dimension is a complexity measure for real-valued functions. In particular, let $\mathcal{H}$ be an arbitrary set of functions that map an abstract domain $\mathcal{Y}$ to $\mathbb{R}$. We convert these real-valued functions into binary-valued functions by defining the set of *below-the-graph indicator functions* $\mathcal{B}_{\mathcal{H}}$, which equals the set of all functions $b_h : \mathcal{Y} \times \mathbb{R} \to \{0, 1\}$ where $b_h(y, z) = \text{sign}(h(y) - z)$. Then the *pseudo-dimension* of $\mathcal{H}$, denoted $\text{Pdim}(\mathcal{H})$, is the VC dimension of $\mathcal{B}_{\mathcal{H}}$: $\text{Pdim}(\mathcal{H}) = \text{VCdim}(\mathcal{B}_{\mathcal{H}})$. In other words, $\text{Pdim}(\mathcal{H})$ is the size of the largest set $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ such that for some set of *targets* $z_1, \ldots, z_N \in \mathbb{R}$,

$$\left|\left\{\begin{pmatrix} \text{sign}(h(y_1) - z_1) \\ \vdots \\ \text{sign}(h(y_N) - z_N) \end{pmatrix} \middle| h \in \mathcal{H}\right\}\right| = 2^N. \tag{1}$$

Classic results from learning theory [103] translate pseudo-dimension bounds into generalization guarantees. For example, suppose $[0, H]$ is the range of the functions in $\mathcal{H}$. For any $\delta \in (0, 1)$ and any distribution $\mathcal{D}$ over $\mathcal{Y}$, with probability $1 - \delta$ over the draw of $\mathcal{S} \sim \mathcal{D}^N$, for all functions $h \in \mathcal{H}$, the difference between the average value of h over $\mathcal{S}$ and its expected value is bounded as follows:

$$\left|\frac{1}{N}\sum_{y \in \mathcal{S}} h(y) - \mathop{\mathbb{E}}_{y \sim \mathcal{D}}[h(y)]\right| = O\left(H\sqrt{\frac{1}{N}\left(\text{Pdim}(\mathcal{H}) + \ln\frac{1}{\delta}\right)}\right). \tag{2}$$

When $\mathcal{H}$ is a set of binary-valued functions that map $\mathcal{Y}$ to $\{0, 1\}$, the pseudo-dimension of $\mathcal{H}$ is more commonly referred to as the *VC-dimension of* $\mathcal{H}$ [120], denoted $\text{VCdim}(\mathcal{H})$.

3 Generalization Guarantees

When tuning an algorithm's parameters, there are two closely related function classes. First, for each parameter setting $\boldsymbol{\rho} \in \mathcal{P}$, $u_{\boldsymbol{\rho}} : \mathcal{X} \to \mathbb{R}$ measures performance as a function of the input $x \in \mathcal{X}$. Similarly, for each input x, there is a function $u_x : \mathcal{P} \to \mathbb{R}$ defined as $u_x(\boldsymbol{\rho}) = u_{\boldsymbol{\rho}}(x)$ that measures performance as a function of the parameter vector $\boldsymbol{\rho}$. The set $\{u_x \mid x \in \mathcal{X}\}$ is equivalent to Assouad's notion of the *dual class* [5].

Definition 3.1 (Dual Class [5]). 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 $\mathcal{H}^* = \{h^*_y : \mathcal{H} \to \mathbb{R} \mid y \in \mathcal{Y}\}$ where $h^*_y(h) = h(y)$. Each function $h^*_y \in \mathcal{H}^*$ 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 $\{u_x \mid x \in \mathcal{X}\}$ is equivalent to the dual class $\mathcal{U}^* = \{u^*_x : \mathcal{U} \to [0, H] \mid x \in \mathcal{X}\}$ in the sense that for every parameter vector $\boldsymbol{\rho} \in \mathcal{P}$ and every problem instance $x \in \mathcal{X}$, $u_x(\boldsymbol{\rho}) = u^*_x(u_{\boldsymbol{\rho}})$.

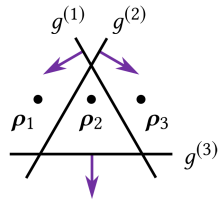


Fig. 2. Boundary functions partitioning $\mathbb{R}^2$. The arrows indicate on which side of each function $g^{(i)}(\boldsymbol{\rho}) = 0$ and on which side $g^{(i)}(\boldsymbol{\rho}) = 1$. For example, $g^{(1)}(\boldsymbol{\rho}_1) = 1$, $g^{(1)}(\boldsymbol{\rho}_2) = 1$, and $g^{(1)}(\boldsymbol{\rho}_3) = 0$.

Many combinatorial algorithms share a clear-cut, useful structure: for each instance $x \in \mathcal{X}$, the function u_x is *piecewise structured*. For example, each function u_x might be piecewise constant with a small number of pieces. Given the equivalence of the functions $\{u_x \mid x \in \mathcal{X}\}$ 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 2 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 1 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. The following definition captures this structure.

Definition 3.2. 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 $\boldsymbol{b}_y = (g^{(1)}(y), \ldots, g^{(k)}(y)) \in \{0, 1\}^k$.

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 3.3. *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:*

$$\text{Pdim}(\mathcal{U}) = O\Big(\big(\text{Pdim}(\mathcal{F}^*) + \text{VCdim}(\mathcal{G}^*)\big) \ln \big(\text{Pdim}(\mathcal{F}^*) + \text{VCdim}(\mathcal{G}^*)\big) + \text{VCdim}(\mathcal{G}^*) \ln k\Big).$$

To help make the proof of Theorem 3.3 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 3.4 bounds the number

of binary vectors

$$(h_1(y), \ldots, h_N(y)), \tag{3}$$

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 2 for example, Lemma 3.4 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 [112] to prove Lemma 3.4. Sauer's lemma [112] 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 $(eN)^{\text{VCdim}(\mathcal{H})}$ [e.g., 4, Theorem 3.7]. Therefore, it does not immediately imply a bound on the number of vectors from Equation (3). To apply Sauer's lemma, we must transition to the dual class.

Lemma 3.4. *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:*

$$|\{(h_1(y), \ldots, h_N(y)) \mid y \in \mathcal{Y}\}| \leq (eN)^{\text{VCdim}(\mathcal{H}^*)}. \tag{4}$$

Proof. We rewrite the left-hand-side of Equation (4) as $|\{(h_y^*(h_1), \ldots, h_y^*(h_N)) \mid y \in \mathcal{Y}\}|$. Since we fix N inputs and vary the function h_y^*, the lemma statement follows from Sauer's lemma [112]. □

We now prove Theorem 3.3.

Proof of Theorem 3.3. Fix an arbitrary set of problem instances $x_1, \ldots, x_N \in \mathcal{X}$ and targets $z_1, \ldots, z_N \in \mathbb{R}$. We bound the number of ways that $\mathcal{U}$ can label the problem instances $x_1, \ldots, x_N$ with respect to the target thresholds $z_1, \ldots, z_N \in \mathbb{R}$. In other words, as per Equation (1), we bound the size of the set

$$\left|\left\{\begin{pmatrix} \text{sign}\left(u_{\boldsymbol{\rho}}(x_1) - z_1\right) \\ \vdots \\ \text{sign}\left(u_{\boldsymbol{\rho}}(x_N) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}\right\}\right| = \left|\left\{\begin{pmatrix} \text{sign}\left(u_{x_1}^*(u_{\boldsymbol{\rho}}) - z_1\right) \\ \vdots \\ \text{sign}\left(u_{x_N}^*(u_{\boldsymbol{\rho}}) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}\right\}\right|, \tag{5}$$

by $(ekN)^{\text{VCdim}(\mathcal{G}^*)}(eN)^{\text{Pdim}(\mathcal{F}^*)}$. Then solving for the largest N such that

$$2^N \leq (ekN)^{\text{VCdim}(\mathcal{G}^*)}(eN)^{\text{Pdim}(\mathcal{F}^*)},$$

gives a bound on the pseudo-dimension of $\mathcal{U}$. Our bound on Equation (5) has two main steps:

(1) In Claim 3.5, we show that there are $M < (ekN)^{\text{VCdim}(\mathcal{G}^*)}$ subsets $\mathcal{P}_1, \ldots, \mathcal{P}_M$ partitioning the parameter space $\mathcal{P}$ such that within any one subset, the dual functions $u_{x_1}^*, \ldots, u_{x_N}^*$ 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 $u_{x_i}^*(u_{\boldsymbol{\rho}}) = f_i(u_{\boldsymbol{\rho}})$ 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 $u_{x_1}^*, \ldots, u_{x_N}^*$.

(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 $x_1, \ldots, x_N$ in at most $(eN)^{\text{Pdim}(\mathcal{F}^*)}$ ways with respect to the target thresholds $z_1, \ldots, z_N$. It follows that the total number of ways that $\mathcal{U}$ can label the problem instances $x_1, \ldots, x_N$ is bounded by $(ekN)^{\text{VCdim}(\mathcal{G}^*)}(eN)^{\text{Pdim}(\mathcal{F}^*)}$.

We now prove the first claim.

Claim 3.5. *There are $M < (ekN)^{\text{VCdim}(\mathcal{G}^*)}$ subsets $\mathcal{P}_1, \ldots, \mathcal{P}_M$ partitioning the parameter space $\mathcal{P}$ such that within any one subset, the dual functions $u_{x_1}^*, \ldots, u_{x_N}^*$ 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 $u^*_{x_i}(u_{\boldsymbol{\rho}}) = f_i(u_{\boldsymbol{\rho}})$ for all parameter settings $\boldsymbol{\rho} \in \mathcal{P}_j$ and $i \in [N]$.

Proof of Claim 3.5. Let $u^*_{x_1}, \ldots, u^*_{x_N} \in \mathcal{U}^*$ be the dual functions corresponding to the problem instances $x_1, \ldots, x_N$. Since $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable, we know that for each function $u^*_{x_i}$, 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 $\hat{\mathcal{G}} = \bigcup_{i=1}^{N} \{g_i^{(1)}, \ldots, g_i^{(k)}\}$ 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 $\mathcal{U}$:

$$M := \left| \left\{ \begin{pmatrix} g_1(u_{\boldsymbol{\rho}}) \\ \vdots \\ g_{kN}(u_{\boldsymbol{\rho}}) \end{pmatrix} : \boldsymbol{\rho} \in \mathcal{P} \right\} \right|. \tag{6}$$

By Lemma 3.4, $M < (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}$. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ be the binary vectors in the set from Equation (6). For each $i \in [M]$, let $\mathcal{P}_j = \{\boldsymbol{\rho} \mid (g_1(u_{\boldsymbol{\rho}}), \ldots, g_{kN}(u_{\boldsymbol{\rho}})) = \boldsymbol{b}_j\}$. By construction, for each set $\mathcal{P}_j$, the values of all the boundary functions $g_1(u_{\boldsymbol{\rho}}), \ldots, g_{kN}(u_{\boldsymbol{\rho}})$ 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 $u^*_{x_i}(u_{\boldsymbol{\rho}}) = f_i(u_{\boldsymbol{\rho}})$ for all parameter vectors $\boldsymbol{\rho} \in \mathcal{P}_j$ and indices $i \in [N]$. Therefore, the claim holds. □

Claim 3.5 and Equation (5) imply that for every subset $\mathcal{P}_j$ of the partition,

$$\left| \left\{ \begin{pmatrix} \mathrm{sign}\left(u_{\boldsymbol{\rho}}(x_1) - z_1\right) \\ \vdots \\ \mathrm{sign}\left(u_{\boldsymbol{\rho}}(x_N) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right| = \left| \left\{ \begin{pmatrix} \mathrm{sign}\left(f_1(u_{\boldsymbol{\rho}}) - z_1\right) \\ \vdots \\ \mathrm{sign}\left(f_N(u_{\boldsymbol{\rho}}) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right|.$$

Switching to the dual class as in Lemma 3.4, we have that

$$\left| \left\{ \begin{pmatrix} \mathrm{sign}\left(u_{\boldsymbol{\rho}}(x_1) - z_1\right) \\ \vdots \\ \mathrm{sign}\left(u_{\boldsymbol{\rho}}(x_N) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right| = \left| \left\{ \begin{pmatrix} \mathrm{sign}\left(f^*_{u_{\boldsymbol{\rho}}}(f_1) - z_1\right) \\ \vdots \\ \mathrm{sign}\left(f^*_{u_{\boldsymbol{\rho}}}(f_N) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right|.$$

By definition of the class of below-the-graph indicator functions from Section 2, we have that

$$\left| \left\{ \begin{pmatrix} \mathrm{sign}\left(u_{\boldsymbol{\rho}}(x_1) - z_1\right) \\ \vdots \\ \mathrm{sign}\left(u_{\boldsymbol{\rho}}(x_N) - z_N\right) \end{pmatrix} \middle| \boldsymbol{\rho} \in \mathcal{P}_j \right\} \right| \leq (eN)^{\mathrm{VCdim}(B_{\mathcal{F}^*})} = (eN)^{\mathrm{Pdim}(\mathcal{F}^*)}. \tag{7}$$

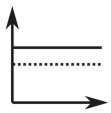
In other words, for any region $\mathcal{P}_j$ of the partition, as we vary the parameter vector $\boldsymbol{\rho} \in \mathcal{P}_j$, $u_{\boldsymbol{\rho}}$ can label the problem instances $x_1, \ldots, x_N$ in at most $(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$ ways with respect to the target thresholds $z_1, \ldots, z_N$. Because there are $M < (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}$ regions $\mathcal{P}_j$ of the partition, we can conclude that $\mathcal{U}$ can label the problem instances $x_1, \ldots, x_N$ in at most $(ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$ distinct ways relative to the targets $z_1, \ldots, z_N$. In other words, Equation (5) is bounded by

$$(ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}.$$

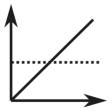
On the other hand, if $\mathcal{U}$ shatters the problem instances $x_1, \ldots, x_N$, then the number of distinct labelings must be 2^N. Therefore, the pseduo-dimension of $\mathcal{U}$ is at most the largest value of N such that $2^N \leq (ekN)^{\mathrm{VCdim}(\mathcal{G}^*)}(eN)^{\mathrm{Pdim}(\mathcal{F}^*)}$, which implies that

$$N = O\left((\mathrm{Pdim}(\mathcal{F}^*) + \mathrm{VCdim}(\mathcal{G}^*)) \ln\left(\mathrm{Pdim}(\mathcal{F}^*) + \mathrm{VCdim}(\mathcal{G}^*)\right) + \mathrm{VCdim}(\mathcal{G}^*) \ln k\right),$$

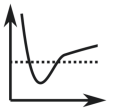
as claimed. □



(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).

Fig. 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.

We describe several lower bounds which show that Theorem 3.3 is tight up to logarithmic factors.

Theorem 3.6. *There is a parameterized sequence alignment algorithm with* $\text{Pdim}(\mathcal{U}) = \Omega(\log n)$ *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* $\text{Pdim}(\mathcal{F}^*) = \text{VCdim}(\mathcal{G}^*) = 1$.

Proof. In Theorem 4.5, 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. □

Moreover, as we describe in Section 5.4, there are several function classes $\mathcal{U}$ from economic mechanism design whose duals $\mathcal{U}^*$ are piecewise decomposable with piece functions $\mathcal{F}$ and boundary functions $\mathcal{G}$ where $\text{Pdim}(\mathcal{U}) = \Omega(\text{Pdim}(\mathcal{F}^*) + \text{VCdim}(\mathcal{G}^*))$ [13].

Applications of our main theorem to representative function classes

We now instantiate Theorem 3.3 in a general setting inspired by structure that many algorithm families exhibit.

One-dimensional functions with a bounded number of oscillations. 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.

Definition 3.7. 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 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 3.8. *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 $z_1, \ldots, z_N \in \mathbb{R}$ such that for every subset $T \subseteq [N]$, there exists a parameter setting $\rho \in \mathbb{R}$ such that $h_\rho^*(h_i) \geq z_i$ 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 $h_i(\rho) \geq z_i$ 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:

$$\left|\left\{\begin{pmatrix}\mathbb{I}_{\{h_1(\rho)\geq z_1\}}\\ \vdots \\ \mathbb{I}_{\{h_N(\rho)\geq z_N\}}\end{pmatrix} : \rho \in \mathcal{P}^*\right\}\right| = 2^N.$$

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 $\rho \mapsto \mathbb{I}_{\{h_i(\rho)\geq z_i\}}$ 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 because otherwise, if $\rho, \rho' \in I_j \cap \mathcal{P}^*$, then

$$\begin{pmatrix}\mathbb{I}_{\{h_1(\rho)\geq z_1\}}\\ \vdots \\ \mathbb{I}_{\{h_N(\rho)\geq z_N\}}\end{pmatrix} = \begin{pmatrix}\mathbb{I}_{\{h_1(\rho')\geq z_1\}}\\ \vdots \\ \mathbb{I}_{\{h_N(\rho')\geq z_N\}}\end{pmatrix},$$

which is a contradiction since each parameter setting in $\mathcal{P}^*$ induces a distinct binary vector. As a result, $2^N \leq BN + 1$. The lemma then follows from Lemma A.1. □

Lemma 3.8 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 3.9. *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 3.8, 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 $z_1, \ldots, z_{N+1} \in \mathbb{R}$. By definition, this means that

$$\left|\left\{\begin{pmatrix}\mathbb{I}_{\left\{f^*_{u_\rho}(f_1)\geq z_1\right\}}\\ \vdots \\ \mathbb{I}_{\left\{f^*_{u_\rho}(f_{N+1})\geq z_{N+1}\right\}}\end{pmatrix} : \rho \in \mathcal{P}\right\}\right| = 2^{N+1}.$$

For any function $f \in \mathcal{F}$ and any parameter setting $\rho \in \mathbb{R}$, $f^*_{u_\rho}(f) = f\left(u_\rho\right) = h_f(\rho) = h^*_\rho(h_f)$. Therefore,

$$\left|\left\{\begin{pmatrix}\mathbb{I}_{\left\{h^*_\rho(h_{f_1})\geq z_1\right\}}\\ \vdots \\ \mathbb{I}_{\left\{h^*_\rho(h_{f_{N+1}})\geq z_{N+1}\right\}}\end{pmatrix} : \rho \in \mathcal{P}\right\}\right| = \left|\left\{\begin{pmatrix}\mathbb{I}_{\left\{f^*_{u_\rho}(f_1)\geq z_1\right\}}\\ \vdots \\ \mathbb{I}_{\left\{f^*_{u_\rho}(f_{N+1})\geq z_{N+1}\right\}}\end{pmatrix} : \rho \in \mathcal{P}\right\}\right| = 2^{N+1},$$

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 3.3. □

Multi-dimensional piecewise-linear functions. For many algorithm families, we find that the boundary functions correspond to halfspace thresholds and the piece functions correspond to constant or linear functions. We handle this case in the following lemma.

Lemma 3.10. *Let $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^d\}$ be a class of utility functions defined over a d-dimensional parameter space. Suppose the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable, where the boundary functions $\mathcal{G} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ are halfspace indicator functions $g_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a}\cdot\boldsymbol{\rho} \le \theta\}}$ and the piece functions $\mathcal{F} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \mathbb{R} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ are linear functions $f_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \boldsymbol{a} \cdot \boldsymbol{\rho} + \theta$. Then* $\text{Pdim}(\mathcal{U}) = O(d \ln(dk))$.

The proof of this lemma follows from classic VC- and pseudo-dimension bounds for linear functions and can be found in Appendix B.

4 Parameterized Computational Biology Algorithms

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.

In some settings, while the correct biological answer can be found with enough time and funding, it is both time and resource intensive, from the order of days to months or more to obtain correct results for each item rather than minutes to hours for prediction. In other cases, the computational prediction is the only feasible way to estimate the answer. With the correct parameter choices, biologists can have high confidence that the predictions made by the algorithms will be equivalent to—and thus can be used in lieu of—bench-based experiments.

The parameters used to fine-tune the prediction algorithms are meant to model biological processes and in many cases the input may not come from a single distribution. This is especially true when looking at rare diseases, and this is all the more reason why we want to have confidence that we have enough training examples from the distribution of interest. If, for instance, one were to study a rare disease, one would want to learn the model parameters that correctly match the biological process within that disease; that is, the distribution from which your samples come from is unique to the biological question at hand. The procedure we provide in this work determines how many examples one needs to properly learn a model that fits this distribution and how confident you can be in the model's prediction.

4.1 Sequence Alignment

4.1.1 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 Σ 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 $|\tau_1| = |\tau_2|$, $\texttt{del}(\tau_1) = S_1$, and $\texttt{del}(\tau_2) = S_2$, where $\texttt{del}$ 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 $A_{\boldsymbol{\rho}}$ [59, 123] returns the alignment L that maximizes the objective function

$$\rho[1] \cdot \ell_1(S_1, S_2, L) + \cdots + \rho[d] \cdot \ell_d(S_1, S_2, L), \tag{8}$$

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 et al. [63] 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 [111]. We then define $u_{\boldsymbol{\rho}}(S_1, S_2) = u(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2))$ to be the utility of the alignment returned by the algorithm $A_{\boldsymbol{\rho}}$.

In the following lemma, we prove that the set of utility functions $u_{\boldsymbol{\rho}}$ has piecewise-structured dual functions.

Lemma 4.1. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2)) \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$ that map sequence pairs $S_1, S_2 \in \Sigma^n$ to $\mathbb{R}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, 4^n n^{4n+2})$-piecewise decomposable, where $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$ and $\mathcal{G} = \{g_{\boldsymbol{a}} : \mathcal{U} \to \{0, 1\} \mid \boldsymbol{a} \in \mathbb{R}^d\}$ consists of halfspace indicator functions $g_{\boldsymbol{a}} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a} \cdot \boldsymbol{\rho} < 0\}}$.*

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, $\mathcal{L} = \{A_{\boldsymbol{\rho}}(S_1, S_2) \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$. In Lemma C.1, we prove that $|\mathcal{L}| \leq 2^n n^{2n+1}$. For any alignment $L \in \mathcal{L}$, the algorithm $A_{\boldsymbol{\rho}}$ will return L if and only if

$$\rho[1] \cdot \ell_1(S_1, S_2, L) + \cdots + \rho[d] \cdot \ell_d(S_1, S_2, L) > \rho[1] \cdot \ell_1(S_1, S_2, L') + \cdots + \rho[d] \cdot \ell_d(S_1, S_2, L'), \tag{9}$$

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 $\boldsymbol{\rho} \in R$. By definition of the dual, this means that $u^*_{S_1, S_2}(u_{\boldsymbol{\rho}}) = u_{\boldsymbol{\rho}}(S_1, S_2) = c_R$ as well.

We now use this structure to show that the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, 4^n n^{4n+2})$-piecewise decomposable, as per Definition 3.2. Recall that $\mathcal{G} = \{g_{\boldsymbol{a}} : \mathcal{U} \to \{0, 1\} \mid \boldsymbol{a} \in \mathbb{R}^d\}$ consists of halfspace indicator functions $g_{\boldsymbol{a}} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a} \cdot \boldsymbol{\rho} < 0\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$. For each pair of alignments $L, L' \in \mathcal{L}$, let $g^{(L, L')} \in \mathcal{G}$ correspond to the halfspace represented in Equation (9). Order these $k := \binom{|\mathcal{L}|}{2}$ functions arbitrarily as $g^{(1)}, \ldots, g^{(k)}$. Every connected component R of $\mathbb{R}^d \setminus \mathcal{H}$ corresponds to a sign pattern of the k hyperplanes. For a given region R, let $\boldsymbol{b}_R \in \{0, 1\}^k$ be the corresponding sign pattern. Define the function $f^{(\boldsymbol{b}_R)} \in \mathcal{F}$ as $f^{(\boldsymbol{b}_R)} = f_{c_R}$, so $f^{(\boldsymbol{b}_R)}(u_{\boldsymbol{\rho}}) = c_R$ for all $\boldsymbol{\rho} \in \mathbb{R}^d$. Meanwhile, for every vector $\boldsymbol{b}$ not corresponding to a sign pattern of the k hyperplanes, let $f^{(\boldsymbol{b})} = f_0$, so $f^{(\boldsymbol{b})}(u_{\boldsymbol{\rho}}) = 0$ for all $\boldsymbol{\rho} \in \mathbb{R}^d$. In this way, for every $\boldsymbol{\rho} \in \mathbb{R}^d$,

$$u^*_{S_1, S_2}(u_{\boldsymbol{\rho}}) = \sum_{\boldsymbol{b} \in \{0,1\}^k} \mathbb{I}_{\{g^{(i)}(u_{\boldsymbol{\rho}}) = b[i], \forall i \in [k]\}} f^{(\boldsymbol{b})}(u_{\boldsymbol{\rho}}),$$

as desired. □

Lemmas 3.10 and 4.1 imply the following corollary.

Corollary 4.2. *The pseudo-dimension of $\mathcal{U}$ is $O(nd \ln n + d \ln d)$.*

In Appendix C, we also prove tighter guarantees for a structured subclass of algorithms [59, 123]. In that case, $d = 4$ and $\ell_1(S_1, S_2, L)$ is the number of matches in the alignment, $\ell_2(S_1, S_2, L)$ is the number of mismatches, $\ell_3(S_1, S_2, L)$ is the number of indels, and $\ell_4(S_1, S_2, L)$ is the number of gaps. Building on prior research [47, 63, 101], we show (Lemma 4.3) that the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, O(n^3))$-piecewise decomposable with $\mathcal{F}$ and $\mathcal{G}$ defined as in Lemma 4.1. This implies a pseudo-dimension bound of $O(\ln n)$, which is significantly tighter than that of Lemma 4.1. We also prove that this pseudo-dimension bound is tight with a lower bound of $\Omega(\ln n)$ (Theorem 4.5). Moreover, in Appendix C.2, we provide guarantees for algorithms that align more than two sequences.

4.1.2 Tighter Guarantees for a Structured Algorithm Subclass: The Affine-gap Model. A line of prior work [47, 63, 101, 102] analyzed a specific instantiation of the objective function (8) where $d = 3$. In this case, we can obtain a pseudo-dimension bound of $O(\ln n)$, which is exponentially better than the bound implied by Lemma 4.1. Given a pair of sequences $S_1, S_2 \in \Sigma^n$, the dynamic programming algorithm $A_{\boldsymbol{\rho}}$ returns the alignment L that 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 $O(n^{3/2})$ different alignments the algorithm family $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^3\}$ might produce as we range over parameter vectors [47, 63, 101]. This bound is exponentially smaller than our generic bound of $4^n n^{4n+2}$ from Lemma C.1.

Lemma 4.3. *Let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u\left(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2)\right) \mid \boldsymbol{\rho} \in \mathbb{R}_{\geq 0}\right\},$$

that map sequence pairs $S_1, S_2 \in \Sigma^n$ to $\mathbb{R}$ under the affine gap model. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, O(n^3))$-piecewise decomposable, where $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$ and where $\mathcal{G} = \{g_{\boldsymbol{a}} : \mathcal{U} \to \{0, 1\} \mid \boldsymbol{a} \in \mathbb{R}\}$ consists of halfspace indicator functions $g_{\boldsymbol{a}} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{a[1]\rho[1] + a[2]\rho[2] + a[3]\rho[3] < a[4]\}}$.

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 [47, 63, 101], 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. The proof now follows by the exact same logic as that of Lemma 4.1. □

Lemmas 3.10 and 4.3 imply the following corollary.

Corollary 4.4. *The pseudo-dimension of $\mathcal{U}$ 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 [39]). 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:

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

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. The full proof of the following theorem is in Appendix C.

Theorem 4.5. *Under the affine gap model, 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* Σ *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* $\Omega(\log n)$*.*

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 $(S_1^{(1)}, S_2^{(1)})$ and $(S_1^{(2)}, S_2^{(2)})$, and ground-truth alignments of these pairs. We detail these elements below:

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

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

$$\begin{matrix} t_1^{(1)} = \mathsf{a}_1\mathsf{b}_1\mathsf{d}_1 \\ t_2^{(1)} = \mathsf{b}_1\mathsf{c}_1\mathsf{d}_1 \end{matrix}, \qquad \begin{matrix} t_1^{(2)} = \mathsf{a}_2\mathsf{a}_2\mathsf{b}_2\mathsf{d}_2 \\ t_2^{(2)} = \mathsf{b}_2\mathsf{c}_2\mathsf{c}_2\mathsf{d}_2 \end{matrix}, \quad \text{and} \quad \begin{matrix} t_1^{(3)} = \mathsf{a}_3\mathsf{a}_3\mathsf{a}_3\mathsf{b}_3\mathsf{d}_3 \\ t_2^{(3)} = \mathsf{b}_3\mathsf{c}_3\mathsf{c}_3\mathsf{c}_3\mathsf{d}_3 \end{matrix}. \tag{10}$$

We define the two sequence pairs as

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

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

$$\begin{matrix} \mathsf{a}_1 & \mathsf{b}_1 & - & \mathsf{d}_1 & \mathsf{a}_2 & \mathsf{a}_2 & \mathsf{b}_2 & - & - & \mathsf{d}_2 & \mathsf{a}_3 & \mathsf{a}_3 & \mathsf{a}_3 & \mathsf{b}_3 & - & - & - & \mathsf{d}_3 \\ \mathsf{b}_1 & - & \mathsf{c}_1 & \mathsf{d}_1 & - & - & \mathsf{b}_2 & \mathsf{c}_2 & \mathsf{c}_2 & \mathsf{d}_2 & \mathsf{b}_3 & - & - & - & \mathsf{c}_3 & \mathsf{c}_3 & \mathsf{c}_3 & \mathsf{d}_3 \end{matrix}. \tag{11}$$

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

$$\begin{matrix} \mathsf{a}_2 & \mathsf{a}_2 & \mathsf{b}_2 & - & - & \mathsf{d}_2 \\ - & - & \mathsf{b}_2 & \mathsf{c}_2 & \mathsf{c}_2 & \mathsf{d}_2 \end{matrix}.$$

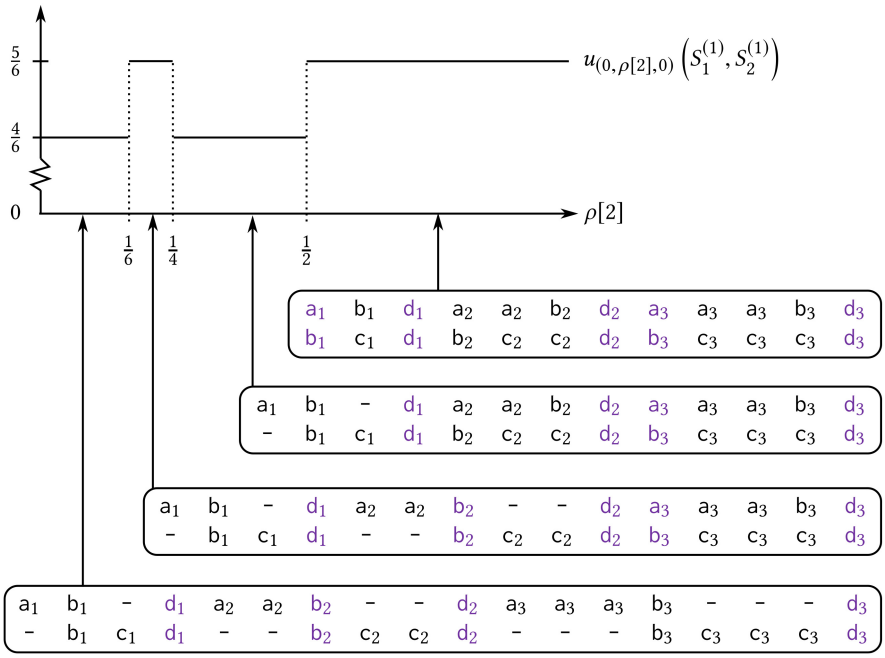


Fig. 4. The form of $u_{(0,\rho[2],0)}(S_1^{(1)}, S_2^{(1)})$ as a function of the indel parameter $\rho[2]$. When $\rho[2] \leq \frac{1}{6}$, the algorithm returns the bottom alignment. When $\frac{1}{6} < \rho[2] \leq \frac{1}{4}$, the algorithm returns the alignment that is second to the bottom. When $\frac{1}{4} < \rho[2] \leq \frac{1}{2}$, 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 (11)).

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

$$u_{(0,\rho[2],0)}\left(S_1^{(1)}, S_2^{(1)}\right) = \begin{cases} \frac{4}{6} & \text{if } \rho[2] \leq \frac{1}{6} \\ \frac{5}{6} & \text{if } \frac{1}{6} < \rho[2] \leq \frac{1}{4} \\ \frac{4}{6} & \text{if } \frac{1}{4} < \rho[2] \leq \frac{1}{2} \\ \frac{5}{6} & \text{if } \rho[2] > \frac{1}{2} \end{cases} \quad \text{and} \quad u_{(0,\rho[2],0)}\left(S_1^{(2)}, S_2^{(2)}\right) = \begin{cases} 1 & \text{if } \rho[2] \leq \frac{1}{4} \\ \frac{1}{2} & \text{if } \rho[2] > \frac{1}{4} \end{cases}. \tag{12}$$

The form of $u_{(0,\rho[2],0)}(S_1^{(1)}, S_2^{(1)})$ is illustrated by Figure 4. It is then straightforward to verify that the two sequence pairs are shattered by the parameter settings $(0, 0, 0)$, $(0, \frac{1}{5}, 0)$, $(0, \frac{1}{3}, 0)$, 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 $\{0, \frac{1}{5}, \frac{1}{3}, 1\}$.

Proof sketch of Equation (12). The full proof that Equation (12) holds follows the following high-level reasoning:

(1) First, we prove that under the algorithm's output alignment, the 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 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 $\text{MT}(S_1^{(1)}, S_2^{(1)}, L) - \rho[2] \cdot \text{ID}(S_1^{(1)}, S_2^{(1)}, L)$, 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 b_j characters are matched, so the characters that are correctly aligned (as per the ground-truth alignment (Equation (11))) in the algorithm's output are $(\mathsf{a}_1, \mathsf{b}_1)$, $(\mathsf{d}_1, \mathsf{d}_1)$, $(\mathsf{d}_2, \mathsf{d}_2)$, $(\mathsf{a}_3, \mathsf{b}_3)$, and $(\mathsf{d}_3, \mathsf{d}_3)$, as illustrated by purple in the top alignment of Figure 4. 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, $u_{(0,\rho[2],0)}(S_1^{(1)}, S_2^{(1)}) = \frac{5}{6}$.

When $\rho[2]$ shifts to the next-smallest interval $(\frac{1}{4}, \frac{1}{2}]$, the indel penalty $\rho[2]$ is sufficiently small that the b_1 characters will align. Thus we lose the correct alignment $(\mathsf{a}_1, \mathsf{b}_1)$, and the Q score drops to $\frac{4}{6}$. Similarly, if we decrease $\rho[2]$ to the next-smallest interval $(\frac{1}{6}, \frac{1}{4}]$, the b_2 characters will align, which is correct under the ground-truth alignment (Equation (11)). 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 $(\mathsf{a}_3, \mathsf{b}_3)$ in favor of the alignment of the b_3 characters, so the Q score falls to $\frac{4}{6}$. In this way, we prove the form of $u_{(0,\rho[2],0)}(S_1^{(1)}, S_2^{(1)})$ from Equation (12). A parallel argument proves the form of $u_{(0,\rho[2],0)}(S_1^{(2)}, S_2^{(2)})$.

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 (10). 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 $u_{(0,\rho[2],0)}(S_1^{(1)}, S_2^{(1)})$ 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 $u_{(0,\rho[2],0)}(S_1^{(1)}, S_2^{(1)})$ to oscillate half as many times, and so on. This construction allows us to shatter $\Theta(\log n)$ sequences. □

In Appendix C.2, we provide guarantees for algorithms that align more than two sequences.

4.2 RNA Folding

RNA molecules have many essential roles including protein coding and enzymatic functions [67]. 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 function and how it may be affected by small changes in the sequence. Given a sequence $S \in \{\mathsf{A}, \mathsf{U}, \mathsf{C}, \mathsf{G}\}^n$, 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 [100]. At a high level, this objective function tradesoff 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_ρ uses dynamic programming to return the folding $A_\rho(S)$ that maximizes

$$\rho\,|\phi| + (1-\rho) \sum_{(i,j)\in\phi} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi\}}, \tag{13}$$

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 sub-structures.

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 ϕ and a "ground-truth" folding, obtained via expensive computation or laboratory experiments. The full proof of the following lemma is in Appendix D.

Lemma 4.6. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho : S \mapsto u(S, A_\rho(S)) \mid \rho \in \mathbb{R}\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^2)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho<a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$.*

Proof sketch. We first prove that for any strand S, $A_\rho(S)$ will return at most $\frac{n}{2} + 1$ different foldings as we vary ρ. Every folding has length at most $\frac{n}{2}$, so for any $k \le \frac{n}{2}$, let ϕ_k be the folding of length k that maximizes the second summand of Equation (13). Due to the form of the objective function, $A_\rho(S) \in \{\phi_0, \ldots, \phi_{n/2}\}$.

We then prove that for any pair of foldings (ϕ, ϕ'), there are two intervals that partition $\mathbb{R}$ where for any ρ in one of the intervals, the objective function (Equation (13)) applied to ϕ is higher than that of ϕ', and in the other interval the opposite is true. This implies that there are $O\left(n^2\right)$ thresholds that split $\mathbb{R}$ into intervals where the optimal folding is constant. In any interval, the utility is also constant, so the lemma holds. □

Since constant functions have zero oscillations, Lemmas 3.9 and 4.6 imply the following corollary.

Corollary 4.7. *The pseudo-dimension of $\mathcal{U}$ is $O(\ln n)$.*

4.3 Prediction of 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. When these TAD boundaries change, the expression of nearby genes can be altered, which can trigger diseases such as congenital malformations and cancer [88]. The goal of predicting TAD boundary locations is to identify when these changes correlate with phenotypic changes (diseases, malformations, etc.), which would aid in the ultimate goal of preventing or reversing these phenotypic changes.

The contact frequency of any two genome locations, denoted by a matrix $M \in \mathbb{R}^{n \times n}$, can be measured via experiments [84]. A dynamic programming algorithm A_ρ introduced by Filippova et al. [48] returns the TAD set $A_\rho(M)$ that maximizes

$$\sum_{(i,j) \in T} s_\rho(i, j) - \mu_\rho(j - i), \tag{14}$$

where $\rho \ge 0$ is a parameter, $s_\rho(i, j) = \frac{1}{(j-i)^\rho} \sum_{i \le p < q \le j} M_{pq}$ is the scaled density of the subgraph induced by the interactions between genomic loci i and j, and $\mu_\rho(d) = \frac{1}{n-d} \sum_{t=0}^{n-d-1} s_\rho(t, t + d)$ is the mean value of s_ρ over all sub-matrices of length d along the diagonal of M. We note that unlike the sequence alignment and RNA folding algorithms, the parameter ρ 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. The full proof of the following lemma is in Appendix E.

Lemma 4.8. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho : M \mapsto u(M, A_\rho(M)) \mid \rho \in \mathbb{R}\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, 2n^2 4^{n^2})$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of*

threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho<a\}}$ *and* $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ *consists of constant functions* $f_c : u_\rho \mapsto c$.

Proof sketch. Fix a matrix $M \in \mathbb{R}^{n\times n}$. We rewrite Equation (14) as $\sum_{(i,j)\in T} \frac{c_{ij}}{(j-i)^\rho}$, where each c_{ij} is a constant. As we vary ρ, the set $A_\rho(M)$ will only change when $\sum_{(i,j)\in T} \frac{c_{ij}}{(j-i)^\rho} - \sum_{(i,j)\in T'} \frac{c_{ij}}{(j-i)^\rho} = 0$ for some pair T and T'. This equation has at most $2n^2$ solutions. Aggregating these $2n^2$ thresholds over all pairs T and T', there are at most $2n^2 4^{n^2}$ thresholds that split the parameters into intervals where the optimal TAD set is constant. In any interval, the utility function u is also constant, so the lemma statement holds. □

Since constant functions have zero oscillations, Lemmas 3.9 and 4.8 imply the following corollary.

Corollary 4.9. *The pseudo-dimension of* $\mathcal{U}$ *is* $O\left(n^2\right)$.

5 Connections to Prior Research

Theorem 3.3 also streamlines many existing guarantees for algorithm parameter tuning. As we describe in this section, these prior works proved that these algorithm families have structure which implies that their dual functions are piecewise-decomposable (though they did not use this language). Our main theorem then immediately implies generalization bounds for these algorithm families. In all of these cases, Theorem 3.3 implies generalization guarantees that match the existing bounds, but in many cases, our approach provides a more succinct proof.

(1) In Section 5.1, we analyze several parameterized clustering algorithms [16], which have piecewise-constant dual functions. These algorithms first run a linkage routine which builds a hierarchical tree of clusters. The parameters interpolate between the popular single, average, and complete linkage. The linkage routine is followed by a dynamic programming procedure that returns a clustering corresponding to a pruning of the hierarchical tree.
(2) Balcan et al. [6] study a family of linkage-based clustering algorithms where the parameters control the distance metric used for clustering in addition to the linkage routine. The algorithm family has two sets of parameters. The first set of parameters interpolate between linkage algorithms, while the second set interpolate between distance metrics. The dual functions are piecewise-constant with *quadratic* boundary functions. We recover their generalization bounds in Section 5.1.2.
(3) In Section 5.2, we analyze several integer programming algorithms, which have piecewise-constant and piecewise-inverse-quadratic dual functions (as in Figure 3(c)). The first is branch-and-bound, which is used by commercial solvers such as CPLEX. Branch-and-bound always finds an optimal solution and its parameters control runtime and memory usage. We also study semidefinite programming approximation algorithms for integer quadratic programming. We analyze a parameterized algorithm introduced by Feige and Langberg [44] which includes the Goemans-Williamson algorithm [53] as a special case. We recover previous generalization bounds in both settings [12, 16].
(4) Gupta and Roughgarden [62] introduced parameterized greedy algorithms for the knapsack and maximum weight independent set problems, which we show have piecewise-constant dual functions. We recover their generalization bounds in Section 5.3.
(5) We provide generalization bounds for parameterized selling mechanisms when the goal is to maximize revenue, which have piecewise-linear dual functions (as in Figure 3(b)). A long line of research has studied revenue maximization via machine learning [9, 29, 34, 36, 40, 56, 58, 61, 85, 86, 93, 95, 110]. In Section 5.4, we recover Balcan,

Sandholm, and Vitercik's generalization bounds [20] which apply to a variety of pricing, auction, and lottery mechanisms. They proved new bounds for mechanism classes not previously studied in the sample-based mechanism design literature and matched or improved over the best known guarantees for many classes.

5.1 Clustering Algorithms

A clustering instance is made up of a set points V from a data domain $\mathcal{X}$ and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$. The goal is to split up the points into groups, or "clusters," so that within each group, distances are minimized and between each group, distances are maximized. Typically, a clustering's quality is quantified by some objective function. Classic choices include the k-means, k-median, or k-center objective functions. Unfortunately, finding the clustering that minimizes any one of these objectives is NP-hard. Clustering algorithms have uses in data science, computational biology [98], and many other fields.

Balcan et al. [6, 16] analyze *agglomerative clustering algorithms*. This type of algorithm requires a merge function $c(A, B; d) \to \mathbb{R}_{\geq 0}$, defining the distances between point sets $A, B \subseteq V$. The algorithm constructs a *cluster tree*. This tree starts with n leaf nodes, each containing a point from V. Over a series of rounds, the algorithm merges the sets with minimum distance according to c. The tree is complete when there is one node remaining, which consists of the set V. The children of each internal node consist of the two sets merged to create the node. There are several common merge function c: $\min_{a\in A, b\in B} d(a, b)$ (single-linkage), $\frac{1}{|A|\cdot|B|}\sum_{a\in A, b\in B} d(a, b)$ (average-linkage), and $\max_{a\in A, b\in B} d(a, b)$ (complete-linkage). Following the linkage procedure, there is a dynamic programming step. This steps finds the tree pruning that minimizes an objective function, such as the k-means, -median, or -center objectives.

To evaluate the quality of a clustering, we assume access to a utility function $u : \mathcal{T} \to [-1, 1]$ where $\mathcal{T}$ is the set of all cluster trees over the data domain $\mathcal{X}$. For example, $u(T)$ might measure the distance between the ground truth clustering and the optimal k-means pruning of the cluster tree $T \in \mathcal{T}$.

In Section 5.1.1, we present results for learning merge functions and in Section 5.1.2, we present results for learning distance functions in addition to merge functions. The latter set of results apply to a special subclass of merge functions called *two-point-based* (as we describe in Section 5.1.2), and thus do not subsume the results in Section 5.1.1, but do apply to the more general problem of learning a distance function in addition to a merge function.

5.1.1 *Learning Merge Functions.*

Balcan et al. [16] study several families of merge functions:

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

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

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

The classes $\mathcal{C}_1$ and $\mathcal{C}_2$ interpolate between single- ($c_{1,-\infty}$ and $c_{2,1}$) and complete-linkage ($c_{1,\infty}$ and $c_{2,0}$). The class $\mathcal{C}_3$ includes as special cases average-, complete-, and single-linkage.

For each class $i \in \{1, 2, 3\}$ and each parameter ρ, let $A_{i,\rho}$ be the algorithm that takes as input a clustering instance (V, d) and returns a cluster tree $A_{i,\rho}(V, d) \in \mathcal{T}$.

Balcan et al. [16] prove the following useful structure about the classes $\mathcal{C}_1$ and $\mathcal{C}_2$:

Lemma 5.1 ([16]). *Let (V, d) be an arbitrary clustering instance over n points. There is a partition of $\mathbb{R}$ into $k \leq n^8$ intervals $I_1, \ldots, I_k$ such that for any interval I_j and any two parameters $\rho, \rho' \in I_j$, the sequences of merges the agglomerative clustering algorithm makes using the merge functions $c_{1,\rho}$ and $c_{1,\rho'}$ are identical. The same holds for the set of merge functions $\mathcal{C}_2$.*

This structure immediately implies that the corresponding class of utility functions has a piecewise-structured dual class.

Corollary 5.2. *Let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{u_\rho : (V, d) \mapsto u\left(A_{1,\rho}(V, d)\right) \mid \rho \in \mathbb{R} \cup \{-\infty, \infty\}\right\}$$

mapping clustering instances (V, d) to $[-1, 1]$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^8)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0, 1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho < a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$. The same holds when $\mathcal{U}$ is defined according to merge functions in $\mathcal{C}_2$ as $\mathcal{U} = \left\{u_\rho : (V, d) \mapsto u\left(A_{2,\rho}(V, d)\right) \mid \rho \in [0, 1]\right\}$.

Lemma 3.9 and Corollary 5.2 imply the following pseudo-dimension bound.

Corollary 5.3. *Let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{u_\rho : (V, d) \mapsto u\left(A_{1,\rho}(V, d)\right) \mid \rho \in \mathbb{R} \cup \{-\infty, \infty\}\right\},$$

mapping clustering instances (V, d) to $[-1, 1]$. Then $\operatorname{Pdim}(\mathcal{U}) = O(\ln n)$. The same holds when $\mathcal{U}$ is defined according to merge functions in $\mathcal{C}_2$ as $\mathcal{U} = \left\{u_\rho : (V, d) \mapsto u\left(A_{2,\rho}(V, d)\right) \mid \rho \in [0, 1]\right\}$.

Balcan et al. [16] prove a similar guarantee for the more complicated class $\mathcal{C}_3$.

Lemma 5.4 ([16]). *Let (V, d) be an arbitrary clustering instance over n points. There is a partition of $\mathbb{R}$ into $k \leq n^2 3^{2n}$ intervals $I_1, \ldots, I_k$ such that for any interval I_j and any two parameters $\rho, \rho' \in I_j$, the sequences of merges the agglomerative clustering algorithm makes using the merge functions $c_{3,\rho}$ and $c_{3,\rho'}$ are identical.*

Again, this structure immediately implies that the corresponding class of utility functions has a piecewise-structured dual class.

Corollary 5.5. *Let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{u_\rho : (V, d) \mapsto u\left(A_{3,\rho}(V, d)\right) \mid \rho \in \mathbb{R} \cup \{-\infty, \infty\}\right\},$$

mapping clustering instances (V, d) to $[-1, 1]$. The dual class $\mathcal{U}^*$ is $\left(\mathcal{F}, \mathcal{G}, n^2 3^{2n}\right)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0, 1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho < a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$.

Lemma 3.9 and Corollary 5.5 imply the following pseudo-dimension bound.

Corollary 5.6. *Let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{u_\rho : (V, d) \mapsto u\left(A_{3,\rho}(V, d)\right) \mid \rho \in \mathbb{R} \cup \{-\infty, \infty\}\right\},$$

mapping clustering instances (V, d) to $[-1, 1]$. Then $\operatorname{Pdim}(\mathcal{U}) = O(n)$.

Corollaries 5.3 and 5.6 match the pseudo-dimension guarantees that Balcan et al. [16] prove.

5.1.2 Learning Merge Functions and Distance Functions. Balcan et al. [6] extend the clustering generalization bounds of Balcan et al. [16] to the case of learning both a distance metric and a merge function. They introduce a family of linkage-based clustering algorithms that simultaneously interpolate between a collection of base metrics $d_1, \ldots, d_L$ and base merge functions $c_1, \ldots, c_L$. The algorithm family is parameterized by $\boldsymbol{\rho} = (\boldsymbol{\alpha}, \boldsymbol{\beta}) \in \Delta_{L'} \times \Delta_L$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are mixing weights for the merge functions and metrics, respectively. The algorithm with parameters $\boldsymbol{\rho} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ starts with each point in a cluster of its own and repeatedly merges the pair of clusters A and B minimizing $c_{\boldsymbol{\alpha}}(A, B; d_{\boldsymbol{\beta}})$, where

$$c_{\boldsymbol{\alpha}}(A, B; d) = \sum_{i=1}^{L'} \alpha_i \cdot c_i(A, B; d) \qquad \text{and} \qquad d_{\boldsymbol{\beta}}(a, b) = \sum_{i=1}^{L} \beta_i \cdot d_i(a, b).$$

We use the notation $A_{\boldsymbol{\rho}}$ to denote the algorithm that takes as input a clustering instance (V, d) and returns a cluster tree $A_{\boldsymbol{\rho}}(V, d) \in \mathcal{T}$ using the merge function $c_{\boldsymbol{\alpha}}(A, B; d_{\boldsymbol{\beta}})$, where $\boldsymbol{\rho} = (\boldsymbol{\alpha}, \boldsymbol{\beta})$.

When analyzing this algorithm family, Balcan et al. [6] prove that the following piecewise-structure holds when all of the merge functions are *two-point-based*, which roughly requires that for any pair of clusters A and B, there exist points $a \in A$ and $b \in B$ such that $c(A, B; d) = d(a, b)$. Single- and complete-linkage are two-point-based, but average-linkage is not.

Lemma 5.7 ([6]). *For any clustering instance V, there exists a collection of $O\left(|V|^{4L'}\right)$ quadratic boundary functions that partition the $(L + L')$-dimensional parameter space into regions where the algorithm's output is constant on each region in the partition.*

This lemma immediately implies that the corresponding class of utility functions has a piecewise-structured dual class.

Corollary 5.8. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} : (V, d) \mapsto u(A_{\boldsymbol{\rho}}(V, d)) \mid \boldsymbol{\rho} \in \Delta_{L'} \times \Delta_L\}$ mapping clustering instances (V, d) to $[-1, 1]$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, O(|V|^{4L'}))$-piecewise decomposable, where $\mathcal{F}$ is the set of constant functions and $\mathcal{G}$ is the set of quadratic functions defined on $\Delta_{L'} \times \Delta_L$.*

Using the fact that $\text{VCdim}(\mathcal{G}^*) = O((L+L')^2)$, we obtain the following pseudo-dimension bound.

Corollary 5.9. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} : (V, d) \mapsto u(A_{\boldsymbol{\rho}}(V, d)) \mid \boldsymbol{\rho} \in \Delta_{L'} \times \Delta_L\}$ mapping clustering instances (V, d) to $[-1, 1]$. Then*

$$\text{Pdim}(\mathcal{U}) = O\left((L + L')^2 \log(L + L') + (L + L')^2 L' \log(n)\right).$$

This matches the generalization bound that Balcan et al. [6] prove.

5.2 Integer Programming

Several articles [12, 16] study algorithm configuration for both integer linear and integer quadratic programming, as we describe below.

Integer linear programming. In the context of integer linear programming, Balcan et al. [12] focus on **branch-and-bound** (**B&B**) [81], an algorithm for solving **mixed integer linear programs** (**MILPs**). A MILP is defined by a matrix $A \in \mathbb{R}^{m \times n}$, a vector $\boldsymbol{b} \in \mathbb{R}^m$, a vector $\boldsymbol{c} \in \mathbb{R}^n$, and a set of indices $I \subseteq [n]$. The goal is to find a vector $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{c} \cdot \boldsymbol{x}$ is maximized, $A\boldsymbol{x} \leq \boldsymbol{b}$, and for every index $i \in I$, x_i is constrained to be binary: $x_i \in \{0, 1\}$.

Branch-and-bound builds a search tree to solve an input MILP Q. At the root of the search tree is the original MILP Q. At each round, the algorithm chooses a leaf of the search tree, which

represents an MILP Q'. It does so using a *node selection policy*; common choices include depth- and best-first search. Then, it chooses an index $i \in I$ using a *variable selection policy*. It next *branches* on x_i: it sets the left child of Q' to be that same integer program, but with the additional constraint that $x_i = 0$, and it sets the right child of Q' to be that same integer program, but with the additional constraint that $x_i = 1$. The algorithm *fathoms* a leaf, which means that it never will branch on that leaf, if it can guarantee that the optimal solution does not lie along that path. The algorithm terminates when it has fathomed every leaf. At that point, we can guarantee that the best solution to Q found so far is optimal. See the article by Balcan et al. [12] for more details.

Balcan et al. [12] show how to learn variable selection policies. Specifically, they study *score-based variable selection policies*, defined below.

Definition 5.10 (Score-based Variable Selection Policy [12]). Let $\texttt{score}$ be a deterministic function that takes as input a partial search tree $\mathcal{T}$, a leaf Q of that tree, and an index i, and returns a real value $\texttt{score}(\mathcal{T}, Q, i) \in \mathbb{R}$. For a leaf Q of a tree $\mathcal{T}$, let $N_{\mathcal{T},Q}$ be the set of variables that have not yet been branched on along the path from the root of $\mathcal{T}$ to Q. A score-based variable selection policy selects the variable $\operatorname{argmax}_{x_i \in N_{\mathcal{T},Q}} \{\texttt{score}(\mathcal{T}, Q, i)\}$ to branch on at the node Q.

This type of variable selection policy is widely used [1, 51, 87]. See the article by Balcan et al. [12] for examples.

Given d arbitrary scoring rules $\texttt{score}_1, \ldots, \texttt{score}_d$, Balcan et al. [12] provide guidance for learning a linear combination $\rho[1]\texttt{score}_1 + \cdots + \rho[d]\texttt{score}_d$ that leads to small expected tree sizes. They assume that all aspects of the tree search algorithm except the variable selection policy, such as the node selection policy, are fixed. In their analysis, they prove the following lemma.

Lemma 5.11 ([12]). *Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules and let Q be an arbitrary MILP over n binary variables. Suppose we limit B&B to producing search trees of size τ. There is a set $\mathcal{H}$ of at most $n^{2(\tau+1)}$ hyperplanes such that for any connected component R of $[0,1]^d \setminus \mathcal{H}$, the search tree B&B builds using the scoring rule $\rho[1]\texttt{score}_1 + \cdots + \rho[d]\texttt{score}_d$ is invariant across all $(\rho[1], \ldots, \rho[d]) \in R$.*

This piecewise structure immediately implies the following guarantee.

Corollary 5.12. *Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules and let Q be an arbitrary MILP over n binary variables. Suppose we limit B&B to producing search trees of size τ. For each parameter vector $\boldsymbol{\rho} = (\rho[1], \ldots, \rho[d]) \in [0,1]^d$, let $u_{\boldsymbol{\rho}}(Q)$ be the size of the tree, divided by τ, that B&B builds using the scoring rule $\rho[1]\texttt{score}_1 + \cdots + \rho[d]\texttt{score}_d$ given Q as input. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho \mid \boldsymbol{\rho} \in [0,1]^d\}$ mapping MILPs to $[0,1]$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^{2(\tau+1)})$-piecewise decomposable, where $\mathcal{G} = \{g_{\boldsymbol{a},\theta} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ consists of halfspace indicator functions $g_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{\rho}\cdot\boldsymbol{a} \le \theta\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$.*

Corollary 5.12 and Lemma 3.10 imply the following pseudo-dimension bound.

Corollary 5.13. *Let $\texttt{score}_1, \ldots, \texttt{score}_d$ be d arbitrary scoring rules and let Q be an arbitrary MILP over n binary variables. Suppose we limit B&B to producing search trees of size τ. For each parameter vector $\boldsymbol{\rho} = (\rho[1], \ldots, \rho[d]) \in [0,1]^d$, let $u_{\boldsymbol{\rho}}(Q)$ be the size of the tree, divided by τ, that B&B builds using the scoring rule $\rho[1]\texttt{score}_1 + \cdots + \rho[d]\texttt{score}_d$ given Q as input. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \left\{u_\rho \mid \boldsymbol{\rho} \in [0,1]^d\right\}$ mapping MILPs to $[0,1]$. Then $\operatorname{Pdim}(\mathcal{U}) = O(d(\tau \ln(n) + \ln(d)))$.*

Corollary 5.13 matches the pseudo-dimension guarantee that Balcan et al. [12] prove.

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 $\mathscr{Z}$, the n-dimensional Gaussian distribution.
2: Solve the SDP (15) for the optimal embedding $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$.
3: Compute set of fractional assignments $r(\langle \boldsymbol{Z}, \boldsymbol{u}_1\rangle), \ldots, r(\langle \boldsymbol{Z}, \boldsymbol{u}_n\rangle)$.
4: For all $i \in [n]$, set x_i to 1 with probability $\frac{1}{2} + \frac{1}{2} \cdot r(\langle \boldsymbol{Z}, \boldsymbol{u}_i\rangle)$ and -1 with probability $\frac{1}{2} - \frac{1}{2} \cdot r(\langle \boldsymbol{Z}, \boldsymbol{u}_i\rangle)$.

Output: $x_1, \ldots, x_n$.

Integer quadratic programming. A diverse array of NP-hard problems, including max-2SAT, max-cut, and correlation clustering, can be characterized as integer quadratic programs (IQPs). An IQP is represented by a matrix $A \in \mathbb{R}^{n\times n}$. The goal is to find a set $X = \{x_1, \ldots, x_n\} \in \{-1, 1\}^n$ maximizing $\sum_{i,j\in[n]} a_{ij}x_ix_j$. The most-studied IQP approximation algorithms operate via an SDP relaxation:

$$\text{maximize} \sum_{i,j\in[n]} a_{ij}\langle \boldsymbol{u}_i, \boldsymbol{u}_j\rangle \qquad \text{subject to } \boldsymbol{u}_i \in S^{n-1}. \tag{15}$$

The approximation algorithm must transform, or "round," the unit vectors into a binary assignment of the variables $x_1, \ldots, x_n$. In the seminal *GW algorithm* [53], the algorithm projects the unit vectors onto a random vector $\boldsymbol{Z}$, which it draws from the n-dimensional Gaussian distribution, which we denote using $\boldsymbol{Z}$. If $\langle \boldsymbol{u}_i, \boldsymbol{Z}\rangle > 0$, it sets $x_i = 1$. Otherwise, it sets $x_i = -1$.

The GW algorithm's approximation ratio can sometimes be improved if the algorithm probabilistically assigns the binary variables. In the final step, the algorithm can use any rounding function $r : \mathbb{R} \to [-1, 1]$ to set $x_i = 1$ with probability $\frac{1}{2} + \frac{1}{2} \cdot r(\langle \boldsymbol{Z}, \boldsymbol{u}_i\rangle)$ and $x_i = -1$ with probability $\frac{1}{2} - \frac{1}{2} \cdot r(\langle \boldsymbol{Z}, \boldsymbol{u}_i\rangle)$. 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 [44].

Balcan et al. [16] analyze s-linear rounding functions [44] $\phi_s : \mathbb{R} \to [-1, 1]$, parameterized by $s > 0$, defined as follows:

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

The goal is to learn a parameter s such that in expectation, $\sum_{i,j\in[n]} a_{ij}x_ix_j$ is maximized. The expectation is over several sources of randomness: first, the distribution $\mathcal{D}$ over matrices A; second, the vector $\boldsymbol{Z}$; and third, the assignment of $x_1, \ldots, x_n$. This final assignment depends on the parameter s, the matrix A, and the vector $\boldsymbol{Z}$. Balcan et al. [16] refer to this value as the *true utility of the parameter s*. Note that the distribution over matrices, which defines the algorithm's input, is unknown and external to the algorithm, whereas the Gaussian distribution over vectors as well as the distribution defining the variable assignment are internal to the algorithm.

The distribution over matrices is unknown, so we cannot know any parameter's true utility. Therefore, to learn a good parameter s, we must use samples. Balcan et al. [16] suggest drawing samples from two sources of randomness: the distributions over vectors and matrices. In other words, they suggest drawing a set of samples $\mathcal{S} = \{(A^{(1)}, \boldsymbol{Z}^{(1)}), \ldots, (A^{(m)}, \boldsymbol{Z}^{(m)})\} \sim (\mathcal{D} \times \mathscr{Z})^m$. Given these samples, Balcan et al. [16] define a parameter's *empirical utility* to be the expected objective value of the solution Algorithm 1 returns given input A, using the vector $\boldsymbol{Z}$ and ϕ_s in Step 3, on average over all $(A, \boldsymbol{Z}) \in \mathcal{S}$. Generally speaking, Balcan et al. [16] suggest sampling the first two randomness sources in order to isolate the third randomness source. They argue that this

third source of randomness has an expectation that is simple to analyze. Using pseudo-dimension, they prove that every parameter s, its empirical and true utilities converge.

A bit more formally, Balcan et al. [16] use the notation $p_{(i,Z,A,s)}$ to denote the distribution that the binary value x_i is drawn from when Algorithm 1 is given A as input and uses the rounding function $r = \phi_s$ and the hyperplane Z in Step 3. Using this notation, the parameter s has a true utility of

$$\mathop{\mathbb{E}}_{A,Z\sim\mathcal{D}\times\mathscr{Z}}\left[\mathop{\mathbb{E}}_{x_i\sim p_{(i,Z,A,s)}}\left[\sum_{i,j} a_{ij}x_ix_j\right]\right].^{2}$$

We also use the notation $u_s(A,Z)$ to denote the expected objective value of the solution Algorithm 1 returns given input A, using the vector Z and ϕ_s in Step 3. The expectation is over the final assignment of each variable x_i. Specifically, $u_s(A,Z) = \mathbb{E}_{x_i\sim p_{(i,Z,A,s)}}[\sum_{i,j} a_{ij}x_ix_j]$. By definition, a parameter's true utility equals $\mathbb{E}_{A,Z\sim\mathcal{D}\times\mathscr{Z}}[u_s(A,Z)]$. Given a set $(A^{(1)},Z^{(1)}),\ldots,(A^{(m)},Z^{(m)}) \sim \mathcal{D}\times\mathscr{Z}$, a parameter's empirical utility is $\frac{1}{m}\sum_{i=1}^{m} u_s(A^{(i)},Z^{(i)})$.

Both we and Balcan et al. [16] bound the pseudo-dimension of the function class $\mathcal{U} = \{u_s : s > 0\}$. Balcan et al. [16] prove that the functions in $\mathcal{U}$ are piecewise structured: roughly speaking, for a fixed matrix A and vector Z, each function in $\mathcal{U}$ is a piecewise, inverse-quadratic function of the parameter s. To present this lemma, we use the following notation: given a tuple (A,Z), let $u_{A,Z} : \mathbb{R}\to\mathbb{R}$ be defined such that $u_{A,Z}(s) = u_s(A,Z)$.

Lemma 5.14 ([16]). *For any matrix A and vector Z, the function $u_{A,Z} : \mathbb{R}_{>0}\to\mathbb{R}$ is made up of $n+1$ piecewise components of the form $\frac{a}{s^2}+\frac{b}{s}+c$ for some $a,b,c\in\mathbb{R}$. Moreover, if the border between two components falls at some $s\in\mathbb{R}_{>0}$, then it must be that $s = |\langle \boldsymbol{u}_i, Z\rangle|$ for some $\boldsymbol{u}_i$ in the optimal SDP embedding of A.*

This piecewise structure immediately implies the following corollary about the dual class $\mathcal{U}^*$.

Corollary 5.15. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_s : s > 0\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F},\mathcal{G},n)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U}\to\{0,1\} \mid a\in\mathbb{R}\}$ consists of threshold functions $g_a : u_s \mapsto \mathbb{I}_{\{s\le a\}}$ and $\mathcal{F} = \{f_{a,b,c} : \mathcal{U}\to\mathbb{R} \mid a,b,c\in\mathbb{R}\}$ consists of inverse-quadratic functions $f_{a,b,c} : u_s \mapsto \frac{a}{s^2}+\frac{b}{s}+c$.*

Lemma 3.9 and Corollary 5.15 imply the following pseudo-dimension bound.

Corollary 5.16. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_s : s > 0\}$. The pseudo-dimension of $\mathcal{U}$ is at most $O(\ln n)$.*

Corollary 5.16 matches the pseudo-dimension bound that Balcan et al. [16] prove.

5.3 Greedy Algorithms

Gupta and Roughgarden [62] provide pseudo-dimension bounds for greedy algorithm configuration, analyzing two canonical combinatorial problems: the maximum weight independent set problem and the knapsack problem. We recover their bounds in both cases.

[2]We, like Balcan et al. [16], use the abbreviated notation

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

Maximum weight independent set (MWIS). In the MWIS problem, the input is a graph G with a weight $w(v) \in \mathbb{R}_{\geq 0}$ per vertex v. The objective is to find a maximum-weight set of non-adjacent (or *independent*) vertices. On each iteration, the classic greedy algorithm adds the vertex v that maximizes $w(v)/(1+\deg(v))$ to the set. It then removes v and its neighbors from the graph. Given a parameter $\rho \geq 0$, Gupta and Roughgarden [62] propose the greedy heuristic $w(v)/(1+\deg(v))^{\rho}$. In this context, the utility function $u_{\rho}(G, w)$ equals the weight of the vertices in the set returned by the algorithm parameterized by ρ. Gupta and Roughgarden [62] implicitly prove the following lemma about each function u_{ρ} (made explicit in work by Balcan et al. [13]). To present this lemma, we use the following notation: let $u_{G,w} : \mathbb{R} \to \mathbb{R}$ be defined such that $u_{G,w}(\rho) = u_{\rho}(G, w)$.

Lemma 5.17 ([62]). *For any weighted graph (G, w), the function $u_{G,w} : \mathbb{R} \to \mathbb{R}$ is piecewise constant with at most n^4 discontinuities.*

This structure immediately implies that the function class $\mathcal{U} = \{u_{\rho} : \rho > 0\}$ has a piecewise-structured dual class.

Corollary 5.18. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\rho} : \rho > 0\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^4)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_{\rho} \mapsto \mathbb{I}_{\{\rho < a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\rho} \mapsto c$.*

Lemma 3.9 and Corollary 5.18 imply the following pseudo-dimension bound.

Corollary 5.19. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\rho} : \rho > 0\}$. The pseudo-dimension of $\mathcal{U}$ is $O(\ln n)$.*

This matches the pseudo-dimension bound by Gupta and Roughgarden [62].

Knapsack. Moving to the classic knapsack problem, the input is a knapsack capacity C and a set of n items i each with a value v_i and a size s_i. The goal is to determine a set $I \subseteq \{1, \ldots, n\}$ with maximium total value $\sum_{i \in I} v_i$ such that $\sum_{i \in I} s_i \leq C$. Gupta and Roughgarden [62] suggest the family of algorithms parameterized by $\rho > 0$ where each algorithm returns the better of the following two solutions:

— Greedily pack items in order of nonincreasing value v_i subject to feasibility.
— Greedily pack items in order of v_i / s_i^{ρ} subject to feasibility.

It is well-known that the algorithm with $\rho = 1$ achieves a 2-approximation. We use the notation $u_{\rho}(\boldsymbol{v}, \boldsymbol{s}, C)$ to denote the total value of the items returned by the algorithm parameterized by ρ given input $(\boldsymbol{v}, \boldsymbol{s}, C)$.

Gupta and Roughgarden [62] implicitly prove the following fact about the functions u_{ρ} (made explicit in work by Balcan et al. [13]). To present this lemma, we use the following notation: given a tuple $(\boldsymbol{v}, \boldsymbol{s}, C)$, let $u_{\boldsymbol{v},\boldsymbol{s},C} : \mathbb{R} \to \mathbb{R}$ be defined such that $u_{\boldsymbol{v},\boldsymbol{s},C}(\rho) = u_{\rho}(\boldsymbol{v}, \boldsymbol{s}, C)$.

Lemma 5.20 ([62]). *For any tuple $(\boldsymbol{v}, \boldsymbol{s}, C)$, the function $u_{\boldsymbol{v},\boldsymbol{s},C} : \mathbb{R} \to \mathbb{R}$ is piecewise constant with at most n^2 discontinuities.*

This structure immediately implies that the function class $\mathcal{U} = \{u_{\rho} : \rho > 0\}$ has a piecewise-structured dual class.

Corollary 5.21. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\rho} : \rho > 0\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^2)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_{\rho} \mapsto \mathbb{I}_{\{\rho < a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\rho} \mapsto c$.*

Lemma 3.9 and Corollary 5.21 imply the following pseudo-dimension bound.

Corollary 5.22. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho : \rho > 0\}$. The pseudo-dimension of $\mathcal{U}$ is $O(\ln n)$.*

This matches the pseudo-dimension bound by Gupta and Roughgarden [62].

5.4 Revenue Maximization

The design of revenue-maximizing multi-item mechanisms is a notoriously challenging problem. Remarkably, the revenue-maximizing mechanism is not known even when there are just two items for sale. In this setting, the mechanism designer's goal is to field a mechanism with high expected revenue on the distribution over agents' values. A line of research has provided generalization guarantees for mechanism design in the context of revenue maximization [29, 34, 36, 40, 56, 58, 61, 93, 95]. These articles focus on sales settings: there is a seller, not included among the agents, who will use a mechanism to allocate a set of goods among the agents. The agents submit bids describing their values for the goods for sale. The mechanism determines which agents receive which items and how much the agents pay. The seller's revenue is the sum of the agents' payments. The mechanism designer's goal is to select a mechanism that maximizes the revenue. All of the mechanisms they analyze are incentive-compatible, meaning that agents are incentivized to report their values truthfully.

We study the problem of selling m heterogeneous goods to n buyers. We denote a bundle of goods as a subset $b \subseteq [m]$. Each buyer $j \in [n]$ has a valuation function $v_j : 2^{[m]} \to \mathbb{R}$ over bundles of goods. In this setting, the set $\mathcal{X}$ of problem instances consists of n-tuples of buyer values $\boldsymbol{v} = (v_1, \ldots, v_n)$. Selling mechanisms are defined by an *allocation function* and a set of *payment functions*. Every auction in the classes we study is incentive-compatible, so we assume that the bids equal the bidders' valuations. An allocation function $\psi : \mathcal{X} \to (2^{[m]})^n$ maps the values $\boldsymbol{v} \in \mathcal{X}$ to a division of the goods $(b_1, \ldots, b_n) \in (2^{[m]})^n$, where $b_i \subseteq [m]$ is the set of goods buyer i receives. For each agent $i \in [n]$, there is a payment function $p_i : \mathcal{X} \to \mathbb{R}$ which maps values $\boldsymbol{v} \in \mathcal{X}$ to a payment $p_i(\boldsymbol{v}) \in \mathbb{R}_{\geq 0}$ that agent i must make.

We recover generalization guarantees proved by Balcan et al. [20] which apply to a variety of widely studied parameterized mechanism classes, including posted-price mechanisms, multi-part tariffs, second-price auctions with reserves, affine maximizer auctions, virtual valuations combinatorial auctions mixed-bundling auctions, and randomized mechanisms. They provided new bounds for mechanism classes not previously studied in the sample-based mechanism design literature and matched or improved over the best known guarantees for many classes. They proved these guarantees by uncovering structure shared by all of these mechanisms: for any set of buyers' values, revenue is a piecewise-linear function of the mechanism's parameters. This structure is captured by our definition of piecewise decomposability.

Each of these mechanism classes is parameterized by a d-dimensional vector $\boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^d$ for some $d \geq 1$. For example, when $d = m$, $\boldsymbol{\rho}$ might be a vector of prices for each of the items. The revenue of a mechanism is the sum of the agents' payments. Given a mechanism parameterized by a vector $\boldsymbol{\rho} \in \mathbb{R}^d$, we denote the revenue as $u_{\boldsymbol{\rho}} : \mathcal{X} \to \mathbb{R}$, where $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = \sum_{i=1}^n p_i(\boldsymbol{v})$.

Balcan et al. [20] provide pseudo-dimension bounds for any mechanism class that is *delineable*. To define this notion, for any fixed valuation vector $\boldsymbol{v} \in \mathcal{X}$, we use the notation $u_{\boldsymbol{v}}(\boldsymbol{\rho})$ to denote revenue as a function of the mechanism's parameters.

Definition 5.23 ((d, t)-delineable [20]). A mechanism class is (d, t)-*delineable* if:

(1) The class consists of mechanisms parameterized by vectors $\boldsymbol{\rho}$ from a set $\mathcal{P} \subseteq \mathbb{R}^d$; and
(2) For any $\boldsymbol{v} \in \mathcal{X}$, there is a set $\mathcal{H}$ of t hyperplanes such that for any connected component $\mathcal{P}'$ of $\mathcal{P} \setminus \mathcal{H}$, the function $u_{\boldsymbol{v}}(\boldsymbol{\rho})$ is linear over $\mathcal{P}'$.

Delineability naturally translates to decomposability, as we formalize below.

Lemma 5.24. *Let $\mathcal{U}$ be a set of revenue functions corresponding to a (d,t)-delineable mechanism class. The dual class $\mathcal{U}^*$ is $(\mathcal{F},\mathcal{G},t)$-piecewise decomposable, where $\mathcal{G} = \{g_{\boldsymbol{a},\theta} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ consists of halfspace indicator functions $g_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{\rho}\cdot\boldsymbol{a}\le\theta\}}$ and $\mathcal{F} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \mathbb{R} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ consists of linear functions $f_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \boldsymbol{\rho}\cdot\boldsymbol{a} + \theta$.*

Lemmas 3.10 and 5.24 imply the following bound.

Corollary 5.25. *Let $\mathcal{U}$ be a set of revenue functions corresponding to a (d,t)-delineable mechanism class. The pseudo-dimension of $\mathcal{U}$ is at most $O(d\ln(td))$.*

Corollary 5.25 matches the pseudo-dimension bound that Balcan et al. [20] prove. Balcan et al. [20] also prove several lower bounds showing that Corollary 5.25 is tight up to logarithmic factors.

6 Experiments

We complement our theoretical guarantees with experiments in several settings to help illustrate our bounds. Our experiments are in the contexts of both new and previously-studied domains: tuning parameters of sequence alignment algorithms in computational biology and tuning parameters of sales mechanisms to maximize revenue.We summarize two observations from the experiments that help illustrate the theoretical message of this article.

Observation 6.1. *First, using sequence alignment data (Section 6.1), we demonstrate that with a finite number of samples, an algorithm's average performance over the samples provides a tight estimate of its expected performance on unseen instances.*

Observation 6.2. *Second, our experiments empirically illustrate that two algorithm families for the same computational problem may require markedly different training set sizes to avoid overfitting, a fact that has also been explored in prior theoretical research [16, 20]. This shows that it is crucial to bound an algorithm family's intrinsic complexity to provide accurate guarantees, and in this article, we provide tools for doing so. Our experiments here are in the context of mechanism design for revenue maximization (Section 6.2) using real-world data. In this setting, we analyze two natural, practical mechanism families where one of the families is intrinsically simple and the other is intrinsically complex. When we use Theorem 3.3 to select enough samples to ensure that overfitting does not occur for the simple class, we indeed empirically observe overfitting when optimizing over the complex class. The complex class requires more samples to avoid overfitting. Despite surface-level similarities between the complex and simple mechanism families, there is a pronounced difference in their intrinsic complexities, as illustrated by these experiments.*

6.1 Sequence Alignment Experiments

Changing the alignment parameter can alter the accuracy of the produced alignments. Figure 5 shows the regions of the gap-open/gap-extension penalty plane divided into regions such that each region corresponds to a different computed alignment. The regions in the figure are produced using the XPARAL software of Gusfield and Stelling [64], with using the BLOSUM62 amino acid replacement matrix, the scores in each region were computed using Robert Edgar's qscore package.[3] The alignment sequences are a single pairwise alignment from the data set described below.

To illustrate Observation 6.1, we use the IPA tool [77] to learn optimal parameter choices for a given set of example pairwise sequence alignments. We used 861 protein multiple sequence alignment benchmarks that had previously been used in DeBlasio and Kececioglu [35], which

[3] http://drive5.com/qscore/

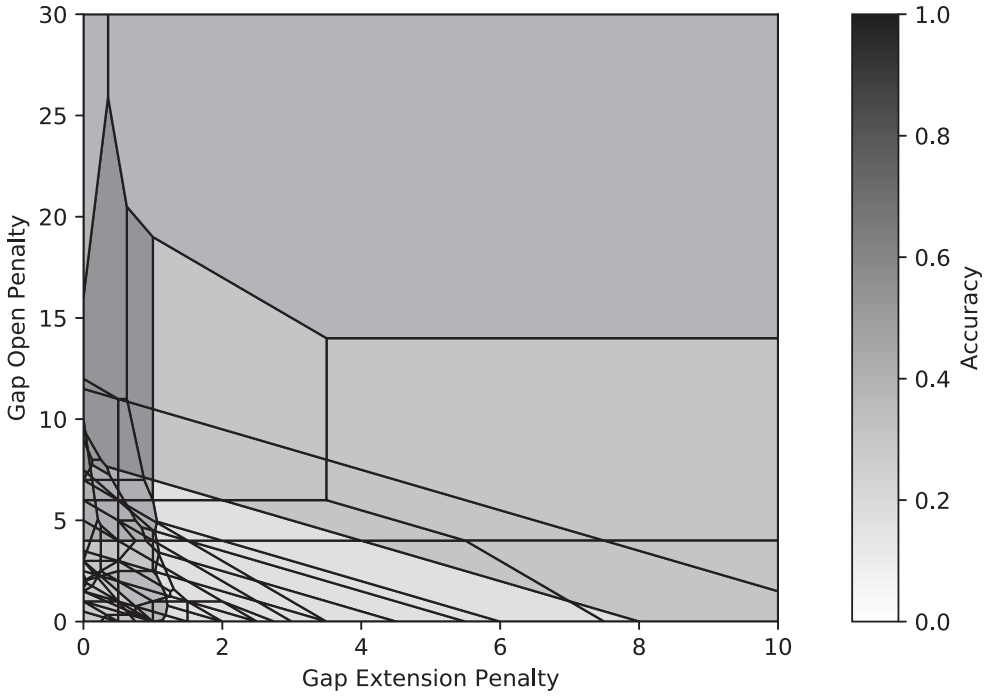


Fig. 5. Parameter space decomposition for a single example.

split these benchmarks into 12 cross-validation folds that evenly distributed the "difficulty" of an alignment (the accuracy of the alignment produced using aligner defaults parameter choice). All pairwise alignments were extracted from each multiple sequence alignment. We then took randomized increasing sized subsets of the pairwise sequence alignments from each training set and found the optimal parameter choices for affine gap costs and alphabet-dependent substitution costs. These parameters were then given to the `Opal` aligner [v3.1b, 128] to realign the pairwise alignments in the associated test sets.

Figure 6 shows the impact of increasing the number of training examples used to optimize parameter choices. As the number of training examples increases, the optimized parameter choice is less able to fit the training data exactly and thus the training accuracy decreases. For the same reason the parameter choices are more general and the test accuracy increases. The test and training accuracies are roughly equal when the training set size is close to $1,000$ examples and remains equal for larger training sets. The test accuracy is actually slightly higher and this is likely due to the training subset not representing the distribution of inputs as well as the full test set due to the randomization being on all of the alignments rather than across difficulty, as was done to create the cross-validation separations.

6.2 Mechanism Design Experiments

In this section, we build off of Section 5.4 by providing experiments that analyze the estimation error of several mechanism classes. We introduced the notion of estimation error in Section 2, but review it here. For a class of mechanisms parameterized by vectors $\boldsymbol{\rho}$, let $u_{\boldsymbol{\rho}}(\boldsymbol{v}) \in [0, H]$ be the revenue of the mechanism parameterized by $\boldsymbol{\rho}$ when the agents' values are defined by the vector $\boldsymbol{v}$. Given a set of N samples $\mathcal{S}$, the mechanism class's estimation error equals the maximum difference, across all parameter vectors $\boldsymbol{\rho}$, between the average revenue of the mechanism over the samples

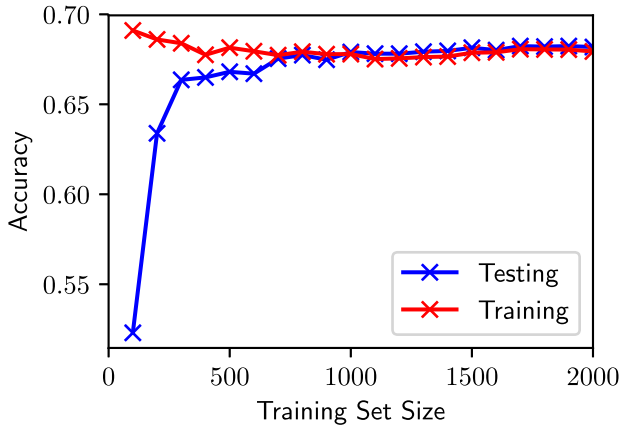


Fig. 6. Pairwise sequence alignment experiments showing the average accuracy on training and test examples using parameter choices optimized for various training set sizes.

$\frac{1}{N}\sum_{\boldsymbol{v}\in\mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v})$ and its expected revenue $\mathbb{E}_{\boldsymbol{v}\sim\mathcal{D}}[u_{\boldsymbol{\rho}}(\boldsymbol{v})]$. We know that with high probability, the estimation error is bounded by $\tilde{O}(H\sqrt{d/N})$, where d is the pseudo-dimension of the mechanism class. In other words, for all parameter vectors $\boldsymbol{\rho}$,

$$\left|\frac{1}{N}\sum_{\boldsymbol{v}\in\mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) - \mathop{\mathbb{E}}_{\boldsymbol{v}\sim\mathcal{D}}\left[u_{\boldsymbol{\rho}}(\boldsymbol{v})\right]\right| = \tilde{O}\left(H\sqrt{\frac{d}{N}}\right).$$

Said another way, $\tilde{O}(\frac{H^2 d}{\epsilon^2})$ samples are sufficient to ensure that with high probability, for every parameter vector $\boldsymbol{\rho}$, the difference between the average and expected utility of the mechanism parameterized by $\boldsymbol{\rho}$ is at most ϵ.

In our experiments, we analyze *second-price auctions with reserves*, one of the most well-studied mechanism classes in economics [121]. In this setting, there is one item for sale and a set of n agents who are interested in buying that item. This mechanism family is defined by a parameter vector $\boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}$, where each entry $\rho[i]$ is the *reserve price* for agent i. The agents report their values for the item to the auctioneer. If the highest bidder—say, agent i^*—reports a bid that is larger than her reserve $\rho[i^*]$, she wins the item and pays the maximum of the second-highest bid and her reserve $\rho[i^*]$. Otherwise, no agent wins the item. The **second-price auction** (**SPA**) is called *anonymous* if every agent has the same reserve price: $\rho[1] = \rho[2] = \cdots = \rho[n]$. Otherwise, the SPA is called *non-anonymous*. Like neutral affine maximizers, SPAs are incentive compatible, so we assume the agents' bids equal their true values. We refer to the class of non-anonymous SPAs as $\mathcal{A}_N$ and the class of anonymous SPAs as $\mathcal{A}_A$.

The class of non-anonymous SPAs $\mathcal{A}_N$ is more complex than the class of anonymous SPAs $\mathcal{A}_A$, and thus the sample complexity of the former is higher than the latter. However, non-anonymous SPAs can provide much higher revenue than anonymous SPAs. We illustrate this tradeoff in our experiments, which helps exemplify Observation 6.2.

In a bit more detail, the pseudo-dimension of the class of non-anonymous SPAs $\mathcal{A}_N$ is $\tilde{O}(n)$, an upper bound proved in prior research [20, 95] which can be recovered using the techniques in this article, as we summarize in Section 5.4. What's more, Balcan et al. [20] proved a pseudo-dimension lower bound of $\Omega(n)$ for this class. Meanwhile, the pseudo-dimension of the class of anonymous SPAs $\mathcal{A}_A$ is $O(1)$. This upper bound was proved in prior research by Morgenstern and Roughgarden [95] and Balcan et al. [20], and it follows from the general techniques presented in

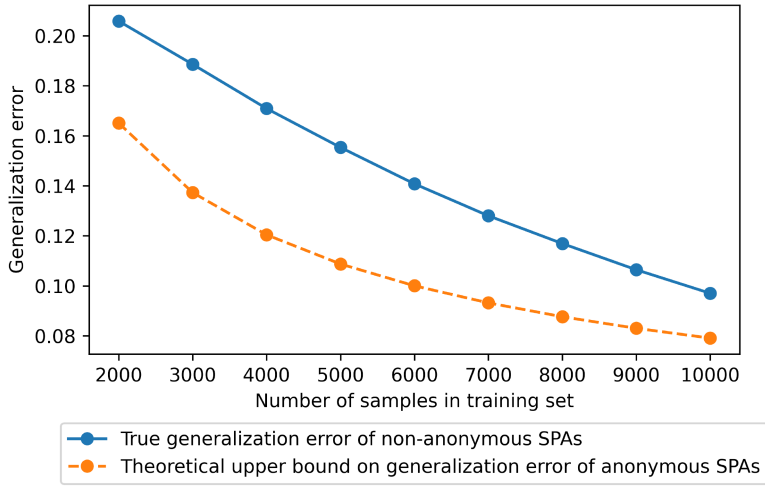


Fig. 7. Revenue maximization experiments. We vary the size of the training set, N, along the x-axis. The orange dashed line is our theoretical upper bound on the estimation error of the class of anonymous SPAs $\mathcal{A}_A$, $\sqrt{\frac{4}{N}\ln(eN)}+\sqrt{\frac{1}{2N}\ln 100}$, which follows from our pseudo-dimension bound (Lemma 6.3). The blue solid line lower bounds the true estimation error of the class of non-anonymous SPAs $\mathcal{A}_N$ over the Jester dataset. For several choices of $N \in [2000, 10000]$, we compute this lower bound by drawing a set of N training instances, finding a mechanism in $\mathcal{A}_N$ with high average revenue over the training set, and calculating the mechanism's estimation error (the difference between its average revenue and expected revenue). For scale, estimation error is a quantity in the range $[0, 1]$.

this article, as we formalize below in Lemma 6.3. In our experiments, we exhibit a distribution over agents' values under which the following properties hold:

(1) The true estimation error of the set of non-anonymous SPAs $\mathcal{A}_N$ is larger than our upper bound on the worst-case estimation error of the set of anonymous SPAs $\mathcal{A}_A$.
(2) The expected revenue of the optimal non-anonymous SPA is significantly larger than the expected revenue of the optimal anonymous SPA: the former is 0.38 and the latter is 0.57.

These two points imply that even though it may be tempting to optimize over the class of non-anonymous SPAs $\mathcal{A}_N$ in order to obtain higher expected revenue, the training set must be large enough to ensure the empirically-optimal mechanism has nearly optimal expected revenue.

The distribution we analyze is over the Jester Collaborative Filtering Dataset [54], which consists of ratings from tens of thousands of users of one hundred jokes—in this example, the jokes could be proxies for comedians, and the agents could be venues who are bidding for the opportunity to host the comedians. Since the auctions we analyze in these experiments are for a single item, we run experiments using agents' values for a single joke, which we select to have a roughly equal number of agents with high values as with low values (we describe this selection in greater detail below). Figure 7 illustrates the outcome of our experiments. The orange dashed line is our upper bound on the estimation error of the class of anonymous SPAs $\mathcal{A}_A$, which equals $\sqrt{\frac{4}{N}\ln(eN)}+\sqrt{\frac{1}{2N}\ln\frac{1}{\delta}}$, with $\delta = 0.01$. This upper bound has been presented in prior research [20, 95], and we recover it using the results presented in this article, as we demonstrate below in Lemma 6.3. The blue solid line is the true estimation error[4] of the class of non-anonymous SPAs $\mathcal{A}_N$ over the Jester dataset, which

[4]Sample complexity and estimation error bounds for SPAs have been studied in prior research [20, 36, 95], and our guarantees match the bounds provided in that literature.

we calculate via the following experiment. For a selection of training set sizes $N \in [2000, 12000]$, we draw N sample agent values, with the number of agents equal to 10,612 (as we explain below). We then find a vector of non-anonymous reserves with maximum average revenue over the samples but low expected revenue on the true distribution, as we describe in greater detail below. We plot this difference between average and expected revenue, averaged over 100 trials of the same procedure. As these experiments illustrate, there is a tradeoff between the sample complexity and revenue guarantees of these two classes, and it is crucial to calculate a class's intrinsic complexity to provide accurate guarantees. We now explain the details of these experiments.

Distribution over agents' values. We use the Jester Collaborative Filtering Dataset [54], which consists of ratings from 24,983 users of 100 jokes. The users' ratings are in the range $[-10, 10]$, so we normalize their values to lie in the interval $[0, 1]$. We select a joke which has at least 5,000 (normalized) ratings in the interval $[\frac{1}{4}, \frac{1}{2}]$ and at least 5,000 (normalized) ratings in the interval $[\frac{3}{4}, 1]$ (specifically, Joke #22). There is a total of 5,334 ratings of the first type and 5,278 ratings of the second type. Let $W = \{w_1, \ldots, w_{10,612}\}$ be all 10,612 values. For every $i \in [10, 612]$, we define the following valuation vector $\boldsymbol{v}^{(i)}$: agent i's value $v_i^{(i)}$ equals w_i and for all other agents $j \neq i$, $v_j^{(i)} = 0$. We define the distribution $\mathcal{D}$ over agents' values to be the uniform distribution over $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(10,612)}$.

Empirically-optimal non-anonymous reserve vectors with poor generalization. Given a set of samples $\mathcal{S} \sim \mathcal{D}^N$, let $p(\boldsymbol{v}^{(1)}), \ldots, p(\boldsymbol{v}^{(10,612)})$ define the empirical distribution over $\mathcal{S}$ (that is, $p(\boldsymbol{v}^{(i)})$ equals the number of times $\boldsymbol{v}^{(i)}$ appears in $\mathcal{S}$ divided by N). Then for any non-anonymous reserve vector $\boldsymbol{\rho} \in \mathbb{R}^n$, the average revenue over the samples is

$$\sum_{i=1}^{10,612} p\left(\boldsymbol{v}^{(i)}\right) \rho[i] \mathbf{1}_{\{w_i \geq \rho[i]\}}. \tag{16}$$

This is because for every vector $\boldsymbol{v}^{(i)}$, the only agent with a non-zero value is agent i, whose value is w_i. Therefore, the highest bidder is agent i, who wins the item if $w_i \geq \rho[i]$, and pays reserve $\rho[i]$, which is the maximum of $\rho[i]$ and the second-highest bid. As is clear from Equation (16), if $\boldsymbol{v}^{(i)} \in \mathcal{S}$, an empirically-optimal reserve vector $\hat{\boldsymbol{\rho}}$ (which maximizes average revenue over the samples) will set $\hat{\rho}_i = w_i$, and if $\boldsymbol{v}^{(i)} \notin \mathcal{S}$, $\hat{\rho}_i$ can be set arbitrarily, because it has no effect on the average revenue over the samples. In our experiments, for all $\boldsymbol{v}^{(i)} \notin \mathcal{S}$, we set $\hat{\rho}_i = 0.7$. The intuition is that those agents i with $\boldsymbol{v}^{(i)} \notin \mathcal{S}$ and $w_i \in [\frac{1}{4}, \frac{1}{2}]$ will not win the item, and therefore will drag down expected revenue.

In Figure 7, the orange dashed line is the difference between the average value of $\hat{\boldsymbol{\rho}}$ over $\mathcal{S}$ and its expected value, which we calculate via the following experiment. For a selection of training set sizes $N \in [2000, 12000]$, we draw the set $\mathcal{S} \sim \mathcal{D}^N$. We then construct the reserve vector $\hat{\boldsymbol{\rho}}$ as described above. We plot the difference between the average value of $\hat{\boldsymbol{\rho}}$ over $\mathcal{S}$ and its expected value, averaged over 100 trials of the same procedure.

Estimation error of anonymous SPAs. We now bound the estimation error of the class of anonymous SPAs $\mathcal{A}_A$ so that we can plot the blue solid line in Figure 7. This pseudo-dimension upper bound has been presented in prior research [20, 95], but here we show that it can be recovered using this article's techniques. We use Lemma 3.8 to obtain a slightly tighter pseudo-dimension bound (up to constant factors) than that of Corollary 5.25.

Given a reserve price $\rho \geq 0$ and valuation vector $\boldsymbol{v} \in \mathbb{R}^n$, let $u_\rho(\boldsymbol{v}) \in [0, 1]$ be the revenue of the anonymous second-price auction with a reserve price of ρ when the bids equal $\boldsymbol{v}$. Using Lemma 3.8, we prove that the pseudo-dimension of this class of revenue functions is at most 2.

Lemma 6.3. *The pseudo-dimension of the class $\mathcal{A}_A = \{u_\rho : \rho \geq 0\}$ is at most 2.*

Proof. Given a vector $\boldsymbol{v}$ of bids, let $v_{(1)}$ be the highest bid in $\boldsymbol{v}$ and let $v_{(2)}$ be the second-highest bid. Under the anonymous SPA, the highest bidder wins the item if $v_{(1)} \geq \rho$ and pays $\max\{v_{(2)}, \rho\}$. Therefore $u_\rho(\boldsymbol{v}) = \max\{v_{(2)}, \rho\}\mathbf{1}_{\{v_{(1)} \geq \rho\}}$. By definition of the dual function,

$$u^*_{\boldsymbol{v}}(u_\rho) = \max\left\{v_{(2)}, \rho\right\} \mathbf{1}_{\left\{v_{(1)} \geq \rho\right\}}.$$

The dual function is thus an increasing function of ρ in the interval $[0, v_{(1)}]$ and is equal to zero in the interval $(v_{(1)}, \infty)$. Therefore, the function has at most two oscillations (as in Definition 3.7). By Lemma 3.8, the pseudo-dimension of $\mathcal{A}_A$ is at most the largest integer D such that $2^D \leq 2D + 1$, which equals 2. Therefore, the theorem statement holds. □

By Equation (2), with probability $1 - \delta$ over the draw of $\mathcal{S} \sim \mathcal{D}^N$, for any reserve $\rho \geq 0$,

$$\left| \frac{1}{N} \sum_{\boldsymbol{v} \in \mathcal{S}} u_\rho(\boldsymbol{v}) - \mathop{\mathbb{E}}_{\boldsymbol{v} \sim \mathcal{D}} \left[u_\rho(\boldsymbol{v})\right] \right| \leq \sqrt{\frac{4}{N} \ln(eN)} + \sqrt{\frac{1}{2N} \ln \frac{1}{\delta}}. \tag{17}$$

In Figure 7, the blue solid line equals the right-hand-side of Equation (17) with $\delta = 0.01$ as a function of N.

Choice of the default reserve. In the above discussion, for all $\boldsymbol{v}^{(i)} \notin \mathcal{S}$, we set a default reserve of $\hat{\rho}_i = 0.7$. As we discussed, for all such i, the choice of $\hat{\rho}_i$ has no effect on the average revenue over the samples. However, it does have an impact on expected revenue. In Figure 8, we compare the expected revenue of the empirically-optimal anonymous SPA and that of the empirically-optimal non-anonymous SPA for different choices of this default reserve. We plot these quantities as a function of the number of samples in the training set. This plot illustrates that a poor choice of the default reserve can cause the expected revenue of the empirically-optimal non-anonymous SPA to be smaller than that of the empirically-optimal anonymous SPA. Intuitively, with a small number of samples, the empirically-optimal non-anonymous SPA will set a large number of reserves to be the default reserve, so a poor choice of the default reserve—in conjunction with a small number of samples—can lead to low expected revenue.

Discussion. In summary, this section exemplifies a distribution over agents' values where:

(1) The true estimation error of the set of non-anonymous SPAs (the orange dashed line in Figure 7) is larger than our upper bound on the worst-case estimation error of the set of anonymous SPAs (the blue solid line in Figure 7).
(2) The expected revenue of the optimal non-anonymous SPA is significantly larger than the expected revenue of the optimal anonymous SPA: the former is 0.38 and the latter is 0.62.

Therefore, there is a tradeoff between the sample complexity and revenue guarantees of these two classes.

7 Conclusions

We provided a general sample complexity theorem for tuning algorithm parameters. Our bound applies whenever a parameterized algorithm's performance is a piecewise-structured function of its parameters: for any fixed problem instance, boundary functions partition the parameters into regions where performance is a well-structured function. We proved this guarantee by exploiting intricate connections between primal function classes (measuring the algorithm's performance as a function of its input) and dual function classes (measuring the algorithm's performance on a fixed input as a function of its parameters). We demonstrated that many parameterized algorithms exhibit this structure and thus our main theorem implies sample complexity guarantees for these algorithms and application domains.

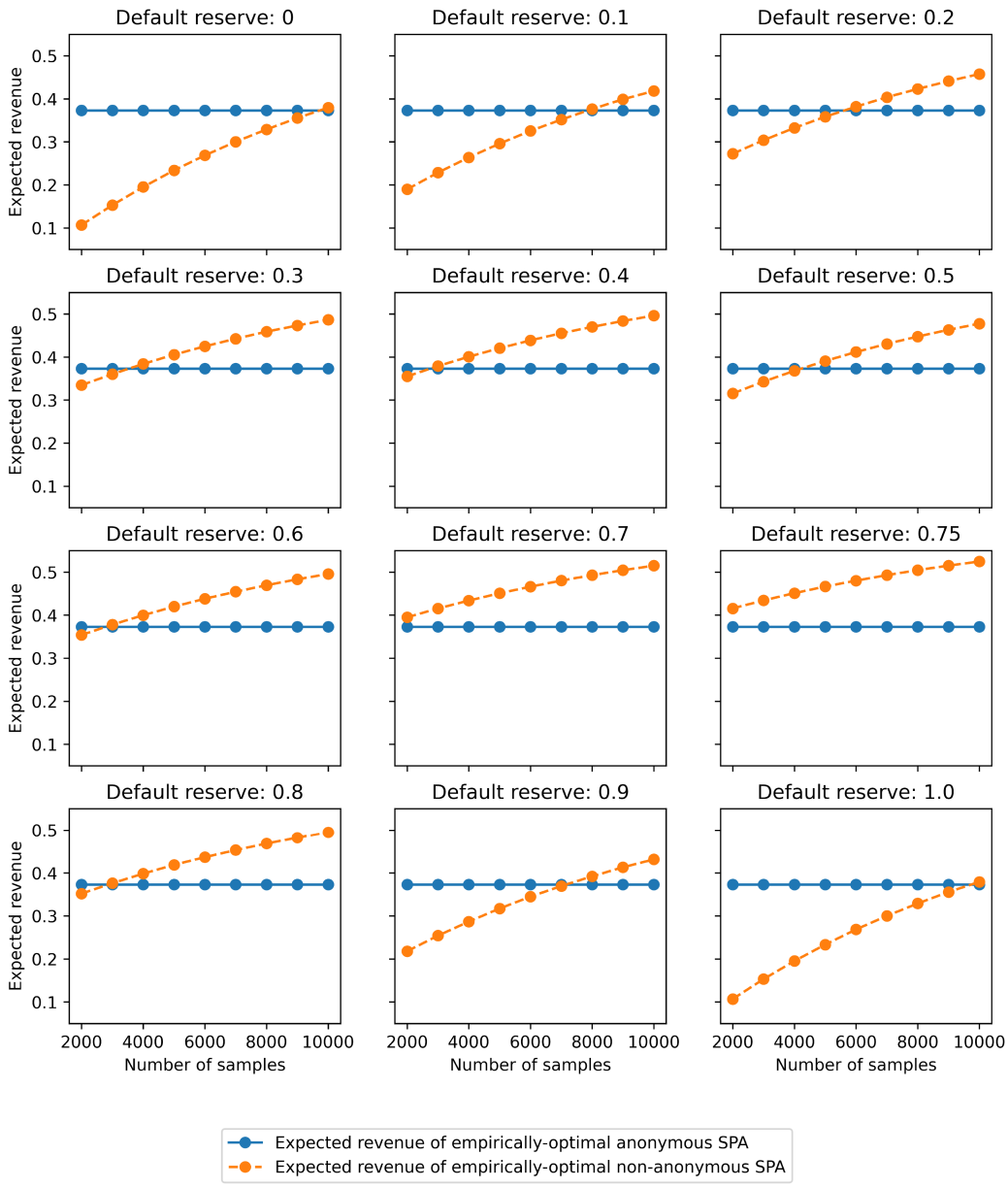


Fig. 8. Comparison of the expected revenue of the empirically-optimal anonymous SPA and that of the empirically-optimal non-anonymous SPA for different choices of the default reserve. We plot these quantities as a function of the number of samples.

A great direction for future research is to build on these ideas for the sake of learning a *portfolio* of configurations, rather than a single high-performing configuration. At runtime, machine learning is used to determine which configuration in the portfolio to employ for the given input. Gupta and Roughgarden [62] and Balcan et al. [22] have provided initial results in this direction, but a general theory is yet to be developed.

Another direction is to provide data-dependent guarantees and problem-specific guarantees that are tailored to the specific algorithm family. For example, as illustrated by the applications we analyze in this article, our guarantees typically grow (at least) linearly with the number of parameters because typically either the set $\mathcal{F}^*$ or $\mathcal{G}^*$ is either linear or even more complex. Related research has proven tighter bounds than those one could achieve using this framework via algorithm-specific analyses [14, 24, 26].

Appendices

A Helpful Lemmas

Lemma A.1 (Shalev-Shwartz and Ben-David [114]). *Let $a \geq 1$ and $b > 0$. If $y < a \ln y + b$, then $y < 4a \ln(2a) + 2b$.*

The following is a corollary of Rolle's theorem that we use in the proof of Lemma 4.8.

Lemma A.2 (Tossavainen [119]). *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.*

Corollary A.3. *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 A.2. □

B Additional Details about Our General Theorem

Lemma 3.10. *Let $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^d\}$ be a class of utility functions defined over a d-dimensional parameter space. Suppose the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, k)$-piecewise decomposable, where the boundary functions $\mathcal{G} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ are halfspace indicator functions $g_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a}\cdot\boldsymbol{\rho} \leq \theta\}}$ and the piece functions $\mathcal{F} = \{f_{\boldsymbol{a},\theta} : \mathcal{U} \to \mathbb{R} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ are linear functions $f_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \boldsymbol{a} \cdot \boldsymbol{\rho} + \theta$. Then $\operatorname{Pdim}(\mathcal{U}) = O(d \ln(dk))$.*

Proof. First, we prove that the VC-dimension of the dual class $\mathcal{G}^*$ is at most $d+1$. The dual class $\mathcal{G}^*$ consists of functions $g^*_{u_{\boldsymbol{\rho}}}$ for all $\boldsymbol{\rho} \in \mathcal{P}$ where $g^*_{u_{\boldsymbol{\rho}}}(g_{\boldsymbol{a},\theta}) = \mathbb{I}_{\{\boldsymbol{a}\cdot\boldsymbol{\rho} \leq \theta\}}$. Let $\hat{\mathcal{G}} = \{\hat{g}_{\boldsymbol{\rho}} : \mathbb{R}^{d+1} \to \{0,1\}\}$ be the class of halfspace thresholds $\hat{g}_{\boldsymbol{\rho}} : (\boldsymbol{a}, \theta) \mapsto \mathbb{I}_{\{\boldsymbol{a}\cdot\boldsymbol{\rho} \leq \theta\}}$. It is well-known that $\operatorname{VCdim}(\hat{\mathcal{G}}) \leq d+1$, which we prove means that $\operatorname{VCdim}(\mathcal{G}^*) \leq d+1$. For a contradiction, suppose $\mathcal{G}^*$ can shatter $d+2$ functions $g_{\boldsymbol{a}_1,\theta_1}, \ldots, g_{\boldsymbol{a}_{d+2},\theta_{d+2}} \in \mathcal{G}$. Then for every subset $T \subseteq [d+2]$, there exists a parameter vector $\boldsymbol{\rho}_T$ such that $\boldsymbol{a}_i \cdot \boldsymbol{\rho}_T \leq \theta_i$ if and only if $i \in T$. This means that $\hat{\mathcal{G}}$ can shatter the tuples $(\boldsymbol{a}_1, \theta_1), \ldots, (\boldsymbol{a}_{d+2}, \theta_{d+2})$ as well, which contradicts the fact that $\operatorname{VCdim}(\hat{\mathcal{G}}) \leq d+1$. Therefore, $\operatorname{VCdim}(\mathcal{G}^*) \leq d+1$.

By a similar argument, we prove that the pseudo-dimension of the dual class $\mathcal{F}^*$ is at most $d+1$. The dual class $\mathcal{F}^*$ consists of functions $f^*_{u_{\boldsymbol{\rho}}}$ for all $\boldsymbol{\rho} \in \mathcal{P}$ where $f^*_{u_{\boldsymbol{\rho}}}(f_{\boldsymbol{a},\theta}) = \boldsymbol{a} \cdot \boldsymbol{\rho} + \theta$. Let $\hat{\mathcal{F}} = \{\hat{f}_{\boldsymbol{\rho}} : \mathbb{R}^{d+1} \to \mathbb{R}\}$ be the class of linear functions $\hat{f}_{\boldsymbol{\rho}} : (\boldsymbol{a}, \theta) \mapsto \boldsymbol{a} \cdot \boldsymbol{\rho} + \theta$. It is well-known that $\operatorname{Pdim}(\hat{\mathcal{F}}) \leq d+1$, which we prove means that $\operatorname{Pdim}(\mathcal{F}^*) \leq d+1$. For a contradiction, suppose $\mathcal{F}^*$

can shatter $d+2$ functions $f_{\boldsymbol{a}_1,\theta_1},\ldots,f_{\boldsymbol{a}_{d+2},\theta_{d+2}} \in \mathcal{F}$. Then there exist witnesses $z_1,\ldots,z_{d+2}$ such that for every subset $T \subseteq [d+2]$, there exists a parameter vector $\boldsymbol{\rho}_T$ such that $\boldsymbol{a}_i \cdot \boldsymbol{\rho}_T + \theta_i \le z_i$ if and only if $i \in T$. This means that $\hat{\mathcal{F}}$ can shatter the tuples $(\boldsymbol{a}_1,\theta_1),\ldots,(\boldsymbol{a}_{d+2},\theta_{d+2})$ as well, which contradicts the fact that $\text{Pdim}(\hat{\mathcal{F}}) \le d+1$. Therefore, $\text{Pdim}(\mathcal{F}^*) \le d+1$.

The lemma statement now follows from Theorem 3.3. □

C Additional Details about Sequence Alignment (Section 4.1)

Lemma C.1. *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. For any alignment (τ_1,τ_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 τ_1 and τ_2 have the same number of gaps. To prove the upper bound, we count the number of alignments (τ_1,τ_2) where τ_1 and τ_2 each have exactly i gaps. There are $\binom{n+i}{i}$ choices for the sequence τ_1. Given a sequence τ_1, we can only pair a gap in τ_2 with a non-gap in τ_1. Since there are i gaps in τ_2 and n non-gaps in τ_1, there are $\binom{n}{i}$ choices for the sequence τ_2 once τ_1 is fixed. This means that there are $\binom{n+i}{i}\binom{n}{i} \le 2^n n^{2n}$ alignments (τ_1,τ_2) where τ_1 and τ_2 each have exactly i gaps. Summing over $i \in [n]$, the total number of alignments is at most $2^n n^{2n+1}$. □

Theorem 4.5. *Under the affine gap model, there exists a set $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^3_{\ge 0}\}$ of co-optimal-constant algorithms and an alphabet Σ such that the set of functions*

$$\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : (S_1,S_2) \mapsto u\left(S_1,S_2,A_{\boldsymbol{\rho}}(S_1,S_2)\right) \mid \boldsymbol{\rho} \in \mathbb{R}^3_{\ge 0}\right\},$$

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 $\Omega(\log n)$.

Proof. To prove this theorem, we identify:

(1) An alphabet Σ,
(2) A set of $N = \Theta(\log n)$ sequence pairs $(S_1^{(1)},S_2^{(1)}),\ldots,(S_1^{(N)},S_2^{(N)}) \in \cup_{i=1}^n \Sigma^i \times \Sigma^i$,
(3) A ground-truth alignment $L_*^{(i)}$ for each sequence pair $(S_1^{(i)},S_2^{(i)})$, 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 $u_{0,\rho[T],0}(S_1^{(i)},S_2^{(i)}) < z_i$ and if $i \notin [T]$, then $u_{0,\rho[T],0}(S_1^{(i)},S_2^{(i)}) \ge z_i$.

We now describe each of these four elements in turn.

The alphabet Σ**.** Let $k = 2^{\lfloor \log \sqrt{n/2} \rfloor} - 1 = \Theta(\sqrt{n})$. The alphabet Σ consists of $4k$ characters[5] we denote as $\{\mathsf{a}_i,\mathsf{b}_i,\mathsf{c}_i,\mathsf{d}_i\}_{i=1}^k$.

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

— The subsequence $t_1^{(i)}$ begins with i a_is followed by $\mathsf{b}_i\mathsf{d}_i$. For example, $t_1^{(3)} = \mathsf{a}_3\mathsf{a}_3\mathsf{a}_3\mathsf{b}_3\mathsf{d}_3$.
— The subsequence $t_2^{(i)}$ begins with 1 b_i, followed by i c_is, followed by 1 d_i. For example, $t_2^{(3)} = \mathsf{b}_3\mathsf{c}_3\mathsf{c}_3\mathsf{c}_3\mathsf{d}_3$.

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

[5]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.

We use these subsequence pairs to define a set of $N = \log(k+1) = \Theta(\log n)$ sequence pairs. The first sequence pair, $(S_1^{(1)}, S_2^{(1)})$ 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 $(2^{i-1})^{th}$ 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 $S_1^{(k+1-2^{i-1})}$ and $S_2^{(k+1-2^{i-1})}$ 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, $(S_1^{(1)}, S_2^{(1)})$. 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.

Example C.2. 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:[6]

$S_1^{(1)} = \mathtt{a_1 b_1 d_1 a_2 a_2 b_2 d_2 a_3 a_3 a_3 b_3 d_3 a_4 a_4 a_4 a_4 b_4 d_4 a_5 a_5 a_5 a_5 a_5 b_5 d_5 a_6 a_6 a_6 a_6 a_6 a_6 b_6 d_6 a_7 a_7 a_7 a_7 a_7 a_7 a_7 b_7 d_7}$

$S_2^{(1)} = \mathtt{b_1 c_1 d_1 b_2 c_2 c_2 d_2 b_3 c_3 c_3 c_3 d_3 b_4 c_4 c_4 c_4 c_4 d_4 b_5 c_5 c_5 c_5 c_5 c_5 d_5 b_6 c_6 c_6 c_6 c_6 c_6 c_6 d_6 b_7 c_7 c_7 c_7 c_7 c_7 c_7 c_7 d_7}$

$S_1^{(2)} = \mathtt{a_2 a_2 b_2 d_2 a_4 a_4 a_4 a_4 b_4 d_4 a_6 a_6 a_6 a_6 a_6 a_6 b_6 d_6}$

$S_2^{(2)} = \mathtt{b_2 c_2 c_2 d_2 b_4 c_4 c_4 c_4 c_4 d_4 b_6 c_6 c_6 c_6 c_6 c_6 c_6 d_6}$

$S_1^{(3)} = \mathtt{a_4 a_4 a_4 a_4 b_4 d_4}$

$S_2^{(3)} = \mathtt{b_4 c_4 c_4 c_4 c_4 d_4}$

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 $(t_1^{(i)}, t_2^{(i)})$. The resulting ground-truth alignments will be a concatenation of these alignments. The first alignment, which we denote as $(h_1^{(i)}, h_2^{(i)})$, is defined as follows: $h_1^{(i)}$ begins with i $\mathtt{a}_i$s, followed by 1 $\mathtt{b}_i$, followed by i

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

gap characters, followed by 1 d_i; $h_2^{(i)}$ begins with i gap characters, followed by 1 b_i, followed by i c_is, followed by 1 d_i. For example,

$$\begin{array}{llllllll} h_1^{(3)} = \mathsf{a}_3 & \mathsf{a}_3 & \mathsf{a}_3 & \mathsf{b}_3 & - & - & - & \mathsf{d}_3 \\ h_2^{(3)} = - & - & - & \mathsf{b}_3 & \mathsf{c}_3 & \mathsf{c}_3 & \mathsf{c}_3 & \mathsf{d}_3 \end{array}.$$

The second alignment, which we denote as $(\ell_1^{(i)}, \ell_2^{(i)})$, is defined as follows: $\ell_1^{(i)}$ begins with i a_is, followed by 1 b_i, followed by i gap characters, followed by 1 d_i; $\ell_2^{(i)}$ begins with 1 b_i, followed by i gap characters, followed by i c_is, followed by 1 d_i. For example,

$$\begin{array}{llllllll} \ell_1^{(3)} = \mathsf{a}_3 & \mathsf{a}_3 & \mathsf{a}_3 & \mathsf{b}_3 & - & - & - & \mathsf{d}_3 \\ \ell_2^{(3)} = \mathsf{b}_3 & - & - & - & \mathsf{c}_3 & \mathsf{c}_3 & \mathsf{c}_3 & \mathsf{d}_3 \end{array}.$$

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

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

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 $z_1 = z_2 = \cdots = z_N = \frac{3}{4}$.

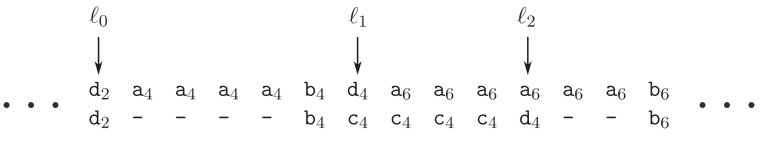
Shattering the N sequence pairs. Our goal is to show that for any subset $T \subseteq [N]$, there exists an indel penalty parameter $\rho[T]$ such that if $i \in [T]$, then $u_{0,\rho[T],0}(S_1^{(i)}, S_2^{(i)}) < \frac{3}{4}$ and if $i \notin [T]$, then $u_{0,\rho[T],0}(S_1^{(i)}, S_2^{(i)}) \geq \frac{3}{4}$.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

Claim C.3. *For any pair $(S_1^{(i)}, S_2^{(i)})$ and indel parameter $\rho[2] \geq 0$, there exists an alignment*

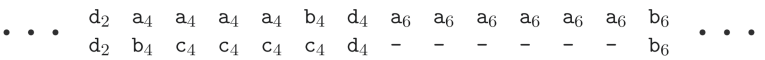
$$L \in \operatorname{argmax}_{L'} MT\left(S_1^{(i)}, S_2^{(i)}, L'\right) - \rho[2] \cdot ID\left(S_1^{(i)}, S_2^{(i)}, L'\right)$$

such that each d_j character in $S_1^{(i)}$ is matched to d_j in $S_2^{(i)}$.

Proof. Let $L_0 \in \operatorname{argmax}_{L'} \text{MT}(S_1^{(i)}, S_2^{(i)}, L') - \rho[2] \cdot \text{ID}(S_1^{(i)}, S_2^{(i)}, L')$ be an alignment such that some d_j character in $S_1^{(i)}$ is not matched to d_j in $S_2^{(i)}$. Denote the alignment L_0 as (τ_1, τ_2). Let $j \in \mathbb{Z}$ be the smallest integer such that for some indices $\ell_1 \neq \ell_2$, $\tau_1[\ell_1] = \mathsf{d}_j$ and $\tau_2[\ell_2] = \mathsf{d}_j$. Next, let ℓ_0 be the maximum index smaller than ℓ_1 and ℓ_2 such that $\tau_1[\ell_0] = \mathsf{d}_{j'}$ for some $j' \neq j$. We illustrate ℓ_0, ℓ_1,



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



(b) An alignment where the gap characters in Figure 9a are shifted such that the $\mathtt{d}_j$ characters are aligned. The objective function value of both alignments is the same.

Fig. 9. Illustration of Claim C.3: we can assume that each $\mathtt{d}_j$ character in $S_1^{(i)}$ is matched to $\mathtt{d}_j$ in $S_2^{(i)}$.

and ℓ_2 in Figure 9(a). By definition of j, we know that $\tau_1[\ell_0] = \tau_2[\ell_0] = \mathtt{d}_{j'}$. We also know there is at least one gap character in $\{\tau_1[i] : \ell_0 + 1 \le i \le \ell_1\} \cup \{\tau_2[i] : \ell_0 + 1 \le i \le \ell_2\}$ because otherwise, the $\mathtt{d}_j$ characters would be aligned in L_0. Moreover, we know there is at most one match among these elements between characters other than $\mathtt{d}_j$ (namely, between the character $\mathtt{b}_j$). If we rearrange all of these gap characters so that they fall directly after $\mathtt{d}_j$ in both sequences, as in Figure 9(b), then we may lose the match between the character $\mathtt{b}_j$, but we will gain the match between the character $\mathtt{d}_j$. 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 $(S_1^{(i)}, S_2^{(i)})$ and indel parameter $\rho[2] \ge 0$, under the alignment $L = A_{0,\rho[2],0}(S_1^{(i)}, S_2^{(i)})$ returned by the algorithm $A_{0,\rho[2],0}$, all $\mathtt{d}_j$ characters in $S_1^{(i)}$ are matched to $\mathtt{d}_j$ in $S_2^{(i)}$.

Claim C.4. *Suppose that the character $\mathtt{b}_j$ is in $S_1^{(i)}$ and $S_2^{(i)}$. The $\mathtt{b}_j$ characters will be matched in $L = A_{0,\rho[2],0}(S_1^{(i)}, S_2^{(i)})$ if and only if $\rho[2] \le \frac{1}{2j}$.*

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

We now use Claims C.3 and C.4 to prove that we can shatter the N sequence pairs $(S_1^{(1)}, S_2^{(1)}), \ldots, (S_1^{(N)}, S_2^{(N)})$.

Claim C.5. *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, $u_{0,\rho[2],0}(S_1^{(i)}, S_2^{(i)})$ switches from above $\frac{3}{4}$ to below $\frac{3}{4}$, or vice versa, beginning with $u_{0,0,0}(S_1^{(i)}, S_2^{(i)}) < \frac{3}{4}$ and ending with $u_{0,1,0}(S_1^{(i)}, S_2^{(i)}) > \frac{3}{4}$.*

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 b_j characters are $\mathsf{b}_{2^{i-1}}, \mathsf{b}_{2\cdot 2^{i-1}}, \mathsf{b}_{3\cdot 2^{i-1}}, \ldots, \mathsf{b}_{k-2^{i-1}+1}$. 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 d_j characters will be aligned, as will all b_j characters. This means we will correctly align all d_j characters and we will correctly align all b_j characters in the $(h_1^{(j)}, h_2^{(j)})$ pairs, but we will incorrectly align the b_j characters in the $(\ell_1^{(j)}, \ell_2^{(j)})$ pairs. The number of $(h_1^{(j)}, h_2^{(j)})$ pairs in this reference alignment is $\frac{k+1}{2^i} - 1$ and the number of $(\ell_1^{(j)}, \ell_2^{(j)})$ 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 $2^{i+1} \le 2(k+1) < 3(k+1)$.

Next, suppose we increase $\rho[2]$ to lie in the interval $(\frac{1}{2(k+1)-2^i}, \frac{1}{2(k+1)-2\cdot 2^i}]$. Since it is no longer the case that $\rho[2] \le \frac{1}{2(k-2^{i-1}+1)}$, we know that the $\mathsf{b}_{k-2^{i-1}+1}$ 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 d_j characters and we will correctly align all b_j characters in the $(h_1^{(j)}, h_2^{(j)})$ pairs, but we will incorrectly align all but one of the b_j characters in the $(\ell_1^{(j)}, \ell_2^{(j)})$ 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 $2^{i+1} \le 2(k+1) < 4(k+1)$.

Next, suppose we increase $\rho[2]$ to lie in the interval $(\frac{1}{2(k+1)-2\cdot 2^i}, \frac{1}{2(k+1)-3\cdot 2^i}]$. Since it is no longer the case that $\rho[2] \le \frac{1}{2(k-2\cdot 2^{i-1}+1)}$, we know that the $\mathsf{b}_{k-2\cdot 2^{i-1}+1}$ 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 d_j characters and we will correctly align the b_j characters in all but one of the $(h_1^{(j)}, h_2^{(j)})$ pairs, but we will incorrectly align all but one of the b_j characters in the $(\ell_1^{(j)}, \ell_2^{(j)})$ 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 $(\frac{1}{2(k+1)-j2^i}, \frac{1}{2(k+1)-(j+1)2^i})$ where $u_{0,\rho[2],0}(S_1^{(i)}, S_2^{(i)})$ 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, $u_{0,\rho[2],0}(S_1^{(i-1)}, S_2^{(i-1)}) < \frac{3}{4}$ and in the second, $u_{0,\rho[2],0}(S_1^{(i-1)}, S_2^{(i-1)}) > \frac{3}{4}$. Therefore, for any subset $T \subseteq [N]$, there exists an indel penalty parameter $\rho[T]$ such that if $i \in [T]$, then $u_{0,\rho[T],0}(S_1^{(i)}, S_2^{(i)}) < \frac{3}{4}$ and if $i \notin [T]$, then $u_{0,\rho[T],0}(S_1^{(i)}, S_2^{(i)}) > \frac{3}{4}$. □

C.1 Tighter Guarantees for a Structured Algorithm Subclass: Sequence Alignment using Hidden Markov Models

While we focused on the affine gap model in the previous section, which was inspired by the results in Gusfield et al. [63], the result in Pachter and Sturmfels [101] helps to provide uniform convergence guarantees for any alignment scoring function that can be modeled as a *hidden Markov model (HMM)*. A bound on the number of parameter choices that emit distinct sets of co-optimal alignments in that work is found by taking an algebraic view of the alignment HMM with d tunable parameters. In fact, the bounds provided can be used to provide guarantees for many types of HMMs.

Lemma C.6. *Let $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$ be a set of co-optimal-constant algorithms and let u be a utility function mapping tuples (S_1, S_2, L) of sequence pairs and alignments to the interval $[0, 1]$. Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u(S_1, S_2, A_{\boldsymbol{\rho}}(S_1, S_2)) \mid \boldsymbol{\rho} \in \mathbb{R}\}$ mapping sequence pairs $S_1, S_2 \in \Sigma^n$ to $[0, 1]$. For some constant $c_1 > 0$, the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, c_1^2 n^{2d(d-1)/(d+1)})$-piecewise decomposable, where $\mathcal{G} = \{g_{\boldsymbol{a}} : \mathcal{U} \to \{0, 1\} \mid \boldsymbol{a} \in \mathbb{R}^{d+1}\}$ consists of halfspace indicator functions $g_{\boldsymbol{a}} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{a_1\rho[1]+\cdots+a_d\rho[d]<a_{d+1}\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$.*

Proof. Fix a sequence pair S_1 and S_2 and consider the function $u^*_{S_1,S_2} : \mathcal{U} \to \mathbb{R}$ from the dual class $\mathcal{U}^*$, where $u^*_{S_1,S_2}(u_{\boldsymbol{\rho}}) = u_{\boldsymbol{\rho}}(S_1, S_2)$. Consider the set of alignments $\mathcal{L}_{S_1,S_2} = \{A_{\boldsymbol{\rho}}(S_1, S_2) \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$. There are at most $O(n^{d(d-1)/(d+1)})$ sets of co-optimal solutions as we range $\boldsymbol{\rho}$ over $\mathbb{R}^d$ [101]. The remainder of the proof is analogous to that for Lemma 4.3. □

Finally the results of Lemma C.6 imply the following pseudo-dimension bound.

Corollary C.7. *Let $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$ be a set of co-optimal-constant algorithms and let u be a utility function mapping tuples (S_1, S_2, L) to $[0, H]$. Let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{u_{\boldsymbol{\rho}} : (S_1, S_2) \mapsto u\left(S_1, S_2, A_{\boldsymbol{\rho}}\left(S_1, S_2\right)\right) \mid \boldsymbol{\rho} \in \mathbb{R}^d\right\}$$

mapping sequence pairs $S_1, S_2 \in \Sigma^n$ to $[0, 1]$. Then $\operatorname{Pdim}(\mathcal{U}) = O(d^2 \ln n)$.

C.2 Progressive Multiple Sequence Alignment

The multiple sequence alignment problem is a generalization of the pairwise alignment problem introduced in Section 4.1. Let Σ be an abstract alphabet and let $S_1, \ldots, S_\kappa \in \Sigma^n$ be a collection of sequences in Σ of length n. A *multiple sequence alignment* is a collection of sequences $\tau_1, \ldots, \tau_\kappa \in (\Sigma \cup \{-\})^*$ such that the following hold:

(1) The aligned sequences are the same length: $|\tau_1| = |\tau_2| = \cdots = |\tau_\kappa|$.
(2) Removing the gap characters from τ_i gives S_i: for all $i \in [\kappa]$, $\texttt{del}(\tau_i) = S_i$.
(3) For every position in the alignment, at least one of the aligned sequences has a non-gap character. In other words, for every position $i \in [|\tau_1|]$, there exists a sequence τ_j such that $\tau_j[i] \neq -$.

The extension from pairwise to multiple sequence alignment is computationally challenging: all common formulations of the problem are NP-complete [75, 122]. As a result, heuristics have been developed to find good but possibly sub-optimal alignments. The most common heuristic

approach is called *progressive multiple sequence alignment*. It leverages efficient pairwise alignment algorithms to heuristically align multiple sequences [45].

The input to a progressive multiple sequence alignment algorithm is a collection of sequences $S_1, \ldots, S_\kappa$ together with a binary *guide tree* G with κ leaves.[7] The tree indicates how the original alignment should be decomposed into a hierarchy of subproblems, each of which can be heuristically solved using pairwise alignment. The leaves of the guide tree correspond to the input sequences $S_1, \ldots, S_\kappa$.

While there are many formalizations of the progressive alignment method, for the sake of analysis we will focus on "partial consensus" described by Higgins and Sharp [66]. Here, we provide a high-level description of the algorithm. At a high level, the algorithm recursively constructs an alignment in two stages: first, it creates a *consensus sequence* for each node in the guide tree using a pairwise alignment algorithm, and then it propagates the node-level alignments to the leaves by inserting additional gap characters.

In a bit more detail, for each node v in the tree, we construct an alignment L'_v of the consensus sequences of its children as well as a consensus sequence $\sigma'_v \in \Sigma^*$. Since each leaf corresponds to a single input sequence, it has a trivial alignment and the consensus sequence is just the input sequence itself. For an internal node v with children c_1 and c_2, we use a pairwise alignment algorithm to construct an alignment of the consensus strings σ'_{v_1} and σ'_{v_2}. Finally, we define the consensus sequence of the node v to be the string $\sigma_v \in \Sigma^*$ such that $\sigma_v[i]$ is the most-frequent non-gap character in the i^{th} position in the alignment L'_v. By defining the consensus sequence in this way, we can represent all of the sub-alignments of the leaves of the subtree rooted at v as a single sequence which can be aligned using existing methods. We obtain a full multiple sequence alignment by iteratively replacing each consensus sequence by the pairwise alignment it represents, adding gap columns to the sub-alignments when necessary. Once we add a gap to a sequence, we never remove it: "once a gap, always a gap."

See Algorithm 2 for more details, which relies on the following definition.

Definition C.8. Let (τ_1, τ_2) be a sequence alignment. The *consensus sequence* of this alignment is the sequence $\sigma \in \Sigma^*$ where $\sigma[j]$ is the most-frequent non-gap character in the j^{th} position in the alignment (breaking ties in a fixed but arbitrary way). For example, the consensus sequence of the alignment

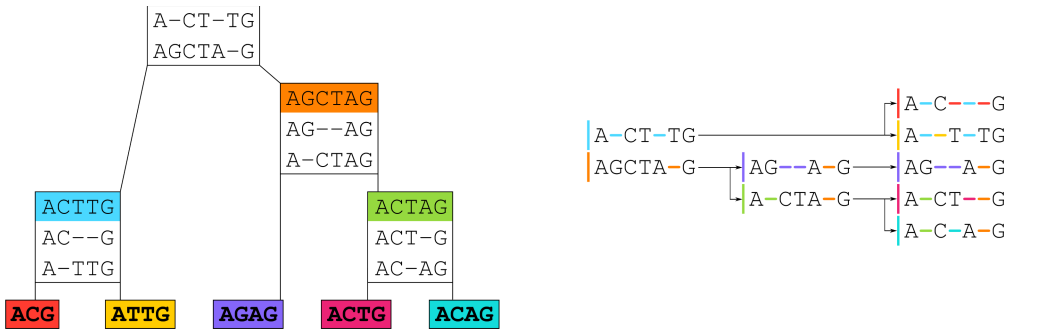
$$\begin{bmatrix} \texttt{A} & \texttt{T} & \texttt{-} & \texttt{C} \\ \texttt{G} & \texttt{-} & \texttt{C} & \texttt{C} \end{bmatrix}$$

is ATCC when ties are broken in favor of A over G.

Figure 10 illustrates an example of this algorithm in action, and corresponds to the pseudo-code given in Algorithm 2. The first loop matches with Figure 10(a), the second and third match with Figure 10(b).

The family $\{A_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$ of parameterized pairwise alignment algorithms introduced in Section 4.1 induces a parameterized family of progressive multiple sequence alignment algorithms $\{M_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^d\}$. In particular, the algorithm $M_{\boldsymbol{\rho}}$ takes as input a collection of input sequences $S_1, \ldots, S_\kappa \in \Sigma^n$ and a guide tree G, and it outputs a multiple-sequence alignment L by applying the pairwise alignment algorithm $A_{\boldsymbol{\rho}}$ at each node of the guide tree. We assume that there is a utility function that characterizes an alignment's quality, denoted $u(S_1, \ldots, S_\kappa, L) \in [0, 1]$. We then define $u_{\boldsymbol{\rho}}(S_1, \ldots, S_\kappa, G) = u(S_1, \ldots, S_\kappa, M_{\boldsymbol{\rho}}(S_1, \ldots, S_\kappa, G))$ to be the utility of the alignment

[7] The problem of constructing the guide tree is also an algorithmic task, often tackled via hierarchical clustering, but we are agnostic to that pre-processing step.



(a) A guide tree (b) Constructing the alignment using the guide tree

Fig. 10. This figure illustrates an example of the progressive sequence alignment algorithm in action. Figure 10(a) depicts a completed guide tree. The five input sequences are represented by the leaves. Each internal leaf, depicts an alignment of the (consensus) sequences contained in the leaf's children. Each internal leaf other than the root also contains the consensus sequence corresponding to that alignment. Figure 10(b) illustrates how to extract an alignment of the five input strings (as well as the consensus strings) from Figure 10(a).

ALGORITHM 2: Progressive Alignment Algorithm ProgressiveAlignment

Input: Binary guide tree G, pairwise sequence alignment algorithm A_{ρ}

Let $v_1, \ldots, v_m$ be an ordering of the nodes in G from deepest to shallowest, with nodes of the same depth ordered arbitrarily

for $i \in \{1, \ldots, m\}$ **do** ▷ Compute the consensus sequences

  if v_i is a leaf **then**

    Set σ'_{v_i} to be the leaf's sequence

  else

    Let c_1 and c_2 be the children of v_i

    Compute the pairwise alignment $L'_{v_i} = A_{\rho}(\sigma'_{c_1}, \sigma'_{c_2})$

    Set σ'_{v_i} to be the consensus sequence of L'_{v_i} (as in Definition C.8)

set $\sigma_{v_m} = \sigma'_{v_m}$ ▷ note v_m is the root of G

for $i \in \{m, \ldots, 1\}$ **do** ▷ Compute the alignment sequences

  if v_i is not a leaf **then**

    Let $L'_{v_i} = (\tau'_1, \tau'_2)$ be the alignment sequences computed at v_i

    Let c_1 and c_2 be the children of v_i

    Set $\sigma_{c_1} = \sigma_{c_2} =$ ""

    Set $k = 0$

    for $j \in [|\sigma_{v_i}|]$ **do**

      if $\sigma_{v_i}[j] =$ '-' **then**

        Append '-' to the end of both σ_{c_1} and σ_{c_2}

      else

        Append $\tau'_1[k]$ to the end of σ_{c_1}

        Append $\tau'_2[k]$ to the end of σ_{c_2}

        Increment k by 1

for $i \in \{1, \ldots, m\}$ **do** ▷ Construct the final alignment

  if v_i is a leaf representing sequence S_j **then**

    Set $\tau_j = \sigma_{v_i}$

returned by the algorithm $M_{\boldsymbol{\rho}}$. The proof of the following lemma follows by the same logic as Lemma 4.1 for pairwise sequence alignment, inductively over the guide tree.

Lemma C.9. *Let G be a guide tree of depth η and let $\mathcal{U}$ be the set of functions*

$$\mathcal{U} = \left\{ u_{\boldsymbol{\rho}} : (S_1, \ldots, S_\kappa, G) \mapsto u\big(S_1, \ldots, S_\kappa, M_{\boldsymbol{\rho}}(S_1, \ldots, S_\kappa, G)\big) \mid \boldsymbol{\rho} \in \mathbb{R}^d \right\}.$$

The dual class $\mathcal{U}^*$ is

$$\left(\mathcal{F}, \mathcal{G}, \left(4^{n\kappa}\,(n\kappa)^{4n\kappa+2}\right)^{2d^\eta}\, 4^{d^{\eta+1}}\right)\text{-piecewise decomposable,}$$

where $\mathcal{G} = \{g_{\boldsymbol{a},\theta} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ consists of halfspace indicator functions $g_{\boldsymbol{a},\theta} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a}\cdot\boldsymbol{\rho} \le \theta\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$.

Proof. A key step in the proof of Lemma 4.1 for pairwise alignment shows that for any pair of sequences $S_1, S_2 \in \Sigma^n$, we can find a set $\mathcal{H}$ of $4^n n^{4n+2}$ hyperplanes such that for any pair $\boldsymbol{\rho}$ and $\boldsymbol{\rho}'$ belonging to the same connected component of $\mathbb{R}^d \setminus \mathcal{H}$, we have $A_{\boldsymbol{\rho}}(S_1, S_2) = A_{\boldsymbol{\rho}'}(S_1, S_2)$. We use this result to prove the following claim.

Claim C.10. *For each node v in the guide tree, there is a set $\mathcal{H}_v$ of hyperplanes where for any connected component R of $\mathbb{R}^d \setminus \mathcal{H}_v$, the alignment and consensus sequence computed by $M_{\boldsymbol{\rho}}$ is fixed across all $\boldsymbol{\rho} \in R$. Moreover, the size of $\mathcal{H}_v$ is bounded as follows:*

$$|\mathcal{H}_v| \le \ell^{d^{\text{height}(v)}} \left(\ell 4^d\right)^{\left(d^{\text{height}(v)}-1\right)/(d-1)},$$

where $\ell := 4^{n\kappa}(n\kappa)^{4n\kappa+2}$.

Before we prove Claim C.10, we remark that the longest consensus sequence computed for any node v of the guide tree has length at most $n\kappa$, which is a bound on the sum of the lengths of the input sequences.

Proof of Claim C.10. We prove this claim by induction on the guide tree G. The base case corresponds to the leaves of G. On each leaf, the alignment and consensus sequence constructed by $M_{\boldsymbol{\rho}}$ is constant for all $\boldsymbol{\rho} \in \mathbb{R}^d$, since there is only one string to align (i.e., the input string placed at that leaf). Therefore, the claim holds for the leaves of G. Moving to an internal node v, suppose that the inductive hypothesis holds for its children v_1 and v_2. Assume without loss of generality that $\text{height}(v_1) \ge \text{height}(v_2)$, so that $\text{height}(v) = \text{height}(v_1) + 1$. Let $\mathcal{H}_{v_1}$ and $\mathcal{H}_{v_2}$ be the sets of hyperplanes corresponding to the children v_1 and v_2. By the inductive hypothesis, these sets are each of size at most

$$s := \ell^{d^{\text{height}(v_1)}} \left(\ell 4^d\right)^{\left(d^{\text{height}(v_1)}-1\right)/(d-1)}$$

Letting $\mathcal{H} = \mathcal{H}_{v_1} \cup \mathcal{H}_{v_2}$, we are guaranteed that for every connected component of $\mathbb{R}^d \setminus \mathcal{H}$, the alignment and consensus string computed by $M_{\boldsymbol{\rho}}$ for both children v_1 and v_2 is constant. Based on work by Buck [28], we know that there are at most $(2s+1)^d \le (3s)^d$ connected components of $\mathbb{R}^d \setminus \mathcal{H}$. For each region, by the same argument as in the proof of Lemma 4.1, there are an additional ℓ hyperplanes that partition the region into subregions where the outcome of the pairwise merge

at node v is constant. Therefore, there is a set $\mathcal{H}_v$ of at most

$$\begin{aligned}
\ell(3s)^d + 2s &\le \ell(4s)^d \\
&= \ell\left(4\ell^{d^{\text{height}(v_1)}}\left(\ell 4^d\right)^{\left(d^{\text{height}(v_1)}-1\right)/(d-1)}\right)^d \\
&= \ell^{d^{\text{height}(v_1)+1}}\left(\ell 4^d\right)^{\left(d^{\text{height}(v_1)+1}-d\right)/(d-1)+1} \\
&= \ell^{d^{\text{height}(v_1)+1}}\left(\ell 4^d\right)^{\left(d^{\text{height}(v_1)+1}-1\right)/(d-1)} \\
&= \ell^{d^{\text{height}(v)}}\left(\ell 4^d\right)^{\left(d^{\text{height}(v)}-1\right)/(d-1)}
\end{aligned}$$

hyperplanes where for every connected component of $\mathbb{R}^d \setminus \mathcal{H}$, the alignment and consensus string computed by $M_{\boldsymbol{\rho}}$ at v is invariant. □

Applying Claim C.10 to the root of the guide tree, the function $\boldsymbol{\rho} \mapsto M_{\boldsymbol{\rho}}(S_1, \ldots, S_\kappa, G)$ is piecewise constant with

$$\ell^{d^{\text{height}(G)}}\left(\ell 4^d\right)^{\left(d^{\text{height}(G)}-1\right)/(d-1)}$$

linear boundary functions. The lemma then follows from the following chain of inequalities:

$$\begin{aligned}
\ell^{d^{\text{height}(G)}}\left(\ell 4^d\right)^{\left(d^{\text{height}(G)}-1\right)/(d-1)} &\le \ell^{d^{\text{height}(G)}}\left(\ell 4^d\right)^{d^{\text{height}(G)}} \\
&= \ell^{2d^{\text{height}(G)}} 4^{d^{\text{height}(G)+1}} \\
&= \left(4^{n\kappa}(n\kappa)^{4n\kappa+2}\right)^{2d^{\text{height}(G)}} 4^{d^{\text{height}(G)+1}} \\
&= \left(4^{n\kappa}\,(n\kappa)^{4n\kappa+2}\right)^{2d^{\text{height}(G)}} 4^{d^{\text{height}(G)+1}} \\
&\le \left(4^{n\kappa}\,(n\kappa)^{4n\kappa+2}\right)^{2d^{\eta}} 4^{d^{\eta+1}}.
\end{aligned}$$

□

This lemma together with Lemma 3.10 implies the following corollary.

Corollary C.11. *The pseudo-dimension of $\mathcal{U}$ is $O\left(d^{\eta+1} n\kappa \ln(n\kappa) + d^{\eta+2}\right)$.*

Therefore, the pseudo-dimension grows only linearly in n and quadratically in κ in the affine-gap model ($d = 3$) when the guide tree is balanced ($\eta \le \log \kappa$).

D Additional Details about RNA Folding (Section 4.2)

Lemma 4.6. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho : S \mapsto u(S, A_\rho(S)) \mid \rho \in \mathbb{R}\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^2)$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho < a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$.*

Proof. Fix a sequence S. Let Φ 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 ϕ_k be the folding of length k that maximizes the right-hand-side of Equation (13):

$$\phi_k = \text{argmax}_{\phi : |\phi| = k} \sum_{(i,j) \in \phi} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1) \in \phi\}}.$$

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 ϕ will be the folding returned by the algorithm $A_\rho(S)$ if and only if

$$\begin{aligned} &\rho\,|\phi| + (1-\rho) \sum_{(i,j)\in\phi} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi\}} \\ \geq\ &\rho\,|\phi'| + (1-\rho) \sum_{(i,j)\in\phi'} M_{S[i],S[j],S[i-1],S[j+1]} \mathbb{I}_{\{(i-1,j+1)\in\phi'\}} \end{aligned} \tag{18}$$

for all $\phi' \in \Phi \setminus \{\phi\}$. Since these functions are linear in ρ, 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 $\rho \in [\rho_i, \rho_{i+1})$. By definition of the dual, this means that $u_S^*(u_\rho) = u_\rho(S) = c_i$ as well.

We now use this structure to show that the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, n^2)$-piecewise decomposable, as per Definition 3.2. Recall that $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho<a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$. We claim that there exists a function $f^{(\boldsymbol{b})} \in \mathcal{F}$ for every vector $\boldsymbol{b} \in \{0,1\}^T$ such that for every $\rho \in [0,1]$,

$$u_S^*(u_\rho) = \sum_{\boldsymbol{b}\in\{0,1\}^T} \mathbb{I}_{\{g_{\rho_i}(u_\rho)=b[i], \forall i\in[T]\}} f^{(\boldsymbol{b})}(u_\rho). \tag{19}$$

To see why, suppose $\rho \in [\rho_i, \rho_{i+1})$ for some $i \in [T]$. Then $g_{\rho_j}(u_\rho) = \mathbb{I}_{\{\rho\leq\rho_j\}} = 1$ for all $j \geq i+1$ and $g_{\rho_j}(u_\rho) = \mathbb{I}_{\{\rho\leq\rho_j\}} = 0$ for all $j \leq i$. Let $\boldsymbol{b}_i \in \{0,1\}^T$ be the vector that has only 0's in its first i coordinates and all 1's in its remaining $T-i$ coordinates. For all $i \in [T]$, we define $f^{(\boldsymbol{b}_i)} = f_{c_i}$, so $f^{(\boldsymbol{b}_i)}(u_\rho) = c_i$ for all $\rho \in [0,1]$. For any other $\boldsymbol{b}$, we set $f^{(\boldsymbol{b})} = f_0$, so $f^{(\boldsymbol{b})}(u_\rho) = 0$ for all $\rho \in [0,1]$. Therefore, Equation (19) holds. □

E Additional Details about Predicting TADs (Section 4.3)

Lemma 4.8. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_\rho : M \mapsto u(M, A_\rho(M)) \mid \rho \in \mathbb{R}\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, 2n^2 4^{n^2})$-piecewise decomposable, where $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho<a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$.*

Proof. Fix a matrix M. We begin by rewriting Equation (14) 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 ρ.

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 $A_{\boldsymbol{\rho}}$ 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 ρ over the positive reals, the TAD set returned by algorithm $A_\rho(M)$ will only change when

$$\sum_{(i,j)\in T} \frac{c_{ij}}{(j-i)^\rho} - \sum_{(i',j')\in T'} \frac{c_{i'j'}}{(j'-i')^\rho} = 0, \tag{20}$$

for some $T, T' \in \mathcal{T}$. As a result of Rolle's Theorem (Corollary A.3), we know that Equation (20) 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 ρ 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 of the dual, this means that $u^*_M(u_\rho) = u_\rho(M) = c_i$ as well.

We now use this structure to show that the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, 2n^2 4^{n^2})$-piecewise decomposable, as per Definition 3.2. Recall that $\mathcal{G} = \{g_a : \mathcal{U} \to \{0,1\} \mid a \in \mathbb{R}\}$ consists of threshold functions $g_a : u_\rho \mapsto \mathbb{I}_{\{\rho<a\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_\rho \mapsto c$. We claim that there exists a function $f^{(\boldsymbol{b})} \in \mathcal{F}$ for every vector $\boldsymbol{b} \in \{0,1\}^t$ such that for every $\rho \geq 0$,

$$u^*_M(u_\rho) = \sum_{\boldsymbol{b}\in\{0,1\}^t} \mathbb{I}_{\{g_{\rho_i}(u_\rho)=b[i], \forall i\in[t]\}} f^{(\boldsymbol{b})}(u_\rho). \tag{21}$$

To see why, suppose $\rho \in [\rho_i, \rho_{i+1})$ for some $i \in [t]$. Then $g_{\rho_j}(u_\rho) = \mathbb{I}_{\{\rho\leq\rho_j\}} = 1$ for all $j \geq i+1$ and $g_{\rho_j}(u_\rho) = \mathbb{I}_{\{\rho\leq\rho_j\}} = 0$ for all $j \leq i$. Let $\boldsymbol{b}_i \in \{0,1\}^t$ be the vector that has only 0's in its first i coordinates and all 1's in its remaining $t-i$ coordinates. For all $i \in [t]$, we define $f^{(\boldsymbol{b}_i)} = f_{c_i}$, so $f^{(\boldsymbol{b}_i)}(u_\rho) = c_i$ for all $\rho \in [0,1]$. For any other $\boldsymbol{b}$, we set $f^{(\boldsymbol{b})} = f_0$, so $f^{(\boldsymbol{b})}(u_\rho) = 0$ for all $\rho \in [0,1]$. Therefore, Equation (21) holds. □

F Parameterized Voting Mechanisms

A large body of research in economics studies how to design *mechanisms* that help groups of agents come to collective decisions. For example, when a town's residents want to build a public resource such as a park, pool, or skating rink, how should they choose what to build (as in participatory democracy [e.g., 52])? 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 mechanism that best answers these questions; the optimal mechanism depends on the setting at hand.

We study a family of mechanisms called ***neutral affine maximizers (NAMs)*** [91, 97, 106]. 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 $\boldsymbol{v} \in \mathbb{R}^{nm}$. This distributional assumption is standard in mechanism design [99].

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 [32, 60, 121]. 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

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

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

Lemma F.1. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}, \{i \mid \rho[i] = 0\} \neq \emptyset\}$. The dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, m^2)$-piecewise decomposable, where $\mathcal{G} = \{g_{\boldsymbol{a}} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^n\}$ consists of halfspace indicators $g_{\boldsymbol{a}} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{\rho} \cdot \boldsymbol{a} \leq 0\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$.*

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

$$\sum_{i=1}^n \rho[i] v_i(j) > \sum_{i=1}^n \rho[i] v_i(j'). \tag{22}$$

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 $u_{\boldsymbol{\rho}}(\boldsymbol{v}) = c_R$ for all $\boldsymbol{\rho} \in R$. By definition of the dual, this means that $u^*_{\boldsymbol{v}}(u_{\boldsymbol{\rho}}) = u_{\boldsymbol{\rho}}(\boldsymbol{v}) = c_R$ as well.

We now use this structure to show that the dual class $\mathcal{U}^*$ is $(\mathcal{F}, \mathcal{G}, m^2)$-piecewise decomposable, as per Definition 3.2. Recall that $\mathcal{G} = \{g_{\boldsymbol{a}} : \mathcal{U} \to \{0,1\} \mid \boldsymbol{a} \in \mathbb{R}^n\}$ consists of halfspace indicator functions $g_{\boldsymbol{a}} : u_{\boldsymbol{\rho}} \mapsto \mathbb{I}_{\{\boldsymbol{a} \cdot \boldsymbol{\rho} < 0\}}$ and $\mathcal{F} = \{f_c : \mathcal{U} \to \mathbb{R} \mid c \in \mathbb{R}\}$ consists of constant functions $f_c : u_{\boldsymbol{\rho}} \mapsto c$. For each pair of alternatives $j, j' \in \mathcal{L}$, let $g^{(j,j')} \in \mathcal{G}$ correspond to the halfspace represented in Equation (22). Order these $k := \binom{m}{2}$ functions arbitrarily as $g^{(1)}, \ldots, g^{(k)}$. Every connected component R of $\mathbb{R}^n \setminus \mathcal{H}$ corresponds to a sign pattern of the k hyperplanes. For a given region R, let $\boldsymbol{b}_R \in \{0,1\}^k$ be the corresponding sign pattern. Define the function $f^{(\boldsymbol{b}_R)} \in \mathcal{F}$ as $f^{(\boldsymbol{b}_R)} = f_{c_R}$, so $f^{(\boldsymbol{b}_R)}(u_{\boldsymbol{\rho}}) = c_R$ for all $\boldsymbol{\rho} \in \mathbb{R}^n$. Meanwhile, for every vector $\boldsymbol{b}$ not corresponding to a sign pattern of the k hyperplanes, let $f^{(\boldsymbol{b})} = f_0$, so $f^{(\boldsymbol{b})}(u_{\boldsymbol{\rho}}) = 0$ for all $\boldsymbol{\rho} \in \mathbb{R}^n$. In this way, for every $\boldsymbol{\rho} \in \mathbb{R}^n$,

$$u^*_{\boldsymbol{v}}(u_{\boldsymbol{\rho}}) = \sum_{\boldsymbol{b} \in \{0,1\}^k} \mathbb{I}_{\{g^{(i)}(u_{\boldsymbol{\rho}}) = b[i], \forall i \in [k]\}} f^{(\boldsymbol{b})}(u_{\boldsymbol{\rho}}),$$

as desired. □

Theorem 3.3 and Lemma F.1 imply the following corollary

Corollary F.2. *The pseudo-dimension of $\mathcal{U}$ is $O(n \ln m)$.*

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 F.3. *Let $\mathcal{U}$ be the set of functions $\mathcal{U} = \{u_{\boldsymbol{\rho}} \mid \boldsymbol{\rho} \in \mathbb{R}^n_{\geq 0}, \{\rho[i] \mid i = 0\} \neq \emptyset\}$. Then* $\text{Pdim}(\mathcal{U}) \geq \frac{n}{2}$.

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 ϵ be an arbitrary number in $(0, \frac{1}{2})$. For each $\ell \in [N]$, define agent i's values for the first and second alternatives under the ℓ^{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 $u_{\boldsymbol{\rho}}(\boldsymbol{v}^{(\ell)}) = \epsilon$ because there is always exactly one agent who has a value of ϵ 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 $u_{\boldsymbol{\rho}}(\boldsymbol{v}^{(\ell)}) = 1$ 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 ℓ in favor of the values of agent $\frac{n}{2} + \ell$. After all, under $\boldsymbol{v}^{(\ell)}$, agent ℓ has a value of 1 for the first alternative and agent $\frac{n}{2} + \ell$ has a value of ϵ 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 ℓ. Specifically, we define $\boldsymbol{\rho} \in \{0, 1\}^n$ as follows: for all $\ell \in [N] = [\frac{n}{2}]$, if $b_\ell = 0$, then $\rho[\ell] = 0$ and $\rho[\frac{n}{2} + \ell] = 1$ and if $b_\ell = 1$, then $\rho[\ell] = 1$ and $\rho[\frac{n}{2} + \ell] = 0$. All other entries of $\boldsymbol{\rho}$ are set to 0.

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

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

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 F.3 implies that the pseudo-dimension upper bound from Lemma F.1 is tight up to logarithmic factors.

Experiments

In this section, we provide similar experiments as those in Section 6.2, but in the context of NAMs. We present a simple subset of NAMs with a small estimation error upper bound. We then experimentally demonstrate that the true estimation error of the class of NAMs is larger than this simple subset's estimation error. Therefore, it is crucial to calculate a class's intrinsic complexity in order to provide accurate guarantees. These experiments further illustrate Observation 6.2.

Our simple set of mechanisms is defined as follows. One agent is selected to be the sink agent (a sink agent i has the weight $\rho[i] = 0$), and every other agent's weight is set to 1. In other words, this class is defined by the set of all vectors $\boldsymbol{\rho} \in \{0, 1\}^n$ where exactly one component of $\boldsymbol{\rho}$ is equal to zero. We use the notation $\mathcal{A}_0$ to denote this simple class, $\mathcal{A}_{NAM}$ to denote the set of all NAMs and $u_{\boldsymbol{\rho}}(\boldsymbol{v})$ to denote the social welfare of the NAM parameterized by $\boldsymbol{\rho}$ given the valuation vector $\boldsymbol{v}$.

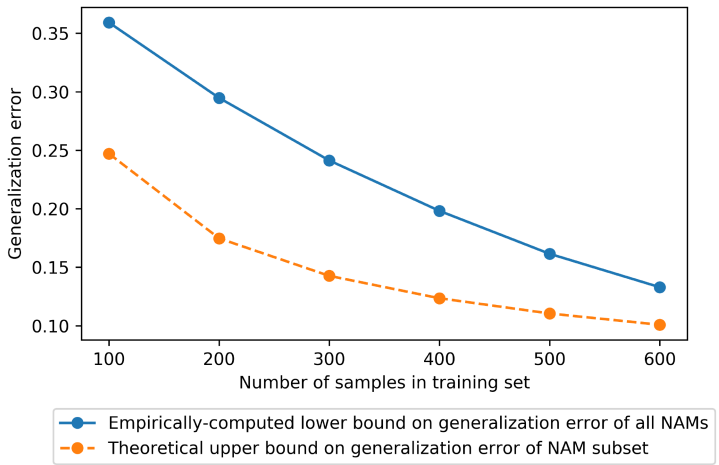


Fig. 11. Neutral affine maximizer experiments. We vary the size of the training set, N, along the x-axis. The orange dashed line is our upper bound on the estimation error of the simple subset of NAMs $\mathcal{A}_0$, $\sqrt{\frac{\ln(200n)}{2N}}$ (Equation (23)). The blue solid line lower bounds the true estimation error of the entire class of NAMs $\mathcal{A}_{NAM}$ over the Jester dataset. For several choices of $N \in [100, 600]$, we compute this lower bound by drawing a set of N training instances, finding a mechanism in $\mathcal{A}_{NAM}$ with high average social welfare over the training set, and calculating the mechanism's estimation error (the difference between its average social welfare and expected social welfare). For scale, estimation error is a quantity in the range $[0, 1]$.

Since there are n NAMs in $\mathcal{A}_0$, a Hoeffding and union bound tells us that for any distribution $\mathcal{D}$, with probability 0.99 over the draw of N valuation vectors $\mathcal{S} \sim \mathcal{D}^N$, for all n parameter vectors $\boldsymbol{\rho}$,

$$\left| \mathop{\mathbb{E}}_{\boldsymbol{v} \sim \mathcal{D}} \left[u_{\boldsymbol{\rho}}(\boldsymbol{v}) \right] - \frac{1}{N} \sum_{\boldsymbol{v} \in \mathcal{S}} u_{\boldsymbol{\rho}}(\boldsymbol{v}) \right| \leq \sqrt{\frac{\ln(200n)}{2N}}. \tag{23}$$

This is the orange dashed line in Figure 11. Meanwhile, as we proved earlier in this section, the pseudo-dimension of the class of all NAMs $\mathcal{A}_{NAM}$ is $\tilde{\Theta}(n)$, so it is a more complex set of mecha-

nisms than $\mathcal{A}_0$. To experimentally compute the lower bound on the true estimation error of the class of all NAMs $\mathcal{A}_{NAM}$, we identify a distribution such that the class has high estimation error, as we describe below.

Distribution. As in the previous section, we use the Jester Collaborative Filtering Dataset [54] to define our distribution. This dataset consists of ratings from 24,983 users of 100 jokes. The users' ratings are in the range $[-10, 10]$, so we normalize their ratings to lie in the interval $[-0.5, 0.5]$. We begin by selecting two jokes (jokes #7 and #15) such that—at a high level—a large number of agents either like the first joke a lot and do not like the second joke, or do not like the first joke and like the second joke a medium amount. We explain the intuition behind this choice below. Specifically, we split the agents into two groups: in the first group, the agents rated joke 1 at least 0.35 and rated joke 2 at most 0, and in the second group, the agents rated joke 1 at most 0 and rated joke 2 between 0 and 0.15. We call the set of ratings corresponding to the first group $A_1 \subseteq \mathbb{R}^2$ and those corresponding to the second group $A_2 \subseteq \mathbb{R}^2$. The set A_1 has size 870 and A_2 has size 1677.

We use A_1 and A_2 to define a distribution $\mathcal{D}$ over the valuations of $n = 1000$ agents for two jokes. The support of $\mathcal{D}$ consists of 500 valuation vectors $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(500)} \in \mathbb{R}^{2\times 1000}$. For $i \in [500]$, $\boldsymbol{v}^{(i)}$ is defined as follows. The values of agent i for the two jokes, $(v_i^{(i)}(1), v_i^{(i)}(2))$, are chosen uniformly at random from A_1 and the values of agent $500 + i$ are chosen uniformly at random from A_2. Every other agent i has a value of $v_i^{(i)}(1) = v_i^{(i)}(2) = 0$.

Parameter vector with poor estimation error. Given a set of samples $\mathcal{S} \subseteq \{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(500)}\}$, we define a parameter vector $\boldsymbol{\rho} \in \{0, 1\}^{1000}$ with high estimation error as follows: for all $i \in [500]$,

$$\rho[i] = \begin{cases} 1 & \text{if } \boldsymbol{v}^{(i)} \in \mathcal{S} \\ 0 & \text{otherwise} \end{cases} \qquad \text{and } \rho[500 + i] = \begin{cases} 0 & \text{if } \boldsymbol{v}^{(i)} \notin \mathcal{S} \\ 1 & \text{otherwise.} \end{cases} \tag{24}$$

Intuitively, this parameter vector[8] has high estimation error for the following reason. Suppose $\boldsymbol{v}^{(i)} \in \mathcal{S}$. The only agents with nonzero values in $\boldsymbol{v}^{(i)}$ are agent i and agent $500 + i$. Since $\boldsymbol{v}^{(i)} \in \mathcal{S}$, $\rho[i] = 1$ and $\rho[500 + i] = 0$. Therefore, agent i's favorite joke is selected. Since agent i's values are from the set A_1, they have a value of at least 0.35 for joke 1 and a value of at most 0 for joke 2. Therefore, joke 1 will be the selected joke. Meanwhile, by the same reasoning for every $\boldsymbol{v}^{(i)} \notin \mathcal{S}$, if we run the NAM defined by $\boldsymbol{\rho}$, joke 2 will be the selected joke. In expectation over $\mathcal{D}$, joke 1 has a significantly higher social welfare than joke 2. Therefore, the NAM defined by $\boldsymbol{\rho}$ will have a high average social welfare over the samples in $\mathcal{S}$ but a low expected social welfare, which means it will have high estimation error. We illustrate this intuition in our experiments.

Experimental procedure. We repeat the following experiment 100 times. For various choices of $N \in [600]$, we draw a set of samples $\mathcal{S} \sim \mathcal{D}^N$, compute the parameter vector $\boldsymbol{\rho}$ defined by Equation (24), and compute the difference between the average social welfare of $\boldsymbol{\rho}$ over $\mathcal{S}$ and its expected social welfare. We plot the difference averaged over all 100 runs.

Discussion. These experiments demonstrate that although the simple and complex NAM families $\mathcal{A}_{NAM}$ and $\mathcal{A}_0$ are artificially similar (they are both defined by the m agent weights), the complex family $\mathcal{A}_{NAM}$ requires far more samples to avoid overfitting than the simple family $\mathcal{A}_0$. This illustrates the importance of using our pseudo-dimension bounds to provide accurate guarantees.

[8] Although this parameter vector has high average social welfare over the samples, it may set multiple agents to be sink agents, which may be wasteful. We leave the problem of finding a parameter vector with high estimation error and only a single sink agent to future research.

References

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

[2] Saba Ahmadi, Hedyeh Beyhaghi, Avrim Blum, and Keziah Naggita. 2022. Setting fair incentives to maximize improvement. arXiv:2203.00134. Retrieved from https://arxiv.org/abs/2203.00134

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

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

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

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

[7] Maria-Florina Balcan, Travis Dick, and Wesley Pegden. 2020. Semi-bandit optimization in the dispersed setting. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

[8] Maria-Florina Balcan. 2020. Data-driven algorithm design. In *Proceedings of the Beyond Worst Case Analysis of Algorithms*, Tim Roughgarden (Ed.). Cambridge University Press.

[9] Maria-Florina Balcan, Avrim Blum, Jason D. Hartline, and Yishay Mansour. 2005. Mechanism design via machine learning. In *Proceedings of the Annual Symposium on Foundations of Computer Science*. 605–614.

[10] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. 2019. How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. arXiv:1908.02894. Retrieved from https://arxiv.org/abs/1908.02894

[11] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. 2021. How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. In *Proceedings of the Annual Symposium on Theory of Computing*.

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

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

[14] Maria-Florina Balcan, Travis Dick, and Colin White. 2018. Data-driven clustering via parameterized Lloyd's families. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 10641–10651.

[15] Maria-Florina Balcan, Mikhail Khodak, Dravyansh Sharma, and Ameet Talwalkar. 2022. Provably tuning the Elastic-Net across instances. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

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

[17] Maria-Florina Balcan, Siddharth Prasad, Tuomas Sandholm, and Ellen Vitercik. 2021. Sample complexity of tree search configuration: Cutting planes and beyond. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[18] Maria-Florina Balcan, Siddharth Prasad, Tuomas Sandholm, and Ellen Vitercik. 2022. Structural analysis of branch-and-cut and the learnability of gomory mixed integer cuts. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[19] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. [n.d.]. Generalization guarantees for multi-item profit maximization: Pricing, auctions, and randomized mechanisms. *Operations Research* ([n. d.]), (to appear).

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

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

[22] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. 2021. Generalization in portfolio-based algorithm selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[23] Maria-Florina Balcan and Dravyansh Sharma. 2021. Data driven semi-supervised learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[24] Peter Bartlett, Piotr Indyk, and Tal Wagner. 2022. Generalization bounds for data-driven numerical linear algebra. In *Proceedings of the Conference on Learning Theory*.

[25] Jon Louis Bentley, David S. Johnson, Frank Thomson Leighton, Catherine C. McGeoch, and Lyle A. McGeoch. 1984. Some unexpected expected behavior results for bin packing. In *Proceedings of the Annual Symposium on Theory of Computing*. 279–288.

[26] Avrim Blum, Chen Dan, and Saeed Seddighin. 2021. Learning complexity of simulated annealing. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.
[27] Sébastien Bubeck, Nikhil R. Devanur, Zhiyi Huang, and Rad Niazadeh. 2017. Online auctions and multi-scale online learning. *Proceedings of the ACM Conference on Economics and Computation* (2017).
[28] Robert Creighton Buck. 1943. Partition of space. *The American Mathematical Monthly* 50 (1943), 541–544.
[29] Yang Cai and Constantinos Daskalakis. 2017. Learning multi-item auctions with (or without) samples. In *Proceedings of the Annual Symposium on Foundations of Computer Science*.
[30] Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. 2020. Pandora's box with correlations: Learning and approximation. In *Proceedings of the Annual Symposium on Foundations of Computer Science*.
[31] Antonia Chmiela, Elias B. Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. 2021. Learning to schedule heuristics in branch-and-bound. arXiv:2103.10294. Retrieved from https://arxiv.org/abs/2103.10294
[32] Ed H. Clarke. 1971. Multipart pricing of public goods. *Public Choice* 11 (1971), 17–33.
[33] Vincent Cohen-Addad and Varun Kanade. 2017. Online optimization of smoothed piecewise constant functions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.
[34] Richard Cole and Tim Roughgarden. 2014. The sample complexity of revenue maximization. In *Proceedings of the Annual Symposium on Theory of Computing*.
[35] Dan DeBlasio and John D. Kececioglu. 2018. *Parameter Advising for Multiple Sequence Alignment*. Springer.
[36] Nikhil R. Devanur, Zhiyi Huang, and Christos-Alexandros Psomas. 2016. The sample complexity of auctions with side information. In *Proceedings of the Annual Symposium on Theory of Computing*.
[37] Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. 2020. Secretaries with advice. arXiv:2011.06726. Retrieved from https://arxiv.org/abs/2011.06726
[38] Talya Eden, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, Sandeep Silwal, and Tal Wagner. 2021. Learning-based support estimation in sublinear time. In *Proceedings of the International Conference on Learning Representations* .
[39] Robert C. Edgar. 2010. Quality measures for protein alignment benchmarks. *Nucleic Acids Research* 38, 7 (2010), 2145–2153.
[40] Edith Elkind. 2007. Designing and learning optimal finite support auctions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*.
[41] Marc Etheve, Zacharie Alès, Côme Bissuel, Olivier Juan, and Safia Kedad-Sidhoum. 2020. Reinforcement learning for variable selection in a branch and bound algorithm. Springer, 176–185.
[42] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. 2020. Learning augmented energy minimization via speed scaling. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.
[43] Étienne Bamas, Andreas Maggiori, and Ola Svensson. 2020. The primal-dual method for learning augmented algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.
[44] Uriel Feige and Michael Langberg. 2006. The RPR^2 rounding technique for semidefinite programs. *Journal of Algorithms* 60, 1 (2006), 1–23.
[45] Da-Fei Feng and Russell F. Doolittle. 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution* 25, 4 (1987), 351–360.
[46] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. 2020. MIPaaL: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 1504–1511.
[47] David Fernández-Baca, Timo Seppäläinen, and Giora Slutzki. 2004. Parametric multiple sequence alignment and phylogeny construction. *Journal of Discrete Algorithms* 2, 2 (2004), 271–287.
[48] Darya Filippova, Rob Patro, Geet Duggal, and Carl Kingsford. 2014. Identification of alternative topological domains in chromatin. *Algorithms for Molecular Biology* 9, 1 (2014), 14.
[49] Nikolaus Furian, Michael O'sullivan, Cameron Walker, and Eranda Çela. 2021. A machine learning-based branch and price algorithm for a sampled vehicle routing problem. *OR Spectrum* (2021), 1–40.
[50] Vikas Garg and Adam Kalai. 2018. Supervising unsupervised learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.
[51] Andrew Gilpin and Tuomas Sandholm. 2011. Information-theoretic approaches to branching in search. *Discrete Optimization* 8, 2 (2011), 147–159. Early version in IJCAI-07.
[52] Ashish Goel, Anilesh K. Krishnaswamy, Sukolsak Sakshuwong, and Tanja Aitamurto. 2019. Knapsack voting for participatory budgeting. *ACM Transactions on Economics and Computation* 7, 2 (2019), 1–27.
[53] Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* 42, 6 (1995), 1115–1145.
[54] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. 2001. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval* 4, 2 (2001), 133–151.

[55] Paul Goldberg and Mark Jerrum. 1993. Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers. In *Proceedings of the Conference on Learning Theory*.
[56] Yannai A. Gonczarowski and Noam Nisan. 2017. Efficient empirical revenue maximization in single-parameter auction environments. In *Proceedings of the Annual Symposium on Theory of Computing*. 856–868.
[57] Yannai A. Gonczarowski and S. Matthew Weinberg. 2018. The sample complexity of up-to-ε multi-dimensional revenue maximization. In *Proceedings of the Annual Symposium on Foundations of Computer Science*.
[58] Yannai A. Gonczarowski and S. Matthew Weinberg. 2021. The sample complexity of up-to-ε multi-dimensional revenue maximization. *Journal of the ACM* 68, 3 (2021), 1–28.
[59] Osamu Gotoh. 1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162, 3 (1982), 705 – 708.
[60] Theodore Groves. 1973. Incentives in teams. *Econometrica* 41 (1973), 617–631.
[61] Chenghao Guo, Zhiyi Huang, and Xinzhi Zhang. 2019. Settling the sample complexity of single-parameter revenue maximization. *Proceedings of the Annual Symposium on Theory of Computing* (2019).
[62] Rishi Gupta and Tim Roughgarden. 2017. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing* 46, 3 (2017), 992–1017.
[63] Dan Gusfield, Krishnan Balasubramanian, and Dalit Naor. 1994. Parametric optimization of sequence alignment. *Algorithmica* 12, 4-5 (1994), 312–326.
[64] Dan Gusfield and Paul Stelling. 1996. Parametric and inverse-parametric sequence alignment with XPARAL. In *Proceedings of the Methods in Enzymology*. Elsevier, 481–494.
[65] Jason Hartline and Samuel Taggart. 2016. Non-revelation mechanism design. arXiv:1608.01875. Retrieved from https://arxiv.org/abs/1608.01875
[66] Desmond G. Higgins and Paul M. Sharp. 1988. CLUSTAL: A package for performing multiple sequence alignment on a microcomputer. *Gene* 73, 1 (1988), 237–244.
[67] Robert W. Holley, Jean Apgar, George A. Everett, James T. Madison, Mark Marquisee, Susan H. Merrill, John Robert Penswick, and Ada Zamir. 1965. Structure of a ribonucleic acid. *Science* 147, 3664 (1965), 1462–1465.
[68] Eric Horvitz, Yongshao Ruan, Carla Gomez, Henry Kautz, Bart Selman, and Max Chickering. 2001. A Bayesian approach to tackling hard computational problems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.
[69] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-based frequency estimation algorithms. In *Proceedings of the International Conference on Learning Representations*.
[70] Zhiyi Huang, Yishay Mansour, and Tim Roughgarden. 2015. Making the most of your samples. In *Proceedings of the ACM Conference on Economics and Computation*.
[71] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36, 1 (2009), 267–306.
[72] Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrović, and Ronitt Rubinfeld. 2020. Online page migration with ML advice. arXiv:2006.05028. Retrieved from https://arxiv.org/abs/2006.05028
[73] Raj Iyer, David Karger, Hariharan Rahul, and Mikkel Thorup. 2002. An experimental study of polylogarithmic, fully dynamic, connectivity algorithms. *ACM Journal of Experimental Algorithmics* 6 (2002), 4–es.
[74] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. 2010. ISAC-instance-specific algorithm configuration.. In *Proceedings of the European Conference on Artificial Intelligence*.
[75] John D Kececioglu and Dean Starrett. 2004. Aligning alignments exactly. In *Proceedings of the Annual International Conference on Computational Molecular Biology*. 85–96.
[76] Mikhail Khodak, Maria-Florina Balcan, Ameet Talwalkar, and Sergei Vassilvitskii. 2022. Learning predictions for algorithms with predictions. In *Proceedings of the Annual Conference on Neural Information Processing Systems* .
[77] Eagu Kim and John Kececioglu. 2007. Inverse sequence alignment from partial examples. *Proceedings of the International Workshop on Algorithms in Bioinformatics* (2007), 359–370.
[78] Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. 2017. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
[79] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. 2019. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *Proceedings of the Annual Conference on Neural Information Processing Systems* (2019).
[80] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. 2021. End-to-end constrained optimization learning: A survey. arXiv:2103.16378. Retrieved from https://arxiv.org/abs/2103.16378
[81] Ailsa H. Land and Alison G. Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society* (1960), 497–520.

[82] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. 2020. Learnable and instance-robust predictions for online matching, flows and load balancing. arXiv:2011.11743. Retrieved from https://arxiv.org/abs/2011.11743
[83] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. 2009. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM* 56, 4 (2009), 1–52.
[84] 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. 2009. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science* 326, 5950 (2009), 289–293. DOI: https://doi.org/10.1126/science.1181369
[85] Anton Likhodedov and Tuomas Sandholm. 2004. Methods for boosting revenue in combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence*. San Jose, CA, 232–237.
[86] Anton Likhodedov and Tuomas Sandholm. 2005. Approximating revenue-maximizing combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence*. Pittsburgh, PA.
[87] Jeff Linderoth and Martin Savelsbergh. 1999. A computational study of search strategies for mixed integer programming. *INFORMS Journal of Computing* 11 (1999), 173–187.
[88] Darío G. Lupiáñez, Malte Spielmann, and Stefan Mundlos. 2016. Breaking TADs: How alterations of chromatin domains result in disease. *Trends in Genetics* 32, 4 (2016), 225–237.
[89] Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive caching with machine learned advice. In *Proceedings of the International Conference on Machine Learning*.
[90] Catherine C. McGeoch. 2012. *A Guide to Experimental Algorithmics*. Cambridge University Press.
[91] Debasis Mishra and Arunava Sen. 2012. Roberts' theorem with neutrality: A social welfare ordering approach. *Games and Economic Behavior* 75, 1 (2012), 283–298.
[92] Michael Mitzenmacher. 2018. A model for learned bloom filters and optimizing by sandwiching. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 464–473.
[93] Mehryar Mohri and Andrés Muñoz. 2014. Learning theory and algorithms for revenue optimization in second price auctions with reserve. In *Proceedings of the International Conference on Machine Learning*.
[94] Jamie Morgenstern and Tim Roughgarden. 2015. On the pseudo-dimension of nearly optimal auctions. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.
[95] Jamie Morgenstern and Tim Roughgarden. 2016. Learning simple auctions. In *Proceedings of the Conference on Learning Theory*.
[96] Andrés Muñoz Medina and Sergei Vassilvitskii. 2017. Revenue optimization with approximate bid predictions. *Proceedings of the Annual Conference on Neural Information Processing Systems* (2017).
[97] Swaprava Nath and Tuomas Sandholm. 2019. Efficiency and budget balance in general quasi-linear domains. *Games and Economic Behavior* 113 (2019), 673 – 693.
[98] Saket Navlakha, James White, Niranjan Nagarajan, Mihai Pop, and Carl Kingsford. 2009. Finding biologically accurate clusterings in hierarchical tree decompositions using the variation of information. In *Proceedings of the Annual International Conference on Research in Computational Molecular Biology*. Springer, 400–417.
[99] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. 2007. *Algorithmic Game Theory*. Cambridge University press.
[100] Ruth Nussinov and Ann B. Jacobson. 1980. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences* 77, 11 (1980), 6309–6313.
[101] Lior Pachter and Bernd Sturmfels. 2004. Parametric inference for biological sequence analysis. *Proceedings of the National Academy of Sciences* 101, 46 (2004), 16138–16143. DOI: https://doi.org/10.1073/pnas.0406011101
[102] Lior Pachter and Bernd Sturmfels. 2004. Tropical geometry of statistical models. *Proceedings of the National Academy of Sciences* 101, 46 (2004), 16132–16137. DOI: https://doi.org/10.1073/pnas.0406010101
[103] David Pollard. 1984. *Convergence of Stochastic Processes*. Springer.
[104] Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. 2020. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. arXiv:2011.06069. Retrieved from https://arxiv.org/abs/2011.06069
[105] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving online algorithms via ML predictions. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 9661–9670.
[106] Kevin Roberts. 1979. The characterization of implementable social choice rules. In *Proceedings of the Aggregation and Revelation of Preferences*, J.-J. Laffont (Ed.). North-Holland Publishing Company.
[107] Tim Roughgarden and Okke Schrijvers. 2016. Ironing in the dark. In *Proceedings of the ACM Conference on Economics and Computation*.
[108] Shinsaku Sakaue and Taihei Oki. 2022. Sample complexity of learning heuristic functions for greedy-best-first and A* search. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[109] Tuomas Sandholm. 2013. Very-large-scale generalized combinatorial multi-attribute auctions: Lessons from conducting $60 Billion of sourcing. In *Proceedings of the Handbook of Market Design*, Zvika Neeman, Alvin Roth, and Nir Vulkan (Eds.). Oxford University Press.

[110] Tuomas Sandholm and Anton Likhodedov. 2015. Automated design of revenue-maximizing combinatorial auctions. *Operations Research* 63, 5 (2015), 1000–1025. Special issue on Computational Economics. Subsumes and extends over a AAAI-05 paper and a AAAI-04 paper.

[111] J. Michael Sauder, Jonathan W. Arthur, and Roland L. Dunbrack Jr. 2000. Large-scale comparison of protein sequence alignment algorithms with structure alignments. *Proteins: Structure, Function, and Bioinformatics* 40, 1 (2000), 6–22.

[112] Norbert Sauer. 1972. On the density of families of sets. *Journal of Combinatorial Theory, Series A* 13, 1 (1972), 145–147.

[113] Daniel Selsam and Nikolaj Bjørner. 2019. Guiding high-performance SAT solvers with unsat-core predictions. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*. Springer, 336–353.

[114] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

[115] Yifei Shen, Yuanming Shi, Jun Zhang, and Khaled B. Letaief. 2019. LORM: Learning to optimize for resource management in wireless networks with few training samples. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 665–679.

[116] Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. 2020. A general large neighborhood search framework for solving integer programs. In *Proceedings of the Annual Conference on Neural Information Processing Systems* .

[117] Vasilis Syrgkanis. 2017. A sample complexity measure with applications to learning optimal auctions. *Proceedings of the Annual Conference on Neural Information Processing Systems* (2017).

[118] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. 2020. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the International Conference on Machine Learning*.

[119] Timo Tossavainen. 2006. On the zeros of finite sums of exponential functions. *Australian Mathematical Society Gazette* 33, 1 (2006), 47–50.

[120] Vladimir Vapnik and Alexey Chervonenkis. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications* 16, 2 (1971), 264–280.

[121] William Vickrey. 1961. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance* 16 (1961), 8–37.

[122] Lusheng Wang and Tao Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology* 1, 4 (1994), 337–348.

[123] Michael S. Waterman, Temple F. Smith, and William A. Beyer. 1976. Some biological sequence metrics. *Advances in Mathematics* 20, 3 (1976), 367–387.

[124] Hugues Wattez, Frédéric Koriche, Christophe Lecoutre, Anastasia Paparrizou, and Sébastien Tabary. 2020. Learning variable ordering heuristics with multi-armed bandits and restarts.

[125] Alexander Wei and Fred Zhang. 2020. Optimal robustness-consistency tradeoffs for learning-augmented online algorithms. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[126] Gellért Weisz, András György, and Csaba Szepesvári. 2018. LeapsAndBounds: A method for approximately optimal algorithm configuration. In *Proceedings of the International Conference on Machine Learning*.

[127] Gellért Weisz, András György, and Csaba Szepesvári. 2019. CapsAndRuns: An improved method for approximately optimal algorithm configuration. In *Proceedings of the International Conference on Machine Learning*.

[128] Travis J. Wheeler and John D. Kececioglu. 2007. Multiple alignment by aligning alignments. *Bioinformatics* 23, 13 (07 2007), i559–i568.

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

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

[131] Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. 2021. Parameterizing branch-and-bound search trees to learn branching policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Received 21 April 2021; revised 15 January 2024; accepted 25 June 2024