Learning the best algorithm for max-cut, clustering, and other partitioning problems*

Maria-Florina Balcan Vaishnavh Nagarajan Ellen Vitercik Colin White

November 14, 2016

Abstract

Many data analysis problems are NP-hard, a reality that has motivated researchers to develop a wealth of approximation algorithms and heuristics over the past few decades. Max-cut, clustering, and many other partitioning problems fall into this camp, with widespread applications ranging from statistical physics to computational biology. In this work, we focus on two widely studied classes of algorithms that can be used for clustering and partitioning problems, including rich parametrized classes of SDP rounding algorithms and hierarchical clustering algorithms. The best algorithm to use often depends on the specific application or distribution over instances. A worst-case analysis is often used to compare algorithms, but this single number may be derived from atypical inputs which are not present in the particular problem domain, and thus this analysis may be misleading when determining which algorithm to apply. Therefore, it is necessary to develop optimization methods which return the algorithms and parameters best suited for typical inputs from the application at hand.

In this work, we address this problem for integer quadratic programming and clustering within a learning-theoretic framework where the learning algorithm is given samples from an application-specific distribution over problem instances and learns an algorithm which performs well over the distribution. We provide strong sample complexity guarantees and computationally efficient algorithms which optimize an algorithm family's parameters to best suit typical inputs from a particular application. We analyze these algorithm families using the learning-theoretic notion of pseudo-dimension. Along with upper bounds, we provide the first pseudo-dimension lower bounds in this line of work, which require an involved analysis of each algorithm family's overall performance on carefully constructed problem instances. Our bounds are tight and therefore nail down the intrinsic complexity of the algorithm classes we analyze, which consist of multi-stage optimization procedures and which are significantly richer and more complex than classes commonly used in learning theory. For example, even for classes of algorithms parameterized by a single parameter, we prove tight superconstant pseudo-dimension bounds. In this way, not only does our work contribute to the study of algorithm design and analysis by providing a theoretical grounding for application-specific algorithm selection, but it also pushes the boundaries of learning theory.

*Authors' addresses: {ninamf,vaishnavh,vitercik,crwhite}@cs.cmu.edu.

1 Introduction

Many NP-hard problems arise in a variety of diverse and oftentimes unrelated application domains. For example, clustering is a widely-studied NP-hard problem in unsupervised machine learning, used to group protein sequences by function, organize documents in databases by subject, and choose the best locations for fire stations or cell-phone towers throughout a city. Although the underlying objective is the same, a "typical problem instance" in one setting may be significantly different from that in another, causing approximation algorithms to have inconsistent performance across the different application domains.

We study the problem of characterizing which algorithms are best for which contexts, a task often referred to as *application-specific algorithm selection*. This line of work allows researchers to compare algorithms according to an application-specific metric, such as expected performance over their problem domain, rather than a more standard, application-agnostic metric, such as worst-case analysis. We conduct a learning-theoretic analysis through a framework introduced by Gupta and Roughgarden [14], wherein an application domain is modeled as a distribution over problem instances. We then fix an infinite class of approximation algorithms for that problem, and design computationally efficient algorithms which learn the approximation algorithm with the best performance with respect to the distribution, and therefore an algorithm which will perform well in the specific application domain. Gupta and Roughgarden focused on greedy and local search heuristics, and left as an open question what other classes of algorithms can be learned over.

In this light, we study infinite classes of semidefinite programming (SDP) rounding algorithms for approximating integer quadratic programs, as well as agglomerative clustering algorithms with dynamic programming. The SDP rounding algorithm classes we focus on are known to perform particularly well on the max-cut problem; they contain well-known algorithms such as the Goemans-Williamson (GW) max-cut approximation algorithm, which performs well in the worst-case [12], and more generally all s-linear rounding functions, which were introduced by Feige and Langberg [11]. We note that these approximation algorithms are randomized, whereas Gupta and Roughgarden's work focused on deterministic algorithms.

We also study infinite classes of clustering algorithms with a linkage-based step and a pruning step. We parameterize both of these modules separately, providing an infinite number of linkage-based procedures and an infinite number of pruning steps, and our clustering algorithm classes consist of all possible pairings of a linkage step and pruning step. The linkage-based modules include the classic single-, complete-, and average-linkage algorithms, known to return nearly-optimal clusterings in a variety of settings [3, 13, 5, 4]. The dynamic programming modules include the k-median, k-means, and k-center objectives, as well as objectives that minimize the distance to a ground truth clustering.

1.1 Problem descriptions and our results

Before describing our results, we provide a more formal overview of application-specific algorithm selection through a learning-theoretic lens, as introduced by Gupta and Roughgarden [14]. We fix a computational problem, such as max-cut or k-means clustering, and assume that there exists an unknown, application-dependent distribution $\mathcal{D}$ over the set of all problem instances Π. For example, the support of $\mathcal{D}$ might be a set of social networks, and the researcher's goal is to choose an algorithm with which to perform a series of clustering analyses. Next, we fix a class of algorithms $\mathcal{A}$. Given a cost function $\texttt{cost} : \mathcal{A} \times \Pi \to [0, H]$, the learner is tasked with finding an algorithm $h \in \mathcal{A}$ that approximately optimizes the expected cost with respect to the distribution $\mathcal{D}$, i.e., the one that approximately minimizes $\mathbb{E}_{x\sim\mathcal{D}}[\texttt{cost}(h, x)]$. We now formalize this notion.

Definition 1 ([14]). *A learning algorithm L (ϵ, δ)-learns* the algorithm class $\mathcal{A}$ with respect to the cost function `cost` *if, for every distribution $\mathcal{D}$ over Π, with probability at least $1-\delta$ over the choice of a sample $\mathcal{S} \sim \mathcal{D}^m$, L outputs an algorithm $\hat{h} \in \mathcal{A}$ such that $\mathbb{E}_{x\sim\mathcal{D}}\left[\texttt{cost}\left(\hat{h}, x\right)\right] - \min_{h\in\mathcal{A}}\{\mathbb{E}_{x\sim\mathcal{D}}[\texttt{cost}(h,x)]\} < \epsilon$. We require that the number of samples be polynomial in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, where n is an upper bound on the size of the problem instances in the support of $\mathcal{D}$.*[1] *Further, we say that L is* computationally efficient *if its running time is also polynomial in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$.*

We present algorithms which learn an approximately optimal algorithm for a given application domain, and we analyze the sample complexity of our algorithms. We use the learning-theoretic concept of *pseudo-dimension*, which we define formally in Section 2, in both our algorithm design and sample complexity analysis. Pseudo-dimension measures the complexity, or richness, of a function class, and extends the well-known *VC dimension* to real-valued functions.

SDP-based methods for integer quadratic programming. Many NP-hard problems, such as max-cut, max-2SAT, and correlation clustering, can be represented as an integer quadratic program of the following form. The input is an $n \times n$ matrix A with nonnegative diagonal entries and the output is a binary assignment to each variable in the set $X = \{x_1, \ldots, x_n\}$ which maximizes $\sum_{i,j\in[n]} a_{ij}x_ix_j$.[2] In this formulation, $x_i \in \{-1, 1\}$ for all $i \in [n]$. Integer quadratic programs have diverse applications, including capital budgeting and financial analysis [19], traffic message management problems [29], machine scheduling [1], and molecular conformation [22]. Max-cut is a special case of integer quadratic programming which has garnered significant attention in both theoretical and empirical communities in computer science. The seminal Goemans-Williamson algorithm is now a textbook example of semidefinite programming [12, 28, 26] and max-cut continues to arise in applications such as machine learning [27], circuit design [31], and computational biology [24]. In fact, libraries of max-cut algorithms have been the subject of empirical work in application-specific algorithm selection [10] and max-cut and integer quadratic programming in general have been used as a test bed for new ideas in heuristic design [18].

The best approximation algorithms for integer quadratic programming relax the problem to a semidefinite program, where the input is the same matrix A, but the output is a set of unit vectors maximizing $\sum_{i,j} a_{ij}\langle\vec{u}_i, \vec{u}_j\rangle$. The final step is to transform, or "round," the set of vectors into an assignment of the binary variables in X. This assignment corresponds to a feasible solution to the original integer quadratic program. There are infinitely many rounding techniques to choose from, many of which are randomized. These algorithms make up the class of *Random Projection, Randomized Rounding* algorithms (RPR^2), a general framework introduced by Feige and Langberg [11].

RPR^2 algorithms are known to perform well in theory and practice. When the integer quadratic program is a formulation of the max-cut problem, the class of RPR^2 algorithms contain the groundbreaking Goemans-Williamson algorithm, which achieves a 0.878 approximation ratio [12]. Assuming the unique games conjecture and $P \neq NP$, this approximation is optimal to within any additive constant [17]. More generally, if A is any real-valued $n \times n$ matrix with nonnegative diagonal entries, then there exists an RPR^2 algorithm that achieves an approximation ratio of $\Omega(1/\log n)$ [8], and in the worst case, this ratio is tight [2]. Finally, if A is positive semi-definite, then there exists an RPR^2 algorithm that achieves a $2/\pi$ approximation ratio [6].

[1] For example, in clustering, n is an upper bound on the number of points in a problem instance, and in integer quadratic programming, n is an upper bound on the number of variables in a problem instance.

[2] When the diagonal entries are allowed to be negative, the ratio between the semidefinite relaxation and the integral optimum can become arbitrarily large, so we restrict the domain to matrices with nonnegative diagonal entries.

We analyze several classes of RPR^2 rounding function classes, including s-linear [11], outward rotation [32], and $\tilde{\epsilon}$-discretized rounding functions [21]. For each class, we derive bounds on the number of samples needed to learn an approximately optimal rounding function with respect to an underlying distribution over problem instances. In particular, we prove that the pseudo-dimension of the class of s-linear RPR^2 algorithms is $\Theta(\log n)$ and the pseudo-dimension of the class of outward rotation RPR^2 algorithms is $O(\log n)$. Therefore, $O\left(\frac{1}{\epsilon^2}\left(\log n \log \frac{1}{\epsilon} + \log \frac{1}{\delta}\right)\right)$ samples are sufficient to (ϵ, δ)-learn the optimal algorithm in either of these classes. Finally, we show that $O\left(\frac{1}{\tilde{\epsilon}^2}\left(\frac{1}{\tilde{\epsilon}^2}\log \frac{1}{\epsilon} + \log \frac{1}{\delta}\right)\right)$ samples are required to (ϵ, δ)-learn the optimal algorithm in the class of $\tilde{\epsilon}$-discretized rounding functions. We also provide a computationally efficient learning algorithm for learning an approximately optimal s-linear or outward rotation rounding function with respect to the distribution.

Finally, our analyses are not specific to RPR^2 algorithms that first find an SDP embedding, though this is the most common technique. Rather, our results also apply more generally to any class of RPR^2 algorithms where the first step is to find some set of vectors on the unit sphere and then round those vectors to a binary solution using one of the rounding functions we study. This generalization has led to faster approximation algorithms with strong empirical performance. For example, in order to quickly find approximate solutions to the max-cut problem, Johansson et al. replace the SDP embedding with an embedding that that is faster to compute, and then run the standard GW algorithm in order to transform the vector solution into a graph cut [16].

Clustering by agglomerative algorithms with dynamic programming. Given a set of n datapoints and the pairwise distances between them, at a high level, the goal of clustering is to partition the points into groups such that distances within each group are minimized and distances between each group are maximized. A classic way to accomplish this task is to use an objective function. Common clustering objective functions include k-means, k-median, and k-center, which we define later on. We focus on a very general problem where the learner's main goal is to minimize an abstract cost function such as the cluster purity or the clustering objective function. We study infinite classes of two-step clustering algorithms consisting of a linkage-based step and a dynamic programming step. First, the algorithm runs one of an infinite number of linkage-based routines to construct a hierarchical tree of clusters. Next, the algorithm runs a dynamic programming procedure to find the pruning of this tree that minimizes one of an infinite number of clustering objectives. For example, if the clustering objective is the k-means objective, then the dynamic programming step will return the optimal k-means pruning of the cluster tree.

For the linkage-based procedure, we consider several natural parameterizations of agglomerative procedures which give rise to a spectrum of algorithms interpolating between the popular single-, average-, and complete-linkage procedures, known to perform nearly optimally in many settings [3, 4, 5, 13]. For the dynamic programming step, we study an infinite class of objectives which include the standard k-means, k-median, and k-center objectives. We show how to learn the best agglomerative algorithm and pruning objective function pair, thus extending our work to multiparameter algorithms. We provide tight pseudo-dimension bounds, ranging from $\Theta(\log n)$ for simpler algorithm classes to $\Theta(n)$ for more complex algorithm classes. This implies that few samples are required to (ϵ, δ)-learn the optimal algorithm in any of these classes. Finally, we present algorithms for learning over these classes.

Key Challenges. One of the key challenges in analyzing the pseudo-dimension of the algorithm classes we study is that to do so, we must develop deep insights into how changes to an algorithm's parameters effect the solution the algorithm returns on an arbitrary input. This obstacle is best exemplified by our clustering analysis, where the cost function could be, for example, the k-means or k-median objective function, or even the distance to some ground-truth clustering. As we range

over algorithm parameters, the cost of the returned clustering may vary unpredictably.

In this way, our algorithm analyses require more care than standard pseudo-dimension derivations commonly found in machine learning contexts. Typically, for well-understood function classes used in machine learning, such as linear separators or other smooth curves in Euclidean spaces, there is a simple mapping from the parameters of a specific hypothesis to its prediction on a given example and a close connection between the distance in the parameter space between two parameter vectors and the distance in function space between their associated hypotheses. Roughly speaking, it is necessary to understand this connection in order to determine how many significantly different hypotheses there are over the full range of parameters. Due to the inherent complexity of the classes we consider, connecting the parameter space to the space of approximation algorithms and their associated costs requires a much more delicate analysis. Indeed, the key technical part of our work involves understanding this connection from a learning theoretic perspective.

1.2 Additional related work

Application-specific algorithm selection has been studied in the artificial intelligence community for decades and is closely related to *automated algorithm configuration* and *algorithm portfolio selection*. The algorithm selection problem was first proposed by Rice in 1979 [23], and has since led to breakthroughs in diverse fields including combinatorial auctions [20], scientific computing [9], vehicle routing [7], and SAT [30], among many others. In this line of work, as in ours, the authors assume a distribution over problem instances, and they aim to design algorithms or portfolios of algorithms that will perform well in expectation over the distribution. They implement algorithm selection techniques which are often based on machine learning in order to satisfy their design goals. Arguably the most prominent algorithm selection system is SATZILLA, a SAT solver developed by Xu et al. [30]. SATZILLA dominated the SAT competition in 2007, thus demonstrating the potential impact of algorithm selection research. However, results in this line of work have come with few theoretical guarantees.

Gupta and Roughgarden leveraged tools from learning theory to conduct a theoretical analysis of several fundamental problems in application-specific algorithm selection [14]. In their work, they study algorithm families such as greedy heuristic algorithms for problems like knapsack, bucket-based sorting algorithms, variants on gradient descent, and they also study application-specific algorithm selection in the online setting. We bolster this line of work by studying algorithms for problems which are ubiquitous in machine learning and optimization: integer quadratic programming and clustering. In this paper, we develop techniques for analyzing randomized algorithms, whereas the algorithms analyzed in the previous work were deterministic. We also provide the first pseudo-dimension lower bounds in this line of work, which require an involved analysis of the each algorithm family's performance on carefully constructed problem instances. We believe that our lower bounds are of particular interest because for several of the single-parameter algorithm classes we study, we prove $\Omega(\log n)$ lower bounds. This contradicts intuition because one would naturally expect the pseudo-dimension of a function class to be directly proportional to the number of parameters defining that class.

2 Preliminaries and definitions

In this section, we provide the notation and definitions we will need from learning theory. We begin with the definition of *pseudo-dimension*, a natural extension of the well-known *VC dimension* to learning over classes of real-valued functions. We state the definition generally in terms of an arbitrary class of real-valued functions $\mathcal{F}$, which we then describe how to instantiate for a class of

algorithms and an arbitrary cost function. We say that a finite subset $S = \{x_1, \ldots, x_m\}$ of X is *shattered* by $\mathcal{F}$ if there exist real-valued *witnesses* $r_1, \ldots, r_m$ such that for all subsets $T \subseteq S$, there exists a function $f_T \in \mathcal{F}$ such that $f_T(x_i) \leq r_i$ if and only if $i \in T$. The *pseudo-dimension* of $\mathcal{F}$, denoted by $Pdim(\mathcal{F})$, is the cardinality of the largest subset of X shattered by $\mathcal{F}$. By bounding $Pdim(\mathcal{F})$, we may derive strong sample complexity guarantees, as follows.

Theorem 1. *[e.g. [15]] Let $\mathcal{F}$ be a class of functions with domain X and range in $[0, H]$, and suppose $\mathcal{F}$ has pseudo-dimension d. For every distribution $\mathcal{D}$ over X, every $\epsilon > 0$, and every $\delta \in (0, 1]$, if $m \geq c\left(\frac{H}{\epsilon}\right)^2 \left(d \log\left(\frac{H}{\epsilon}\right) + \log\left(\frac{1}{\delta}\right)\right)$ for a suitable constant c (independent of all other parameters), then with probability at least $1-\delta$ over m samples $x_1, \ldots, x_m \sim \mathcal{D}$, $\left|\left(\frac{1}{m}\sum_{i=1}^{m} f(x_i)\right) - \mathbb{E}_{x\sim\mathcal{D}}[f(x)]\right| < \epsilon$ for every $f \in \mathcal{F}$.*

We may similarly derive the pseudo-dimension of a class of algorithms $\mathcal{A}$ together with a cost function $\texttt{cost}(h, x)$ for $h \in \mathcal{A}$. Specifically, we say that a set of input problem instances $\mathcal{S}$ is shattered by $\mathcal{A}$ if we can similarly define $|\mathcal{S}|$ witnesses and $2^{|\mathcal{S}|}$ algorithms in $\mathcal{A}$ which in turn induce all $2^{|\mathcal{S}|}$ binary labelings of $\mathcal{S}$: whether or not $\texttt{cost}(h, x)$ is less than its witness. Theorem 1 then characterizes the sample complexity required to ensure that for all algorithms $h \in \mathcal{A}$, the empirical cost of h on the sample is close to the expected cost of h with respect to the distribution. It is easy to show that if h^* is the algorithm that minimizes the expected cost and $\hat{h}$ is the algorithm that minimizes empirical cost, then with high probability, the expected cost of running $\hat{h}$ is close to optimal. Specifically, with probability at least $1 - \delta$, $\mathbb{E}_{x\sim\mathcal{D}}\left[\texttt{cost}\left(\hat{h}, x\right)\right] - \mathbb{E}_{x\sim\mathcal{D}}\left[\texttt{cost}\left(h^*, x\right)\right] < \epsilon$. Therefore, if a learning algorithm receives as input a sufficiently large set of samples and returns the algorithm which performs best on that sample, we can be guaranteed that this algorithm is close to optimal with respect to the underlying distribution.

3 SDP-based methods for integer quadratic programming

In this section, we study several infinite classes of integer-quadratic programming (IQP) approximation algorithms. These classes consist of SDP rounding algorithms and are a generalization of the seminal Goemans-Williamson (GW) max-cut algorithm [12]. We prove that it is possible to learn the optimal algorithm from a fixed class over a specific application domain, and for many of the classes we study, this learning procedure is computationally efficient.

We focus on integer-quadratic programs of the form $\sum_{i,j\in[n]} a_{ij} x_i x_j$, where the input is a matrix A with nonnegative diagonal entries and the output is an assignment of the binary variables $X = \{x_1, \ldots, x_n\}$ maximizing this sum. Specifically, each variable in X is set to either -1 or 1. This problem is also known as MaxQP [8]. Most algorithms with the best approximation guarantees use an SDP relaxation. The SDP relaxation has the form

$$\text{maximize} \sum_{i,j\in[n]} a_{ij} \langle \vec{u}_i, \vec{u}_j \rangle \tag{1}$$

$$\text{subject to } \vec{u}_i \in S^{n-1}.$$

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

The GW algorithm can be generalized, and sometimes improved upon, by probabilistically assigning each binary variable to 1 or -1. Specifically, in the final rounding step, any rounding function $r : \mathbb{R} \to [-1, 1]$ can be used to specify that a variable x_i is set to 1 with probability $\frac{1}{2} + \frac{1}{2} \cdot r\left(\langle \vec{Z}, \vec{u}_i \rangle\right)$ and -1 with probability $\frac{1}{2} - \frac{1}{2} \cdot r\left(\langle \vec{Z}, \vec{u}_i \rangle\right)$. See Algorithm 1 for the pseudocode of this generalization. This type of algorithm is known as a *Random Projection, Randomized*

Algorithm 1 SDP rounding algorithm with rounding function r

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

1: Solve the SDP (1) for the optimal embedding $U = (\vec{u}_1, \ldots, \vec{u}_n)$.

2: Choose a random vector $\vec{Z} \in \mathbb{R}^n$ according to the n-dimensional Gaussian distribution.

3: Define the fractional assignment $h : X \to [-1, 1]$ such that $h(x_i) = r\left(\langle \vec{Z}, \vec{u}_i \rangle\right)$.

Output: h.

Rounding (RPR^2) algorithm, so named by Feige and Langberg [11]. The randomized assignment h produced by an RPR^2 algorithm is called a *fractional assignment*. Based on the output h, we can derive a proper assignment of the variables $x_1, \ldots, x_n$ where x_i is set to 1 with probability $\frac{1}{2} + \frac{1}{2}h(x_i)$ and -1 with probability $\frac{1}{2} - \frac{1}{2}h(x_i)$.

In this work, we study several rich function classes from which the rounding function r may be chosen. In the following section, we analyze the class of s-linear functions. For the max-cut problem, Feige and Langberg proved that when the maximium cut in the graph is not very large, an approximation ratio above the GW ratio is possible using an s-linear rounding function [11]. For example, they proved that if the optimal cut contains at most a 0.6 fraction of the edges, then the ratio is at least 0.9128. The optimal choice of s depends on the graph, but we give a polynomial time algorithm to learn a nearly optimal value for s in expectation over a distribution of problem instances.

As we describe in Section 1.1, our techniques are not specific to RPR^2 algorithms that first find the optimal embedding of the SDP relaxation. They also apply to any class of RPR^2 algorithms where the first step is to find some fixed type of embedding on the unit sphere and then round the vector solution to a binary solution using one of the classes of rounding functions we study. For example, Johansson et al. quickly find strong approximate solutions to the max-cut problem by first approximating an embedding based on the Lovász theta function and then running the standard GW algorithm to round this embedding into a graph cut [16].

We also analyze rounding functions beyond s-linear rounding functions. In Appendix B, we study the class of $\tilde{\epsilon}$-discretized rounding functions as defined by O'Donnell and Wu [21] and the class of outward rotation algorithms proposed by Zwick [32]. We then extend these results to general classes of "sigmoid-like" rounding functions, which include the classes of s-linear and outward rotation functions.

3.1 s-linear rounding functions

An s-linear rounding function $\phi_s : \mathbb{R} \to [-1, 1]$ is parameterized by a constant $s > 0$, where

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

See Figure 1 for a graph of ϕ_2.

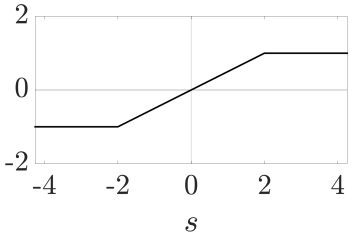

Figure 1: A graph of the 2-linear function ϕ_2.

Recall that our goal is to devise an algorithm L_{slin} that (ϵ, δ)-learns the best s-linear rounding function with respect to a distribution $\mathcal{D}$ over MaxQPs problem instances, given m samples from $\mathcal{D}$. Specifically, let $\texttt{slin}_s^*(A)$ be the expected value of the solution returned by RPR^2 using the rounding function ϕ_s when evaluated on the MaxQP problem instance defined by the matrix A. We instantiate the cost function $\texttt{cost}$ to be $\texttt{cost}(A, s) = -\texttt{slin}_s^*(A)$. Here, we take the negative of $\texttt{slin}_s^*(A)$ since our goal is to find the s parameter that will maximize this value in expectation, and minimizing $\texttt{cost}(A, s)$ now amounts to maximizing $\texttt{slin}_s^*(A)$. In accordance with our goal, we require that L_{slin} returns a value $\hat{s}$ such that with probability at least $1 - \delta$, if s^* maximizes $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}[\texttt{slin}_s^*(A)]$, then $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}[\texttt{slin}_{s^*}^*(A)] - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}[\texttt{slin}_{\hat{s}}^*(A)] < \epsilon$. Naturally, one might expect that the first step would be to bound the pseudo-dimension of the function class $\mathcal{H}_{slin^*} = \{\texttt{slin}_s^* : \mathbb{A} \to [0, 1] \mid s > 0\}$, where $\mathbb{A}$ is the set of all real-valued $n \times n$ matrices with nonnegative diagonal entries. However, we pursue an alternative route that provides a simpler sample complexity and algorithmic analysis.

In particular, we instead analyze the class $\mathcal{H}_{slin} = \{\texttt{slin}_s : \mathbb{A} \times \mathbb{R}^n \to [0, 1] \mid s > 0\}$, where $\texttt{slin}_s\left(A, \vec{Z}\right)$ is the value of the fractional assignment produced by projecting the SDP embedding of A onto $\vec{Z}$ and rounding the directed distances of the resulting projections (multiplied by $||\vec{Z}||$) using the rounding function ϕ_s. Explicitly, $\texttt{slin}_s\left(A, \vec{Z}\right) = \sum_{i,j} a_{ij}\phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right) \cdot \phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right)$. Notice that $\texttt{slin}_s^*(A) = \mathbb{E}_{Z\sim\mathcal{Z}}\left[\texttt{slin}_s\left(A, \vec{Z}\right)\right]$, where $\mathcal{Z}$ denotes the standard n-dimensional Gaussian distribution.[3]

We provide tight bounds on the pseudo-dimension of $\mathcal{H}_{slin}$, rather than $\mathcal{H}_{slin^*}$. Nonetheless, we show that we can use our bound on the pseudo-dimension of $\mathcal{H}_{slin}$ to derive strong generalization guarantees for optimizing over the class of algorithms we ultimately care about, $\mathcal{H}_{slin^*}$.

Specifically, we then show that empirical maximization over $\mathcal{H}_{slin}$ amounts to empirical maximization over $\mathcal{H}_{slin^*}$. In particular, suppose that a learning algorithm L_{slin} takes as input m samples $\left(A^{(i)}, \vec{Z}^{(i)}\right) \sim \mathcal{D}\times\mathcal{Z}$ and finds a value $\hat{s}$ such that $\texttt{slin}_{\hat{s}} \in \mathcal{H}_{slin}$ maximizes $\frac{1}{m}\sum_{i=1}^m \texttt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right)$. We prove that RPR^2 using the $\hat{s}$-linear rounding function is nearly optimal with respect to $\mathcal{D}$ as well. Formally, we prove the following theorem, the proof of which can be found in Appendix A.

Theorem 2. *Let d be the pseudo-dimension of $\mathcal{H}_{slin}$. Suppose that L_{slin} is an algorithm that takes as input m samples $\left(A^{(i)}, \vec{Z}^{(i)}\right) \sim \mathcal{D} \times \mathcal{Z}$, where $m = O\left(\frac{1}{\epsilon^2}\left(d\log\frac{1}{\epsilon} + \log\frac{1}{\delta}\right)\right)$, and returns the parameter $\hat{s}$ which maximizes $\frac{1}{m}\sum_{i=1}^m \texttt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right)$. Then L_{slin} (ϵ, δ)-learns the class of*

[3]There are, in fact, two sources of randomness that contribute to the value of $\texttt{slin}_s^*(A)$. First, there is the draw of the random hyperplane $Z \sim \mathcal{Z}$. Stripping away this layer of randomness, we obtain the function $\texttt{slin}_s\left(A, \vec{Z}\right) = \sum_{i,j} a_{ij}\phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right) \cdot \phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right)$. This sum hides a second layer of randomness, since $\mathbb{E}[x_i] = 1 \cdot \left(\frac{1}{2} + \frac{1}{2} \cdot \phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right)\right) + (-1) \cdot \left(\frac{1}{2} - \frac{1}{2} \cdot \phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right)\right) = \phi_s\left(\langle\vec{u}_i, \vec{Z}\rangle\right)$, which means that $\texttt{slin}_s\left(A, \vec{Z}\right) = \sum_{i,j} a_{ij}\mathbb{E}[x_i]\mathbb{E}[x_j]$.

s-linear rounding functions with respect to the cost function $-\texttt{slin}_s^*$ *and it is computationally efficient.*

Again, we set $\texttt{cost}(A, s) = -\texttt{slin}_s^*(A)$ since minimizing $\texttt{cost}(A, s)$ then amounts to maximizing $\texttt{slin}_s^*(A)$. In the remainder of this section, we will prove that the pseudo-dimension of $\mathcal{H}_{slin}$ is $\Theta(\log n)$ and then we will present a computationally efficient algorithm L_{slin}.

Theorem 3. *Let* $\mathcal{H}_{slin} = \{\texttt{slin}_s : \mathbb{A} \times \mathbb{R}^n \to [0, 1] \mid \ s > 0\}$. *Then* $Pdim(\mathcal{H}_{slin}) = \Theta(\log n)$.

Proof. Theorem 3 follows from Lemma 2 and Lemma 3, wherein we provide matching upper and lower bounds on $\text{Pdim}(\mathcal{H}_{slin})$ of $O(\log n)$ and $\Omega(\log n)$. □

Our lower bound is particularly strong because it holds for a family of positive semidefinite matrices, rather than a more general family of real-valued matrices. In our analysis, we will often fix a tuple $\left(A, \vec{Z}\right)$ and consider $\texttt{slin}_s\left(A, \vec{Z}\right)$ *as a function of* s. We denote this function as $\texttt{slin}_{A,\vec{Z}}(s)$. We begin with a helpful lemma.

Lemma 1. *The function* $\texttt{slin}_{A,\vec{Z}} : R_{>0} \to \mathbb{R}$ *is made up of* $n+1$ *piecewise quadratic components. Moreover, if the border between two components falls at some* $s \in \mathbb{R}_{>0}$*, then it must be that* $s = \left|\langle \vec{u}_i, \vec{Z}\rangle\right|$ *for some* $\vec{u}_i$ *in the optimal SDP embedding of* A.

Proof. First, let $X = \{\vec{u}_1, \ldots, \vec{u}_n\} \subset S^{n-1}$ be the optimal embedding of A. Then we may write $\texttt{slin}_{A,\vec{Z}}(s) = \sum_{i,j} a_{ij}\phi_s\left(\langle \vec{u}_i, \vec{Z}\rangle\right) \cdot \phi_s\left(\langle \vec{u}_i, \vec{Z}\rangle\right)$. For any $\vec{u}_i \in X$, the specific form of $\phi_s\left(\langle \vec{Z}, \vec{u}_i\rangle\right)$ depends solely on whether $\langle \vec{Z}, \vec{u}_i\rangle < -s$, $-s \leq \langle \vec{Z}, \vec{u}_i\rangle \leq s$, or $s < \langle \vec{Z}, \vec{u}_i\rangle$. Of course, if $\langle \vec{Z}, \vec{u}_i\rangle \geq 0$ or $\langle \vec{Z}, \vec{u}_i\rangle < 0$, we can disregard the possibility that $\langle \vec{Z}, \vec{u}_i\rangle < -s$ or $\langle \vec{Z}, \vec{u}_i\rangle > s$, respectively. Then so long as $\left|\langle \vec{Z}, \vec{u}_i\rangle\right| > s$, we have that $\phi_s\left(\langle \vec{Z}, \vec{u}_i\rangle\right) = \pm 1$, where the sign depends on the sign of $\langle \vec{Z}, \vec{u}_i\rangle$. Further, when s grows to the point where $s > \left|\langle \vec{Z}, \vec{u}_j\rangle\right|$, we have that $\phi_s\left(\langle \vec{Z}, \vec{u}_i\rangle\right) = \langle \vec{Z}, \vec{u}_j\rangle / s$. Therefore, if we order the set of real values $\left\{\left|\langle \vec{Z}, \vec{u}_1\rangle\right|, \ldots, \left|\langle \vec{Z}, \vec{u}_n\rangle\right|\right\}$, then so long as s falls between two consecutive elements of this ordering, the form of $\texttt{slin}_{A,\vec{Z}}(s)$ is fixed. In particular, each summand is either a constant, a constant multiplied by $\frac{1}{s}$, or a constant multiplied by $\frac{1}{s^2}$, perhaps accompanied by an additive constant term. This means that we may partition the positive real line into $n+1$ intervals where the form of $\texttt{slin}_{\left(A,\vec{Z}\right)}(s)$ is a fixed quadratic function, as claimed. □

Lemma 2. $Pdim(\mathcal{H}_{slin}) = O(\log n)$.

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

Now, $\mathcal{S}$ consists of m tuples $\left(A^{(i)}, \vec{Z}^{(i)}\right)$, each of which corresponds to its own partition of the positive real line. If we merge these partitions (as shown in Figure 2), simple algebra shows that

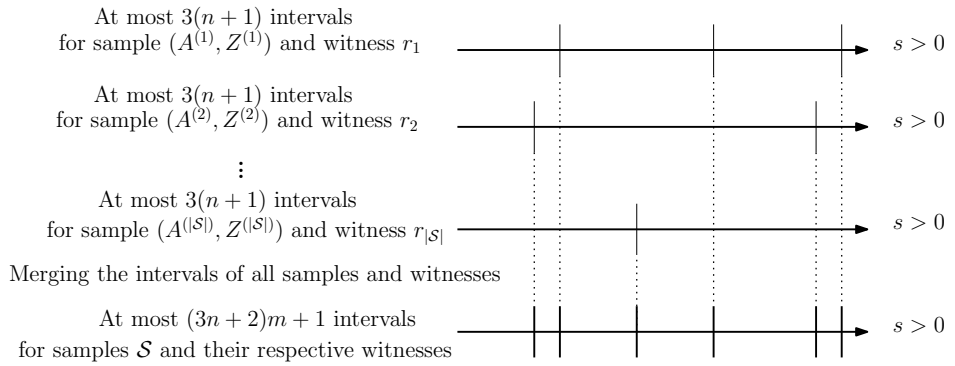

Figure 2: Partitioning $s > 0$ into intervals given a set $\mathcal{S}$ of m tuples $\left(A^{(i)}, \vec{Z}(i)\right)$ and witnesses r_i such that within each interval for each i, $\texttt{slin}_{(A^{(i)}, Z^{(i)})}(s)$ is always greater than r_i or lesser than r_i.

we are left with at most $(3n+2)m+1$ intervals such that for all $i \in [m]$, $\texttt{slin}_{A^{(i)}, \vec{Z}^{(i)}}(s)$ is always either less than its witness r_i or greater than r_i as s varies over one fixed interval. In other words, in one interval, the binary labeling of $\mathcal{S}$, defined by whether each sample is less than or greater than its witness, is fixed. This means that if $\mathcal{S}$ is shatterable, the 2^m values of s which induce all 2^m binary labelings of $\mathcal{S}$ must come from distinct intervals. Therefore $2^m \leq (3n+2)m+1$, so $m = O(\log n)$. □

Lemma 3. *$Pdim(\mathcal{H}_{slin}) = \Omega(\log n)$.*

Proof sketch. In order to prove that the pseudo dimension of $\mathcal{H}_{slin}$ is at least $c \log n$ for some c, we present a set $\mathcal{S} = \left\{\left(G^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(G^{(m)}, \vec{Z}^{(m)}\right)\right\}$ of $m = c \log n$ graphs and projection vectors that can be shattered by $\mathcal{H}_{slin}$. In other words, there exist m witnesses $r_1, \ldots, r_m$ and $2^m = n^c$ s values $H = \{s_1, \ldots, s_{n^c}\}$ such that for all $T \subseteq [m]$, there exists $s_T \in H$ such that if $j \in T$, then $\texttt{slin}_{S_T}\left(G^{(j)}, \vec{Z}^{(j)}\right) > r_j$ and if $j \notin T$, then $\texttt{slin}_{S_T}\left(G^{(j)}, \vec{Z}^{(j)}\right) \leq r_j$.

To build $\mathcal{S}$, we use the same graph G for all $G^{(j)}$ and we vary $\vec{Z}^{(j)}$. We set G to be the graph composed of $\lfloor n/4 \rfloor$ disjoint copies of K_4. Via a careful choice of the vectors $\vec{Z}^{(j)}$ and witnesses r_j, we pick out 2^m critical values of s, which we call C, such that $\texttt{slin}_{G, \vec{Z}^{(1)}}(s)$ switches from above its witness to below its witness for every other value of critical values in C. Meanwhile, $\texttt{slin}_{G, \vec{Z}^{(2)}}(s)$ switches from below its witness to above its witness half as often as $\texttt{slin}_{G, \vec{Z}^{(1)}}(s)$. Similarly, $\texttt{slin}_{G, \vec{Z}^{(3)}}(s)$ switches from below its witness to above its witness half as often as $\texttt{slin}_{G, \vec{Z}^{(2)}}(s)$, and so on. This is illustrated by Figure 3. Therefore, we achieve every binary labeling of $\mathcal{S}$ (whether $\texttt{slin}_s\left(G, \vec{Z}^{(i)}\right)$ is less than its witness or greater than its witness) using the functions $\{\texttt{slin}_s \mid s \in C\}$, so $\mathcal{S}$ is shattered by $\mathcal{H}_{slin}$. □

We now claim that Algorithm 2 learns the best s-linear function over a sample in polynomial time.

Lemma 4. *Algorithm 2 produces the value $\hat{s}$ which maximizes $\frac{1}{m}\sum_{i=1}^{m} \texttt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right)$ over the input sample $\mathcal{S}$. Algorithm 2 has running time polynomial in m and n.*

Proof of Lemma 4. First, define $h_{\mathcal{S}, \vec{Z}}(s) = \frac{1}{m}\sum_{i=1}^{m} \texttt{slin}_{A^{(i)}, \vec{Z}^{(i)}}(s)$, which we claim that Algorithm 2 maximizes. In Lemma 1, we proved that each function $\texttt{slin}_{A^{(i)}, \vec{Z}^{(i)}}(s)$ is made up of

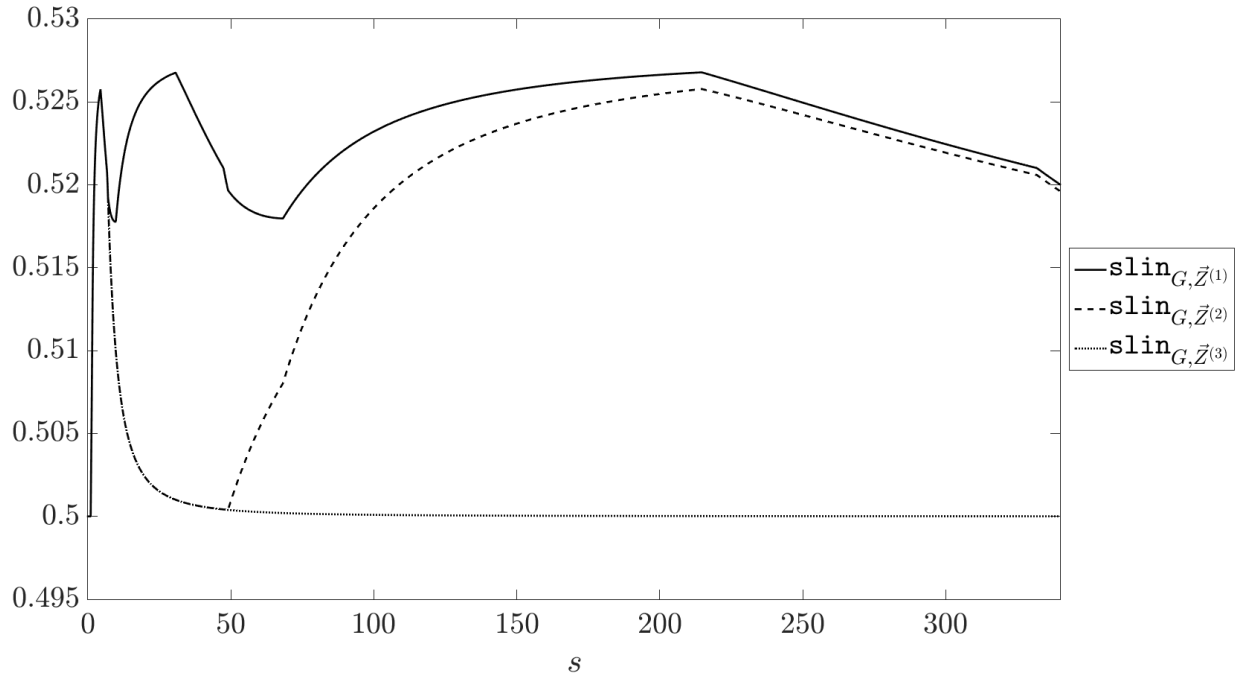

Figure 3: Graph of $\mathtt{slin}_{G,\vec{Z}^{(1)}}$, $\mathtt{slin}_{G,\vec{Z}^{(2)}}$, and $\mathtt{slin}_{G,\vec{Z}^{(3)}}$ when $n = 12$.

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

Input: Sample $\mathcal{S} = \left\{\left(A^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}\right)\right\}$

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

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

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

4: Let $\hat{s}$ be the value in $\left\{\hat{s}_1, \ldots, \hat{s}_{|T|-1}\right\}$ that maximizes $\frac{1}{m}\sum_{i=1}^{m} \mathtt{slin}_{A^{(i)},\vec{Z}^{(i)}}(s)$.

Output: $\hat{s}$

at most $n+1$ piecewise quadratic components. Therefore, $h_{\mathcal{S},\vec{Z}}(s)$ is made up of at most $mn+1$ piecewise quadratic components. Moreover, by Lemma 1, if the border between two components falls at some $s \in \mathbb{R}_{>0}$, then it must be that $\left|\langle \vec{Z}^{(i)}, \vec{x}_j^{(i)} \rangle\right| = s$ for some $\vec{x}_j^{(i)}$ in the optimal max-cut SDP embedding of $A^{(i)}$. These are the thresholds which are computed in Step 2 of Algorithm 2. Therefore, as we increase s starting at 0, s will be a fixed quadratic function between the thresholds, so it is simple to find the optimal value of s between any pair of consecutive thresholds (Step 3), and then the value maximizing $h_{\mathcal{S},\vec{Z}}(s) = \frac{1}{m}\sum_{i=1}^{m} \texttt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right)$ (Step 4), which is the global optimum. □

Lemma 4 together with Theorem 2 and Theorem 3 imply our main result.

Theorem 4. *Given an input sample of size $m = O\left(\frac{1}{\epsilon^2}\left(\log n \log\frac{1}{\epsilon} + \frac{1}{\delta}\right)\right)$ drawn from $(\mathcal{D}\times\mathcal{Z})^m$, Algorithm 2 (ϵ,δ)-learns the class of s-linear rounding functions with respect to the cost function $-\texttt{slin}_s^*$ and it is computationally efficient.*

Proof. Let $\mathcal{S} = \left\{\left(A^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}\right)\right\}$ be a sample of size m. In Lemma 4, we prove that Algorithm 2 on input $\mathcal{S}$ returns the value $\hat{s}$ which maximizes $\frac{1}{m}\sum_{i=1}^{m} \texttt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right)$ in polynomial time. Theorem 2 together with Theorem 3 imply that so long as $m = O\left(\frac{1}{\epsilon^2}\left(\log n \log\frac{1}{\epsilon} + \frac{1}{\delta}\right)\right)$, then Algorithm 2 (ϵ,δ)-learns the best s-linear function with respect to $\mathcal{D}$. □

4 Agglomerative algorithms with dynamic programming

In this section, we start with an overview of agglomerative algorithms with dynamic programming, which include several widely-studied clustering algorithms, and then we define several parameterized classes of such algorithms. As in the previous section, we prove it is possible to learn the optimal algorithm from a fixed class for a specific application, and for many of the classes we analyze, this procedure is computationally efficient.

We focus on agglomerative algorithms with dynamic programming for *clustering* problems. A clustering instance $\mathcal{V} = (V, d)$ consists of a set V of n points and a distance metric $d : V \times V \to \mathbb{R}_{\geq 0}$ specifying all pairwise distances between these points. At a high level, the goal of clustering is to partition the points into groups such that distances within each group are minimized and distances between each group are maximized. A classic way to accomplish this task is to use an objective function Φ. For example, Φ can be set to k-means, k-median, k-center, or the distance to the ground truth clustering.[4] We define the following rich class of clustering objectives. For $p \in [1,\infty)\cup\{\infty\}$, set $\Phi^{(p)}(\mathcal{C}, \mathbf{c}) = \sum_{i=1}^{k}(\sum_{q\in C_i} d(q, c_i)^p)^{1/p}$. The k-means, k-median, and k-center objective functions are $\Phi^{(2)}$, $\Phi^{(1)}$, and $\Phi^{(\infty)}$, respectively.

Next, we define agglomerative clustering algorithms with dynamic programming, which are prevalent in practice and enjoy strong theoretical guarantees in a variety of settings [3, 4, 5, 13]. For example, the *complete-linkage* algorithm yields a constant-factor approximation to the k-center objective for any metric that is induced by a norm, so long as the dimension is constant [13]. The

[4] If Φ is the distance to ground truth clustering, then Φ cannot be directly measured when the clustering algorithm is used on new data. However, we assume that the learning algorithm has access to training data which consists of clustering instances labeled by the ground truth clustering. The learning algorithm uses this data to optimize the parameters defining the clustering algorithm family. With high probability, on a new input drawn from the same distribution as the training data, the clustering algorithm will return a clustering that is close to the unknown ground truth clustering. We discuss this in more detail in Section 4.3.

single-linkage algorithm with dynamic programming returns the optimal solution for any center-based objective when the optimal solution is resilient to small perturbations on the input distances [3]. The *average-linkage* algorithm also returns the optimal solution when the solution is resilient to perturbations, for the *min-sum* objective [5].

An agglomerative clustering algorithm with dynamic programming is defined by two functions: a merge function and a pruning function. A merge function $\xi(A, B) \to \mathbb{R}_{\geq 0}$ defines the distance between two sets of points $A, B \subseteq V$. Based on ξ, the algorithm iteratively merges points in the input instance $\mathcal{V} = (V, d)$ to build a *cluster tree* $\mathcal{T}$ whose nodes are sets of points in V such that all leaf nodes are singletons and the root node is V itself. The children of any node T in this tree correspond to the two sets of points that were merged to form T during the sequence of merges.

A pruning function Ψ takes as input a *pruning* of any subtree of $\mathcal{T}$ and returns a score $\mathbb{R}_{\geq 0}$ for that pruning. More formally, a k'-pruning (for any $k' \leq n$) for any node T in $\mathcal{T}$, is a partition of T into a set of k' clusters $\mathcal{C} = \{C_1, \ldots, C_{k'}\}$ and k' centers $\mathbf{c} = \{c_1, \ldots c_{k'}\}$ such that each cluster in $\mathcal{C}$ is a descendant node of T in $\mathcal{T}$. Thus, $\Psi(\mathcal{C}, \mathbf{c}) \in \mathbb{R}_{\geq 0}$ is the score of the pruning $(\mathcal{C}, \mathbf{c})$. The algorithm returns the k-pruning of the tree $\mathcal{T}$ that is optimal according to Ψ.

Algorithm 3 details how the merge function and pruning function work together to form an agglomerative clustering algorithm with dynamic programming. In the dynamic programming step, to find the 1-pruning of any node T, we only need to find the best center $c \in T$. When $k' > 1$, we recursively find the best k'-pruning of T by considering different combinations of the best i'-pruning of the left child T_L and the best $k' - i'$-pruning of the right child T_R for $i' \in \{1, \ldots, k-1\}$ and choosing the best combination.

Algorithm 3 Agglomerative algorithm with dynamic programming

Input: Clustering instance $\mathcal{V} = (V, d)$, merge function ξ, pruning function Ψ.

1: **Agglomerative merge step to build a cluster tree $\mathcal{T}$ according to ξ:**
- Start with n singleton sets $\{v\}$ for each $v \in V$.
- Iteratively merge the two sets A and B which minimize $\xi(A, B)$ until a single set remains.
- Let $\mathcal{T}$ be the cluster tree corresponding to the sequence of merges.

2: **Dynamic programming to find the k-pruning of $\mathcal{T}$ minimizing Ψ :**
- For each node T, find the best k'-pruning of the subtree rooted at T in $\mathcal{T}$, denoted by $\left(\mathcal{C}_{T,k'}, \mathbf{c}_{T,k'}\right)$ according to following dynamic programming recursion:

$$\Psi\left(\mathcal{C}_{T,k'}, \mathbf{c}_{T,k'}\right) = \begin{cases} \min_{c \in T} \Psi\left(\{T\}, c\right) & \text{if } k' = 1, \\ \min_{i' \in [k'-1]} \Psi\left(\mathcal{C}_{T_L,i'} \cup \mathcal{C}_{T_R,k'-i'}, \mathbf{c}_{T_L,i'} \cup \mathbf{c}_{T_R,k'-i'}\right) & \text{otherwise.} \end{cases}$$

where T_L and T_R denote the left and right children of T, respectively.

Output: The best k-pruning of the root node T_{root} of $\mathcal{T}$.

Pictorially, Figure 4 depicts an array of available choices when designing an agglomerative clustering algorithm with dynamic programming. Each path in the chart corresponds to an alternative choice of a merge function ξ and pruning function Ψ. The algorithm designer's goal is to determine the path that is optimal for her specific application domain. Some examples of merge functions ξ used in common agglomerative algorithms include $\min_{a \in A, b \in B} d(a, b)$ (single linkage), $\frac{1}{|A| \cdot |B|} \sum_{a \in A, b \in B} d(a, b)$ (average linkage) and $\max_{a \in A, b \in B} d(a, b)$ (complete linkage). Commonly used pruning functions Ψ include the k-means, k-median, and k-center objectives.

In Sections 4.1 and 4.2, we analyze several classes of algorithms where the merge function comes from an infinite family of functions while the pruning function is an arbitrary, fixed function. Finally, in Section 4.3, we expand our analysis to include algorithms defined over an infinite family

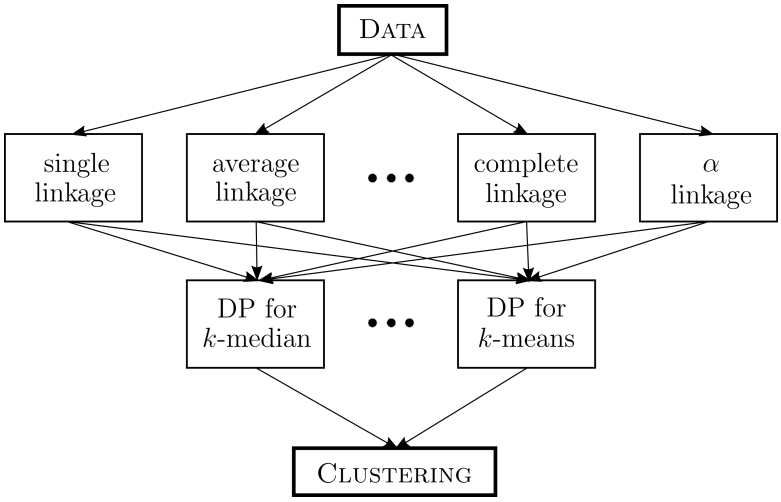

Figure 4: A schematic for a class of agglomerative clustering algorithms with dynamic programming.

of pruning functions in conjunction with any family of merge functions. We note that our results hold even when there is a preprocessing step that precedes the agglomerative merge step (Step 1 in Algorithm 3) such as in [5] which merges some points to begin with. This is because such a pre-processing merge step is independent of ξ or Ψ, so we may learn the optimal choice of ξ and Ψ following the preprocessing stage.

4.1 General classes of linkage-based merge functions

We now define two infinite families of merge functions for which we will provide tight pseudo-dimension bounds and computationally efficient learning algorithms, for any fixed but arbitrary pruning function. Both of the classes we propose are defined by merge functions $\xi(A, B)$ that depend only on the maximum and minimum of all pairwise distances between A and B. While both classes are parametrized by a single value α, they differ in that one is defined by a linear combination of these two distances while the other is defined by a non-linear combination. We use $\mathcal{A}_1$ and $\mathcal{A}_2$ to denote the classes of merge functions

$$\mathcal{A}_1 = \left\{ \alpha \min_{u \in A, v \in B} d(u,v) + (1-\alpha) \max_{u \in A, v \in B} d(u,v) \;\middle|\; \alpha \in [0,1] \right\},$$
$$\mathcal{A}_2 = \left\{ \left(\min_{u \in A, v \in B} (d(u,v))^\alpha + \max_{u \in A, v \in B} (d(u,v))^\alpha \right)^{1/\alpha} \;\middle|\; \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\}.$$

Given a set of pruning functions $\mathcal{F}$, let $\mathcal{A}_i \times \mathcal{F}$ denote the class of algorithms defined by the set of merge function and pruning function pairs $(\xi, \Psi) \in \mathcal{A}_i \times \mathcal{F}$. As noted, in Sections 4.1 and 4.2, we analyze classes of algorithms where the set of pruning functions $\mathcal{F}$ is a singleton set consisting of a fixed but arbitrary function Ψ. Given a pruning function $\Psi \in \mathcal{F}$ and a parameter α, let $(\mathcal{A}_i(\alpha), \Psi)$ be the algorithm which first builds a cluster tree according to merge function in $\mathcal{A}_i$ defined by α and then prunes the tree according to Ψ. For any pruning function Ψ, both $\mathcal{A}_1 \times \{\Psi\}$ and $\mathcal{A}_2 \times \{\Psi\}$ define a spectrum of agglomerative clustering algorithms with dynamic programming, ranging from complete-linkage with dynamic programming (i.e., $(\mathcal{A}_1(0), \Psi)$ and $(\mathcal{A}_2(\infty), \Psi)$) to single-linkage with dynamic programming (i.e., $(\mathcal{A}_1(1), \Psi)$ and $(\mathcal{A}_2(-\infty), \Psi)$). To reduce notation in Sections 4.1 and 4.2 where $\mathcal{F} = \{\Psi\}$, we often refer to the algorithm $(\mathcal{A}_i(\alpha), \Psi)$ as $\mathcal{A}_i(\alpha)$ and the

class of algorithms $\mathcal{A}_i \times \{\Psi\}$ as $\mathcal{A}_i$ when Ψ is clear from context. In particular, if the cost function is $\Phi^{(p)}$, then we set Ψ to minimize the $\Phi^{(p)}$ objective. In Section 4.3, we expand our analysis to richer classes of algorithms $\mathcal{A}_i \times \mathcal{F}$ where $\mathcal{F}$ consists of infinitely many pruning functions.

Recall that for a given class of merge functions and a `cost` function (a generic clustering objective Φ), our goal is to learn a near-optimal value of α in expectation over an unknown distribution of clustering instances. Naturally, one might wonder if there is some α which is optimal across all instances, which would preclude the need for a learning algorithm. We prove this is not the case in the following theorem, proven in Appendix C. To formally describe this result, we set up notation similar to Section 3.1. Let $\mathbb{V}$ denote the set of all clustering instances over at most n points. With a slight abuse of notation, we will use $\Phi_{(\mathcal{A}_i(\alpha),\Psi)}(\mathcal{V})$ to denote the abstract cost of the clustering produced by $(\mathcal{A}_i(\alpha), \Psi)$ on the instance $\mathcal{V}$.

Theorem 5. *For $b \in \{1, 2\}$ and a permissible value of α for $\mathcal{A}_b$, there exists a distribution $\mathcal{D}$ over clustering instances $\mathbb{V}$ such that $\mathbb{E}_{\mathcal{V}\sim\mathcal{D}}\left[\Phi^{(p)}_{\mathcal{A}_b(\alpha)}(\mathcal{V})\right] < \mathbb{E}_{\mathcal{V}\sim\mathcal{D}}\left[\Phi^{(p)}_{\mathcal{A}_b(\alpha')}(\mathcal{V})\right]$ for all permissible values of $\alpha' \neq \alpha$ for $\mathcal{A}_b$.*

Now for an arbitrary objective function Φ and arbitrary pruning function Ψ, we bound the pseudo-dimension of both classes

$$\mathcal{H}_{\mathcal{A}_1\times\{\Psi\},\Phi} = \left\{ \Phi_{(\mathcal{A}_1(\alpha),\Psi)} : \mathbb{V} \to \mathbb{R}_{\geq 0} \;\middle|\; \alpha \in [0,1] \right\} \text{ and}$$
$$\mathcal{H}_{\mathcal{A}_2\times\{\Psi\},\Phi} = \left\{ \Phi_{(\mathcal{A}_2(\alpha),\Psi)} : \mathbb{V} \to \mathbb{R}_{\geq 0} \;\middle|\; \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\}.$$

We drop the subscript Φ from $\mathcal{H}_{\mathcal{A}_1\times\{\Psi\},\Phi}$ and $\mathcal{H}_{\mathcal{A}_2\times\{\Psi\},\Phi}$ when the objective function is clear from the context.

Theorem 6. *For all objective functions $\Phi^{(p)}$, $Pdim\left(\mathcal{H}_{\mathcal{A}_1,\Phi^{(p)}}\right) = \Theta(\log n)$ and $Pdim\left(\mathcal{H}_{\mathcal{A}_2,\Phi^{(p)}}\right) = \Theta(\log n)$. For all other objective functions Φ and all pruning functions Ψ, $Pdim\left(\mathcal{H}_{\mathcal{A}_1\times\{\Psi\},\Phi}\right) = O(\log n)$ and $Pdim\left(\mathcal{H}_{\mathcal{A}_2\times\{\Psi\},\Phi}\right) = O(\log n)$.*

This theorem follows from Lemma 5, Lemma 16, and Lemma 6.

Lemma 5. *For any objective function Φ and any pruning function Ψ, $Pdim\left(\mathcal{H}_{\mathcal{A}_1\times\{\Psi\},\Phi}\right) = O(\log n)$.*

Proof. Suppose $\mathcal{S} = \left\{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\right\}$ is a set of clustering instances that can be shattered by $\mathcal{H}_{\mathcal{A}_1}$ using the witnesses $r_1, \ldots, r_m$. We must show that $m = O(\log n)$. For each value of $\alpha \in [0, 1]$, the algorithm $\mathcal{A}_1(\alpha)$ induces a binary labeling on each $\mathcal{V}^{(i)}$, based on whether or not $\Phi_{\mathcal{A}_1(\alpha)}\left(\mathcal{V}^{(i)}\right) \leq r_i$.

For a fixed $\mathcal{V} = (V, d)$, we use the notation $\Phi_{\mathcal{A}_1,\mathcal{V}}(\alpha)$ to analyze how $\Phi_{\mathcal{A}_1(\alpha)}(\mathcal{V})$ changes as a function of α. We first claim that $\Phi_{\mathcal{A}_1,\mathcal{V}}(\alpha)$ is a piecewise constant function with $O(n^8)$ discontinuities. To see why this is, first note that for $\alpha \neq \alpha'$, the clustering returned by $\mathcal{A}_1(\alpha)$ and the associated cost are both identical to that of $\mathcal{A}_1(\alpha')$ if both the algorithms construct the same merge tree.

Now, as we increase α from 0 to 1 and observe the run of the algorithm for each α, at what values of α do we expect $\mathcal{A}_1(\alpha)$ to produce different merge trees? To answer this, suppose that at some point in the run of algorithm $\mathcal{A}_1(\alpha)$, there are two pairs of subsets of V, (A, B) and (X, Y), that could potentially merge. There exist eight points $p, p' \in A$, $q, q' \in B$, $x, x' \in X$, and $y, y' \in Y$ such that the decision of which pair to merge depends on whether $\alpha d(p, q) + (1 - \alpha)d(p', q')$ or $\alpha d(x, y) + (1 - \alpha)d(x', y')$ is larger. Clearly, there is at most one value of α for which these expressions are equal, unless both expressions are zero for all α. Assuming that ties are broken arbitrarily but consistently, this implies that there is at most one $\alpha \in [0, 1]$ such that the choice of whether to merge (A, B) before (X, Y) is identical for all $\alpha < \alpha'$, and similarly identical for $\alpha \geq \alpha'$.

Since each merge decision is defined by eight points, iterating over all pairs (A, B) and (X, Y) it follows that we can identify all $O(n^8)$ unique 8-tuples of points which correspond to a value of α at which some decision flips. This means we can divide $[0, 1]$ into $O(n^8)$ intervals over each of which the merge tree, and therefore the output of $\Phi_{\mathcal{A}_1,\mathcal{V}}(\alpha)$, is fixed. Every sample $\mathcal{V}^{(i)}$ partitions $[0, 1]$ into $O(n^8)$ intervals in this way. Merging all m partitions, we can divide $[0, 1]$ into $O(mn^8)$ intervals over each of which $\Phi_{\mathcal{A}_1,\mathcal{V}^{(i)}}(\alpha)$, and therefore the labeling induced by the witnesses, is fixed for all $i \in [m]$ (similar to Figure 2). This means that $\mathcal{H}_{\mathcal{A}_1}$ can achieve only $O(mn^8)$ binary labelings, which is at least 2^m since $\mathcal{S}$ is shatterable, so $m = O(\log n)$. □

A similar proof technique allows us to show that the pseudo-dimension of $\mathcal{A}_2$ is $O(\log n)$. We defer this proof to the appendix (Lemma 16). Now, we provide a sketch of the lower bounds, with the full details in the appendix.

Lemma 6. *For any objective function $\Phi^{(p)}$, $Pdim\left(\mathcal{H}_{\mathcal{A}_1,\Phi^{(p)}}\right) = \Omega(\log n)$ and $Pdim\left(\mathcal{H}_{\mathcal{A}_2,\Phi^{(p)}}\right) = \Omega(\log n)$.*

Proof sketch. We present a general proof outline that applies to both classes. Let $b \in \{1, 2\}$. We construct a set $S = \left\{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\right\}$ of $m = \log n - 3$ clustering instances that can be shattered by $\mathcal{A}_b$. There are $2^m = n/8$ possible labelings for this set, so we need to show there are $n/8$ choices of α such that each of these labelings is achievable by some $\mathcal{A}_b(\alpha)$ for some α. The crux of the proof lies in showing that given a sequence $\alpha_0 < \alpha_1 < \cdots < \alpha_{n'} < \alpha_{n'+1}$ (where $n' = \Omega(n)$), it is possible to design an instance $\mathcal{V} = (V, d)$ over n points and choose a witness r such that $\Phi_{\mathcal{A}_b(\alpha)}(\mathcal{V})$ alternates $n'/2$ times above and below r as α traverses the sequence of intervals (α_i, α_{i+1}).

Here is a high level description of our construction. There will be two "main" points, a and a' in V. The rest of the points are defined in groups of 6: $(x_i, y_i, z_i, x'_i, y'_i, z'_i)$, for $1 \le i \le (n-2)/6$. We will define the distances between all points such that initially for all $\mathcal{A}_b(\alpha)$, x_i merges to y_i to form the set A_i, and x'_i merges to y'_i to form the set A'_i. As for (z_i, z'_i), depending on whether $\alpha < \alpha_i$ or not, $\mathcal{A}_b(\alpha)$ merges the points z_i and z'_i with the sets A_i and A'_i respectively or vice versa. This means that there are $(n-2)/6$ values of α such that $\mathcal{A}_b(\alpha)$ has a unique behavior in the merge step. Finally, for all α, sets A_i merge to $\{a\}$, and sets A'_i merge to $\{a'\}$. Let $A = \{a\} \cup \bigcup_i A_i$ and $A' = \{a'\} \cup \bigcup_i A'_i$. There will be $(n-2)/6$ intervals (α_i, α_{i+1}) for which $\mathcal{A}_b(\alpha)$ returns a unique partition $\{A, A'\}$. By carefully setting the distances, we cause the cost $\Phi(\{A, A'\})$ to oscillate above and below a specified value r along these intervals. □

The upper bound on the pseudo-dimension implies a computationally efficient learning algorithm. First, from Theorem 1, we know that $m = \tilde{O}\left(\left(\frac{H}{\epsilon}\right)^2 (\log n \log(H/\epsilon) + \log(1/\delta))\right)$ samples are sufficient to (ϵ, δ)-learn the optimal algorithm in $\mathcal{A}_b$ for $b \in \{1, 2\}$. Next, as we described in the proofs of Lemmas 5 and 16, the range of feasible values of α can be partitioned into $O(mn^8)$ intervals, such that the output of $\mathcal{A}_b(\alpha)$ is fixed over the entire set of samples on a given interval. Moreover, these intervals are easy to compute. Therefore, a learning algorithm can iterate over the set of intervals, and for each interval I, choose an arbitrary $\alpha \in I$ and compute the average cost of $\mathcal{A}_b(\alpha)$ evaluated on the samples. The algorithm then outputs the α that minimizes the average cost.

Theorem 7. *Let Φ be a clustering objective and let Ψ be a pruning function computable in polynomial time. Given an input sample of size $m = O\left(\left(\frac{H}{\epsilon}\right)^2 \left(\log n \log \frac{H}{\epsilon} + \log \frac{1}{\delta}\right)\right)$ and a value $b \in \{1, 2\}$, Algorithm 4 (ϵ, δ)-learns the class $\mathcal{A}_b \times \{\Psi\}$ with respect to the cost function Φ and it is computationally efficient.*

Algorithm 4 An algorithm for finding an empirical cost minimizing algorithm in $\mathcal{A}_1$ and $\mathcal{A}_2$

Input: Sample $\mathcal{S} = \left\{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\right\}$, algorithm class $b \in \{1, 2\}$.

1: Let $T = \emptyset$. For each sample $\mathcal{V}^{(i)} = \left(V^{(i)}, d^{(i)}\right) \in \mathcal{S}$, and for each ordered set of 8 points $\{v_1, \ldots, v_8\} \subseteq V^{(i)}$, solve for α (if a solution exists) in the following equations and add the solutions to T:

$$\begin{array}{lrcl} \text{If } b = 1: & \alpha d(v_1, v_2) + (1-\alpha) d(v_3, v_4) & = & \alpha d(v_5, v_6) + (1-\alpha) d(v_7, v_8). \\ \text{If } b = 2: & d(v_1, v_2)^\alpha + d(v_3, v_4)^\alpha & = & d(v_5, v_6)^\alpha + d(v_7, v_8)^\alpha. \end{array}$$

2: Order the elements of set $T \cup \{-\infty, +\infty\}$ as $\alpha_1 < \ldots < \alpha_{|T|}$. For each $0 \leq i \leq |T|$, pick an arbitrary α in the interval (α_i, α_{i+1}) and run $\mathcal{A}_b(\alpha)$ on all clustering instances in $\mathcal{S}$ to compute $\sum_{\mathcal{V} \in \mathcal{S}} \Phi_{\mathcal{A}_b(\alpha)}(\mathcal{V})$. Let $\hat{\alpha}$ be the value which minimizes $\sum_{\mathcal{V} \in \mathcal{S}} \Phi_{\mathcal{A}_b(\alpha)}(\mathcal{V})$.

Output: $\hat{\alpha}$

Proof. If Algorithm 4 outputs the value of α that minimizes the average cost of clustering for a set of m samples, then it follows from Theorem 1, Lemma 5 and Lemma 16 that

$$m = O\left(\left(\frac{H}{\epsilon}\right)^2 \left(\log n \log \frac{H}{\epsilon} + \log \frac{1}{\delta}\right)\right)$$

samples are sufficient for Algorithm 4 to (ϵ, δ)-learn the optimal algorithm in $\mathcal{A}_i$ for $i \in \{1, 2\}$.

To prove that Algorithm 4 indeed finds the empirically best α, recall from the pseudo-dimension analyses that the cost as a function of α for any instance is a piecewise constant function with $O(n^8)$ discontinuities. In Step 1 of Algorithm 4, we solve for the values of α at which the discontinuities occur and add them to the set T. T therefore partitions α's range into $O(mn^8)$ subintervals. Within each of these intervals, $\sum_{\mathcal{V} \in \mathcal{S}} \Phi_{\mathcal{A}_b(\alpha)}(\mathcal{V})$ is a constant function. Therefore, we pick any arbitrary α within each interval to evaluate the empirical cost over all samples, and find the empirically best α.

Algorithm 4 is computationally efficient because there are only $O(mn^8)$ many subintervals and the runtime of $\mathcal{A}_b(\alpha)$ on a given instance is polytime. □

4.2 A richer class of linkage-based merge functions

In this section, we will analyze the following linkage criterion, which is more algebraically complex than the criteria analyzed in Section 4.1.

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

Intuitively, compared to $\mathcal{A}_1$ or $\mathcal{A}_2$, this class is defined by a more natural and richer notion of the similarity between two clusters because it considers the distances between all pairs of points in each set, not just the two extreme distances. Consequently, it includes all three common linkage-based algorithms described earlier; setting $\alpha = \infty, 1$, or $-\infty$ correspond to complete-, average-, and single-linkage, respectively. As before, for an arbitrary objective function Φ let $\mathcal{H}_{\mathcal{A}_3, \Phi} = \left\{ \Phi_{\mathcal{A}_3(\alpha)} : \mathbb{V} \to \mathbb{R}_{\geq 0} \,\middle|\, \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\}$.

Theorem 8. *For objective functions $\Phi^{(p)}$, $Pdim\left(\mathcal{H}_{\mathcal{A}_3,\Phi^{(p)}}\right) = \Theta(n)$. For all other objective functions Φ and all pruning functions Ψ, $Pdim\left(\mathcal{H}_{\mathcal{A}_3\times\{\Psi\},\Phi}\right) = O(n)$.*

The proof follows from Lemma 7 and Lemma 8.

Lemma 7. *For all objective functions Φ and all pruning functions Ψ, $Pdim\left(\mathcal{H}_{\mathcal{A}_3\times\{\Psi\},\Phi}\right) = O(n)$.*

Proof. Recall the proofs of Lemma 5 and Lemma 16. We are interested in studying how the merge trees constructed by $\mathcal{A}_3(\alpha)$ changes over m instances as we increase α over $\mathbb{R}$. To do this, as in the proofs of Lemma 5 and Lemma 16, we fix an instance and consider two pairs of sets A, B and X, Y that could be potentially merged. Now, the decision to merge one pair before the other is determined by the sign of the expression $\frac{1}{|A||B|}\sum_{p\in A,q\in B}(d(p,q))^\alpha - \frac{1}{|X||Y|}\sum_{x\in X,y\in Y}(d(x,y))^\alpha$. First note that this expression has $O(n^2)$ terms, and by a consequence of Rolle's Theorem from e.g. [25] (which we state in Appendix C as Theorem 19), it has $O(n^2)$ roots. Therefore, as we iterate over the $O\left((3^n)^2\right)$ possible pairs (A, B) and (X, Y), we can determine $O\left(3^{2n}\right)$ unique expressions each with $O(n^2)$ values of α at which the corresponding decision flips. Thus we can divide $\mathbb{R}$ into $O\left(n^2 3^{2n}\right)$ intervals over each of which the output of $\Phi_{\mathcal{A}_3,\mathcal{V}}(\alpha)$ is fixed. In fact, suppose $\mathcal{S} = \left\{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\right\}$ is a shatterable set of size m with witnesses $r_1, \ldots, r_m$. We can divide $\mathbb{R}$ into $O\left(mn^2 3^{2n}\right)$ intervals over each of which $\Phi_{\mathcal{A}_3,\mathcal{V}^{(i)}}(\alpha)$ is fixed for all $i \in [m]$ and therefore the corresponding labeling of $\mathcal{S}$ according to whether or not $\Phi_{\mathcal{A}_3(\alpha)}\left(\mathcal{V}^{(i)}\right) \le r_i$ is fixed as well for all $i \in [m]$. This means that $\mathcal{H}_{\mathcal{A}_3}$ can achieve only $O\left(mn^2 3^{2n}\right)$ labelings, which is at least 2^m for a shatterable set $\mathcal{S}$, so $m = O(n)$. □

Lemma 8. *For all objective functions $\Phi^{(p)}$, $Pdim\left(\mathcal{H}_{\mathcal{A}_3,\Phi^{(p)}}\right) = \Omega(n)$.*

Proof sketch. The crux of the proof is to show that there exists a clustering instance $\mathcal{V}$ over n points, a witness r, and a set of α's $1 = \alpha_0 < \alpha_1 < \cdots < \alpha_{2^N} < \alpha_{2^N+1} = 3$, where $N = \lfloor (n-8)/4 \rfloor$, such that $\Phi_{\mathcal{A}_3,\mathcal{V}}(\alpha)$ oscillates above and below r along the sequence of intervals (α_i, α_{i+1}). We finish the proof in a manner similar to Lemma 6 by constructing instances with fewer oscillations.

To construct $\mathcal{V}$, first we define two pairs of points which merge together regardless of the value of α. Call these merged pairs A and B. Next, we define a sequence of points p_i and q_i for $1 \le i \le N$ with distances set such that merges involving points in this sequence occur one after the other. In particular, each p_i merges with one of A or B while q_i merges with the other. Therefore, there are potentially 2^N distinct merge trees which can be created. Using induction to precisely set the distances, we show there are 2^N distinct values of α, each corresponding to a unique merge tree, thus enabling $\mathcal{A}_3$ to achieve all possible merge tree behaviors. Finally, we carefully add more points to the instance to control the oscillation of the cost function over these intervals as desired. □

This pseudo-dimension allows us to prove the following theorem.

Theorem 9. *Let Φ be a clustering objective and let Ψ be a pruning function. Given an input sample of size $m = O\left(\left(\frac{H}{\epsilon}\right)^2\left(n\log\frac{H}{\epsilon} + \log\frac{1}{\delta}\right)\right)$, Algorithm 5 (ϵ, δ)-learns the class $\mathcal{A}_3 \times \{\Psi\}$ with respect to the cost function Φ.*

Proof. The sample complexity analysis follows the same logic as the proof of Theorem 7. To prove that Algorithm 5 indeed finds the empirically best α, recall from the pseudo-dimension analysis that the cost as a function of α for any instance is a piecewise constant function with $O(n^2 3^{2n})$ discontinuities. In Step 1 of Algorithm 5, we solve for the values of α at which the discontinuities occur and add them to the set T. T therefore partitions α's range into $O(mn^2 3^{2n})$

Algorithm 5 An algorithm for finding an empirical cost minimizing algorithm in $\mathcal{A}_3$

Input: Sample $\mathcal{S} = \left\{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\right\}$.

1: Let $T = \emptyset$. For each sample $\mathcal{V}^{(i)} = \left(V^{(i)}, d^{(i)}\right) \in \mathcal{S}$, and for all $A, B, X, Y \subseteq V^{(i)}$, solve for α (if a solution exists) in the following equation and add the solutions to T:

$$\frac{1}{|A||B|} \sum_{p \in A, q \in B} (d(p,q))^\alpha = \frac{1}{|X||Y|} \sum_{x \in X, y \in Y} (d(x,y))^\alpha.$$

2: Order the elements of set $T \cup \{-\infty, +\infty\}$ as $\alpha_1 < \ldots < \alpha_{|T|}$. For each $0 \leq i \leq |T|$, pick an arbitrary α in the interval (α_i, α_{i+1}) and run $\mathcal{A}_3(\alpha)$ on all clustering instances in $\mathcal{S}$ to compute $\sum_{\mathcal{V} \in \mathcal{S}} \Phi_{\mathcal{A}_3(\alpha)}(\mathcal{V})$. Let $\hat{\alpha}$ be the value which minimizes $\sum_{\mathcal{V} \in \mathcal{S}} \Phi_{\mathcal{A}_3(\alpha)}(\mathcal{V})$.

Output: $\hat{\alpha}$

subintervals. Within each of these intervals, $\sum_{\mathcal{V} \in \mathcal{S}} \Phi_{\mathcal{A}_3(\alpha)}(\mathcal{V})$ is a constant function. Therefore, we pick any arbitrary α within each interval to evaluate the empirical cost over all samples, and find the empirically best α. □

In Appendix C.1, we prove more results about $\mathcal{A}_3$ assuming a natural restriction on the instance space $\mathbb{V}$. In particular, we show the pseudo-dimension can be drastically reduced if the number of unique distances in each problem instance is not too large. In Appendix C.2, we analyze classes of algorithms that interpolate between $\mathcal{A}_2$ and $\mathcal{A}_3$. A summary of results for all of our algorithm classes can be found in Table 2.

4.3 Dynamic programming pruning functions

In the previous section, we analyzed several classes of linkage-based merge functions assuming a fixed pruning function in the dynamic programming step of the standard linkage-based clustering algorithm, i.e. Step 2 of Algorithm 3. In this section, we analyze an infinite class of dynamic programming pruning functions and derive comprehensive sample complexity guarantees for learning the best merge function and pruning function in conjunction.

By allowing an application-specific choice of a pruning function, we significantly generalize the standard linkage-based clustering algorithm framework. Recall that in the algorithm selection model, we instantiated the `cost` function to be a generic clustering objective Φ. In the standard clustering algorithm framework, where Φ is defined to be any general $\Phi^{(p)}$ (which include objectives like k-means), the best choice of the pruning function for the algorithm selector is $\Phi^{(p)}$ itself as it would return the optimal pruning of the cluster tree for that instantiation of `cost`. However, when the goal of the algorithm selector is, for example, to provide solutions that are close to a ground truth clustering for each problem instance, the best choice for the pruning function is not obvious. In this case, we assume that the learning algorithm's training data consists of clustering instances that have been labeled by an expert according to the ground truth clustering. For example, this ground truth clustering might be a partition of a set of images based on their subject, or a partition of a set of proteins by function. On a fresh input data, we no longer have access to the expert or the ground truth, so we cannot hope to prune a cluster tree based on distance to the ground truth.

Instead, the algorithm selector must empirically evaluate how well pruning according to alternative objective functions, such as k-means or k-median, approximate the ground truth clustering on the labeled training data. In this way, we instantiate `cost` to be the distance of a clustering from the ground truth clustering. We guarantee that the empirically best pruning function from

a class of computable objectives is near-optimal in expectation over new problem instances drawn from the same distribution as the training data. Crucially, we are able to make this guarantee even though it is not possible to compute the `cost` of the algorithm's output on these fresh instances because the ground truth clustering is unknown.

Along these lines, we can also handle the case where the training data consists of clustering instances, each of which has been clustered according to an objective function that is NP-hard to compute. In this scenario, our learning algorithm returns a pruning objective function that is efficiently computable and which best approximates the NP-hard objective on the training data, and therefore will best approximate the NP-hard objective on future data. Hence, in this section, we analyze a richer class of algorithms defined by a class of merge functions and a class of pruning functions. The learner now has to learn the best combination of merge and pruning functions from this class.

To define this more general class of agglomerative clustering algorithms, let $\mathcal{A}$ denote a generic class of linkage-based merge functions (such as any of the classes $\mathcal{A}_i$ defined in Sections 4.1 and 4.2) parameterized by α. We also define a rich class of center-based clustering objectives for the dynamic programming step: $\mathcal{F} = \left\{\Psi^{(p)} \mid p > 0\right\}$ where $\Psi^{(p)}$ takes as input a partition $\mathcal{C} = \{C_1, C_2, \ldots, C_{k'}\}$ of n' points and a set of centers $\mathbf{c} = \{c_1, c_2, \ldots, c_{k'}\}$ such that $c_i \in C_i$. The function $\Psi^{(p)}$ is defined such that

$$\Psi^{(p)}(\mathcal{C}, \mathbf{c}) = \sqrt[p]{\sum_{C_i \in \mathcal{C}} \sum_{q \in C_i} (d(q, c_i))^p}. \tag{2}$$

Note that the definition of $\Psi^{(p)}$ is identical to $\Phi^{(p)}$, but we use this different notation so as not to confuse the dynamic programming function with the clustering objective function. Let $\mathcal{A}(\alpha)$ denote the α-linkage merge function from $\mathcal{A}$ and $\mathcal{F}(p)$ denote the pruning function $\Psi^{(p)}$. Earlier, for an abstract objective Φ, we bounded the pseudodimension of $\mathcal{H}_{\mathcal{A}\times\{\Psi\},\Phi} = \left\{\Phi_{(\mathcal{A}(\alpha),\Psi)} : \mathbb{V} \to \mathbb{R}_{\geq 0}\right\}$, where $\Phi_{(\mathcal{A}(\alpha),\Psi)}(\mathcal{V})$ denoted the cost of the clustering produced by building the cluster tree on $\mathcal{V}$ using the merge function $\mathcal{A}(\alpha)$ and then pruning the tree using a fixed pruning function Ψ. Now, we are interested in doubly-parameterized algorithms of the form $(\mathcal{A}(\alpha), \mathcal{F}(p))$ which uses the merge function $\mathcal{A}(\alpha)$ to build a cluster tree and then use the pruning function $\mathcal{F}(p)$ to prune it. To analyze the resulting class of algorithms, which we will denote by $\mathcal{A} \times \mathcal{F}$, we have to bound the pseudodimension of $\mathcal{H}_{\mathcal{A}\times\mathcal{F},\Phi} = \left\{\Phi_{(\mathcal{A}(\alpha),\mathcal{F}(p))} : \mathbb{V} \to \mathbb{R}_{\geq 0} \;\middle|\; p > 0\right\}$.

Recall that in order to show that pseudodimension of $\mathcal{H}_{\mathcal{A}\times\{\Psi\},\Phi}$ is upper bounded by $d_{\mathcal{H}_{\mathcal{A}}}$, we proved that, given a sample of m clustering instances over n nodes, we can split the real line into at most $O\left(m2^{d_{\mathcal{H}_{\mathcal{A}}}}\right)$ intervals such that as α ranges over a single interval, the m cluster trees returned by the α-linkage merge function are fixed. To extend this analysis to $\mathcal{H}_{\mathcal{A}\times\mathcal{F},\Phi}$, we first prove a similar fact in Lemma 9. Namely, given a single *cluster tree*, we can split the real line into a fixed number of intervals such that as p ranges over a single interval, the pruning returned by using the function $\Psi^{(p)}$ is fixed. We then show in Theorem 10 how to combine this analysis of the rich class of dynamic programming algorithms with our previous analysis of the possible merge functions to obtain a comprehensive analysis of agglomerative algorithms with dynamic programming.

We visualize the dynamic programming step of Algorithm 3 with pruning function $\Psi^{(p)}$ using a table such as Table 1, which corresponds to the cluster tree in Figure 5. Each row of the table corresponds to a sub-clustering value $k' \leq k$, and each column corresponds to a node of the corresponding cluster tree. In the column corresponding to node T and the row corresponding to the value k', we fill in the cell with the partition of T into k' clusters that corresponds to the best k'-pruning of the subtree rooted at T, $\left(\mathcal{C}_{T,k'}, \mathbf{c}_{T,k'}\right)$ as defined in Step 2 of Algorithm 3.

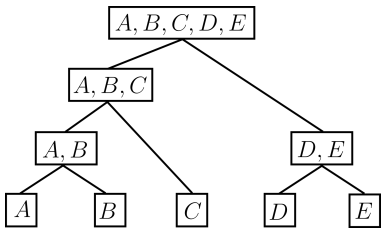

Figure 5: Cluster tree corresponding to Table 1.

		A	B	C	D	E	A, B	D, E	A, B, C	A, B, C, D, E
1	Clusters	$\{A\}$	$\{B\}$	$\{C\}$	$\{D\}$	$\{E\}$	$\{A, B\}$	$\{D, E\}$	$\{A, B, C\}$	$\{A, B, C, D, E\}$
	Centers	$\{A\}$	$\{B\}$	$\{C\}$	$\{D\}$	$\{E\}$	$\{A\}$	$\{E\}$	$\{C\}$	$\{C\}$
2	Clusters						$\{A\}, \{B\}$	$\{D\}, \{E\}$	$\{A, B\}, \{C\}$	$\{A, B, C\}, \{D, E\}$
	Centers						$\{A\}, \{B\}$	$\{D\}, \{E\}$	$\{A, C\}$	$\{C, E\}$
3	Clusters								$\{A\}, \{B\}, \{C\}$	$\{A, B\}, \{C\}, \{D, E\}$
	Centers								$\{A, B, C\}$	$\{A, C, E\}$

Table 1: Example dynamic programming table corresponding to the cluster tree in Figure 5 for $k = 3$.

Lemma 9. *Given a cluster tree $\mathcal{T}$ for a clustering instance $\mathcal{V} = (V, d)$ of n points, the positive real line can be partitioned into a set $\mathcal{I}$ of $O\left(n^{2(k+1)}k^{2k}\right)$ intervals such that for any $I \in \mathcal{I}$, the cluster tree pruning according to $\Psi^{(p)}$ is identical for all $p \in I$.*

Proof. To prove this claim, we will examine the dynamic programming (DP) table corresponding to the given cluster tree and the pruning function $\Psi^{(p)}$ as p ranges over the positive real line. As the theorem implies, we will show that we can split the positive real line into a set of intervals so that on a fixed interval I, as p ranges over I, the DP table under $\Psi^{(p)}$ corresponding to the cluster tree is invariant. No matter which $p \in I$ we choose, the DP table under $\Psi^{(p)}$ will be identical, and therefore the resulting clustering will be identical. After all, the output clustering is the bottom-right-most cell of the DP table since that corresponds to the best k-pruning of the node containing all points (see Table 1 for an example). We will prove that the total number of intervals is bounded by $O(n^{2(k+1)}k^{2k})$.

We will prove this lemma using induction on the row number k' of the DP table. Our inductive hypothesis will be the following. *The positive real line can be partitioned into a set $\mathcal{I}^{(k')}$ of $O\left(n^2 \prod_{j=1}^{k'} n^2 j\right)$ intervals such that for any $I^{(k')} \in \mathcal{I}^{(k')}$, as p ranges over $I^{(k')}$, the first k' rows of the DP table corresponding to $\Psi^{(p)}$ are invariant.* Notice that this means that the positive real line can be partitioned into a set $\mathcal{I}$ of $O\left(n^2 \prod_{j=1}^{k} n^2 j^2\right) = O\left(n^{2(k+1)}k^{2k}\right)$ intervals such that for any $I \in \mathcal{I}$, as p ranges over I, the DP table corresponding to $\Psi^{(p)}$ is invariant. Therefore, the resulting output clustering is invariant as well.

Base case ($k' = 1$). Let p be a positive real number. Consider the first row of the DP table corresponding to $\Psi^{(p)}$. Recall that each column in the DP table corresponds to a node T in the clustering tree where $T \subseteq V$. In the first row of the DP table and the column corresponding to node T, we fill in the cell with the single node T and the point $c \in T$ which minimizes $\Psi^{(p)}(\{T\}, \{c\}) = \sum_{q \in T}(d(q, c))^p$. The only thing that might change as we vary p is the center minimizing this objective.

Let v_1 and v_2 be two points in T. The point v_1 is a better candidate for the center of T than v_2 if and only if $\Psi^{(p)}(\{T\}, \{v_1\}) \leq \Psi^{(p)}(\{T\}, \{v_2\})$ which means that $\Psi^{(p)}(\{T\}, \{v_1\}) -$

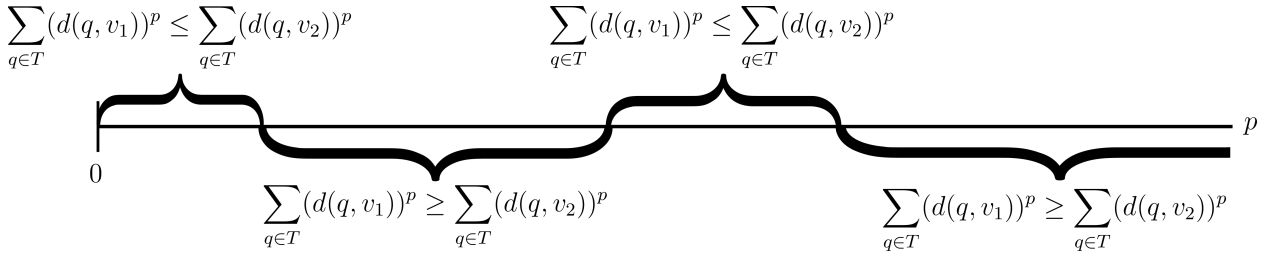

Figure 6: Partition of the positive real line based on whether or not $\sum_{q\in T}(d(q,v_1))^p \le \sum_{q\in T}(d(q,v_2))^p$ as p ranges $\mathbb{R}_{>0}$.

$\Psi^{(p)}(\{T\},\{v_2\}) \le 0$, or in other words, $\sum_{q\in T}(d(q,v_1))^p - \sum_{q\in T}(d(q,v_2))^p \le 0$. The equation $\sum_{q\in T}(d(q,v_1))^p - \sum_{q\in T}(d(q,v_2))^p$ has at most $2|T|$ zeros, so there are at most $2|T|+1$ intervals $I_1,\ldots,I_t$ which partition the positive real line such that for any I_i, as p ranges over I_i, whether or not $\Psi^{(p)}(\{T\},\{v_1\}) \le \Psi^{(p)}(\{T\},\{v_2\})$ is fixed. For example, see Figure 6. Every pair of points in T similarly partitions the positive real line into $2|T|+1$ intervals. If we merge all $|T|^2/2$ partitions — one partition for each pair of points in T — then we are left with at most $\frac{|T|^2}{2}\cdot 2|T|+1 = |T|^3+1$ intervals $I_1,\ldots,I_w$ partitioning the positive real line such that for any I_i, as p ranges over I_i, the point $v\in T$ which minimizes $\Psi^{(p)}(\{T\},\{v\})$ is fixed.

Since T is arbitrary, we can thus partition the real line for each node T' in the cluster tree. Again, this partition defines the center of the cluster T' as p ranges over the positive real line. If we merge the partition for every node $T\in\mathcal{T}$, then we are left with $\left(\sum_{T\in\mathcal{T}}|T|^3\right)+1 = O(n^4)$ intervals $I_1,\ldots,I_\ell$ such that as p ranges over any one interval I_i, the centers of all nodes in the cluster tree are fixed. In other words, for each T, the point $v_i\in T$ which minimizes $\Psi^{(p)}(\{T\},\{v_i\})$ is fixed. Of course, this means that the first row of the DP table is fixed as well. Therefore, the inductive hypothesis holds for the base case.

Inductive step. Consider the k'th row of the DP table. We know from the inductive hypothesis that the positive real line can be partitioned into a set $\mathcal{I}^{(k'-1)}$ of $O\left(n^2\prod_{j=1}^{k'-1}n^2j^2\right)$ intervals such that for any $I^{(k'-1)}\in\mathcal{I}^{(k'-1)}$, as p ranges over $I^{(k'-1)}$, the first $k'-1$ rows of the DP table corresponding to $\Psi^{(p)}$ are invariant.

Fix some interval $I^{(k'-1)}\in\mathcal{I}^{(k'-1)}$. Let T be a node in the cluster tree $\mathcal{T}$ and let T_L and T_R be the left and right children of T in $\mathcal{T}$ respectively. Notice that the pruning which belongs in the cell in the ith row and the column corresponding to T does not depend on the other cells in the ith row, but only on the cells in rows 1 through $i-1$. In particular, the pruning which belongs in this cell depends on the inequalities defining which $i'\in\{1,\ldots,k'-1\}$ minimizes $\Psi^{(p)}\left(\mathcal{C}_{T_L,i'}\cup\mathcal{C}_{T_R,k'-i'},\mathbf{c}_{T_L,i'}\cup\mathbf{c}_{T_R,k'-i'}\right)$. We will now examine this objective function and show that the minimizing i', and therefore the optimal pruning, only changes a small number of times as p ranges over $I^{(k'-1)}$.

For an arbitrary $i'\in\{1,\ldots,k'-1\}$, since i' and $k'-i'$ are both strictly less than k', the best i'-pruning of T_L $(\mathcal{C}_{T_R,i'},\mathbf{c}_{T_R,i'})$ is exactly the entry in the i'th row of the DP table and the column corresponding to T_L. Similarly, the best $k'-i'$-pruning of T_R, $(\mathcal{C}_{T_R,k'-i'},\mathbf{c}_{T_R,k'-i'})$ is exactly the entry in the $k'-i'$th row of the DP table and the column corresponding to T_R. Crucially, these entries do not change as we vary $p\in I^{(k'-1)}$, thanks to the inductive hypothesis.

Therefore, for any $i',i''\in\{1,\ldots,k'-1\}$, we know that for all $p\in I^{(k'-1)}$, the k'-pruning of T corresponding to the combination of the best i'-pruning of T_L and the best $k'-i'$ pruning of T_R is fixed and can be denoted as $(\mathcal{C}',\mathbf{c}')$. Similarly, the k'-pruning of T corresponding to the combination of the best i''-pruning of T_L and the best $k'-i''$ pruning of T_R is fixed and can

be denoted as $(\mathcal{C}'', \mathbf{c}'')$. Then, for any $p \in I^{(k'-1)}$, $(\mathcal{C}', \mathbf{c}')$ is a better pruning than $(\mathcal{C}'', \mathbf{c}'')$ if and only if $\Psi^{(p)}(\mathcal{C}', \mathbf{c}') \leq \Psi^{(p)}(\mathcal{C}'', \mathbf{c}'')$. In order to analyze this inequality, let us consider the equivalent inequality $\left(\Psi^{(p)}(\mathcal{C}', \mathbf{c}')\right)^p \leq \left(\Psi^{(p)}(\mathcal{C}'', \mathbf{c}'')\right)^p$ i.e., $\left(\Psi^{(p)}(\mathcal{C}', \mathbf{c}')\right)^p - \left(\Psi^{(p)}(\mathcal{C}'', \mathbf{c}'')\right)^p \leq 0$. Now, to expand this expression let $\mathcal{C}' = \{C'_1, C'_2, \ldots, C'_{k'}\}$ and $\mathbf{c}' = \{c'_1, c'_2 \ldots, c'_{k'}\}$ and similarly $\mathcal{C}'' = \{C''_1, C''_2, \ldots, C'_{k'}\}$ and $\mathbf{c}'' = \{c''_1, c''_2 \ldots, c''_{k'}\}$. Then, this inequality can then be written as,

$$\sum_{i=1}^{k'} \sum_{q \in C'_i} \left(d(q, c'_i)\right)^p - \sum_{i=1}^{k'} \sum_{q \in C''_i} \left(d(q, c''_i)\right)^p \leq 0.$$

The equation $\sum_{i=1}^{k'} \sum_{q \in C'_i} (d(q, c'_i))^p - \sum_{i=1}^{k'} \sum_{q \in C''_i} (d(q, c''_i))^p$ has has at most $2n$ zeros as p ranges over $I^{(k'-1)}$. Therefore, there are at most $2n+1$ subintervals partitioning $I^{(k'-1)}$ such that as p ranges over one subinterval, the smaller of $\Psi^{(p)}(\mathcal{C}', \mathbf{c}')$ and $\Psi^{(p)}(\mathcal{C}'', \mathbf{c}'')$ is fixed. In other words, as p ranges over one subinterval, either the combination of the best i'-pruning of T's left child and the best $(k'-i')$-pruning of T's right child is better than the combination of the best i''-pruning of T's left child with the best $(i-i'')$-pruning of T's right child, or vice versa. For all pairs $i', i'' \in \{1, \ldots, k'-1\}$, we can similarly partition I into at most $2n+1$ subintervals defining the better of the two prunings. If we merge all $(k'-1)^2/2$ partitions of $I^{(k'-1)}$, we have $\frac{(k'-1)^2}{2} \cdot 2n + 1 = (k'-1)^2 n + 1$ total subintervals of $I^{(k'-1)}$ such that as p ranges over a single subinterval,

$$\text{argmin}_{i' \in \{1,\ldots,k'-1\}} \Psi^{(p)}\left(\mathcal{C}_{T_L,i'} \cup \mathcal{C}_{T_R,k'-i'}, \mathbf{c}_{T_L,i'} \cup \mathbf{c}_{T_R,k'-i'}\right)$$

is fixed. Since these equations determine the entry in the ith row of the DP table and the column corresponding to the node T, we have that this entry is also fixed as p ranges over a single subinterval in $I^{(k'-1)}$.

The above partition of $I^{(k'-1)}$ corresponds to only a single cell in the k'th row of the DP table. Considering the k'th row of the DP table as a whole, we must fill in at most $2n$ entries, since there are at most $2n$ columns of the DP table. For each column, there is a corresponding partition of $I^{(k'-1)}$ such that as p ranges over a single subinterval in the partition, the entry in the k'th row and that column is fixed. If we merge all such partitions, we are left with a partition of $I^{(k'-1)}$ consisting of at most $2n^2(i-1)^2 + 1$ intervals such that as p ranges over a single interval, the entry in every column of the k'th row is fixed. As these intervals are subsets of $I^{(k'-1)}$, by assumption, the first $k'-1$ rows of the DP table are also fixed. Therefore, the first k' rows are fixed.

To recap, we fixed an interval $I^{(k'-1)}$ such that as p ranges over $I^{(k'-1)}$, the first $k'-1$ rows of the DP table are fixed. By the inductive hypothesis, there are $O\left(n^2 \prod_{j=1}^{k'-1} n^2 j^2\right)$ such intervals. Then, we showed that $I^{(k'-1)}$ can be partitioned into $2n^2(k'-1)^2 + 1$ intervals such that for any one subinterval $I^{(k')}$, as p ranges over $I^{(k')}$, the first k' rows of the DP table are fixed. Therefore, there are $O\left(n^2 \prod_{j=1}^{k'} n^2 j^2\right)$ total intervals such that as p ranges over a single interval, the first k' rows of the DP table are fixed.

Aggregating this analysis over all k rows of the DP table, we have that there are

$$O\left(n^2 \prod_{k=1}^{k} n^2 k'^2\right) = O\left(n^{2(k+1)} k^{2k}\right)$$

intervals such that the entire DP table is fixed so long as p ranges over a single interval. □

Theorem 10. *Let $\mathcal{A}$ be a class of merge functions parameterized by a real value α and let Φ be an arbitrary clustering objective. Given m clustering instances, suppose we can partition the*

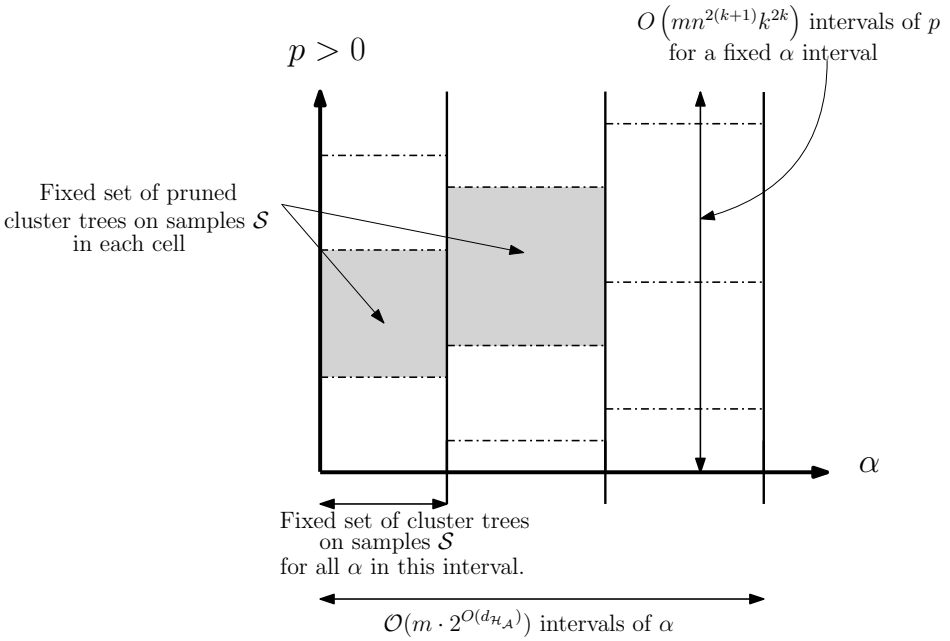

Figure 7: For a fixed set of samples $\mathcal{S}$ and we can first partition the range of values of α such that each α interval corresponds to a fixed set of cluster trees. Then, for each α interval, we can partition the range of values of p such that each interval of p results in the same set of pruned trees. In other words, we can partition the parameter space in $\mathbb{R}^2$ into cells that produce the same set of solutions for all values of the parameters within that cell.

domain of α into $O\left(m \cdot 2^{O(d_{\mathcal{H}_\mathcal{A}})}\right)$ intervals such that as α ranges over a single interval, the m cluster trees returned by the α-linkage merge function from $\mathcal{A}$ are fixed. Then $Pdim(\mathcal{H}_{\mathcal{A}\times\mathcal{F},\Phi}) = O\left(d_{\mathcal{H}_\mathcal{A}} + k \log n\right)$.

Proof. Let $\mathcal{S}$ be a set of clustering instances. Fix a single interval of α (as shown along the horizontal axis in Figure 7) where the set of cluster trees returned by the α-linkage merge function from $\mathcal{A}$ is fixed across all samples. We know from Lemma 9 that we can split the real line into a fixed number of intervals such that as p ranges over a single interval (as shown along the vertical axis in Figure 7), the dynamic programming (DP) table is fixed for all the samples, and therefore the resulting set of clusterings is fixed. In particular, for a fixed α interval, each of the m samples has its own $O\left(n^{2(k+1)}k^{2k}\right)$ intervals of p, and when we merge them, we are left with $O\left(mn^{2(k+1)}k^{2k}\right)$ intervals such that as p ranges over a single interval, each DP table for each sample is fixed, and therefore the resulting clustering for each sample is fixed. Since there are $O\left(m2^{d_{\mathcal{H}_\mathcal{A}}}\right)$ such α intervals, each inducing $O\left(mn^{2(k+1)}k^{2k}\right)$ such p intervals in total, we have $O\left(2^{d_{\mathcal{H}_\mathcal{A}}}m^2n^{2(k+1)}k^{2k}\right)$ cells in $\mathbb{R}^2$ such that if (α, p) is in one fixed cell, the resulting clustering across all samples is fixed. If $\mathcal{H}_{\mathcal{A}\times\mathcal{F}}$ shatters $\mathcal{S}$, then it must be that $2^m = O\left(2^{d_{\mathcal{H}_\mathcal{A}}}m^2n^{2(k+1)}k^{2k}\right)$, which means that $m = O\left(\log\left(2^{d_{\mathcal{H}_\mathcal{A}}}n^{2(k+1)}k^{2k}\right)\right) = O\left(d_{\mathcal{H}_\mathcal{A}} + k\log n\right)$. □

Given a set of samples, a technique for finding the empirically best algorithm from $\mathcal{A} \times \mathcal{F}$ follows naturally from the above analysis. We first partition the range of feasible values of α into $O\left(2^{O(d_{\mathcal{H}_\mathcal{A}})}\right)$ intervals (as discussed in detail in Section 4.1). For each α interval, we can find the

fixed set of cluster trees on the samples by running $\mathcal{A}(\alpha)$ on the samples for a single arbitrary value of α in that interval. Now, based on these cluster trees, we can partition the values of p into $O\left(m \cdot n^{2(k+1)}k^{2k}\right)$ intervals as discussed in the proof for Lemma 9. For each interval of p, we only need to pick an arbitrary value of p and run $\mathcal{F}(p)$ to prune the cluster trees and determine the fixed empirical cost corresponding to that interval of p and α. Iterating over all partitions of the parameter space in this manner, we can find parameters that result in the best empirical cost. We summarize this result in the following theorem.

Theorem 11. *Let Φ be a clustering objective. Given an input sample of size*

$$m = O\left(\left(\frac{H}{\epsilon}\right)^2\left((d_{\mathcal{H}_\mathcal{A}} + \log n)\log\frac{H}{\epsilon} + \log\frac{1}{\delta}\right)\right),$$

it is possible to (ϵ, δ)-learn the class of algorithms $\mathcal{A} \times \mathcal{F}$ with respect to the cost function Φ. Moreover, this procedure is efficient if the following conditions hold:

1. *k is constant, which ensures that the partition of p values is polynomial in n.*
2. *$2^{d_{\mathcal{H}_\mathcal{A}}}$ is polynomial in n, which ensures that the partition of α values is polynomial in n.*
3. *It is possible to efficiently compute the partition of α into intervals so that on a single interval I, for all $\alpha \in I$, the m cluster trees returned by α-linkage performed on $\mathcal{S}$ are fixed.*

5 Discussion and open questions

In this work, we show how to learn near-optimal algorithms over several infinite, rich classes of SDP rounding algorithms and agglomerative clustering algorithms with dynamic programming. We provide computationally efficient learning algorithms for many of these problems and we push the boundaries of learning theory by developing techniques to compute the pseudo-dimension of intricate, multi-stage classes of integer-quadratic programming approximation algorithms and clustering algorithms. We derive tight pseudo-dimension bounds for the classes we study, which lead to strong sample complexity guarantees. We hope that our techniques will lead to theoretical guarantees in other areas where empirical methods for application-specific algorithm selection and portfolio selection have been developed.

There are many open avenues for future research in this area. In this work, we focused on algorithm families containing only computationally efficient algorithms. However, oftentimes in empirical AI research, the algorithm families in question contain procedures that are too slow to run to completion on many training instances. In this situation, we would not be able to determine the exact empirical cost of an algorithm on the training set. Could we still make strong, provable guarantees for application-specific algorithm selection in this scenario?

Acknowledgments. This work was supported in part by grants NSF-CCF 1535967, NSF CCF-1422910, NSF IIS-1618714, a Sloan Fellowship, a Microsoft Research Fellowship, a NSF Graduate Research Fellowship, a Microsoft Research Women's Fellowship, and a National Defense Science and Engineering Graduate (NDSEG) fellowship.

References

[1] Bahram Alidaee, Gary Kochenberger, and Ahmad Ahmadian. 0-1 quadratic programming approach for optimum solutions of two scheduling problems. *International Journal of Systems Science*, 25(2):401–408, 1994.

[2] Noga Alon, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. Quadratic forms on graphs. *Inventiones mathematicae*, 163(3):499–522, 2006.

[3] Pranjal Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1):49–54, 2012.

[4] Maria-Florina Balcan, Nika Haghtalab, and Colin White. k-center clustering under perturbation resilience. In *Proceedings of the 43rd annual International Colloquium on Automata, Languages, and Programming (ICALP)*, 2016.

[5] Maria-Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155, 2016.

[6] Ahron Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*, volume 2. Siam, 2001.

[7] Yves Caseau, François Laburthe, and Glenn Silverstein. A meta-heuristic factory for vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 144–158. Springer, 1999.

[8] Moses Charikar and Anthony Wirth. Maximizing quadratic programs: extending Grothendieck's inequality. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 54–60, 2004.

[9] Jim Demmel, Jack Dongarra, Victor Eijkhout, Erika Fuentes, Antoine Petitet, Rich Vuduc, R Clint Whaley, and Katherine Yelick. Self-adapting linear algebra algorithms and software. *Proceedings of the IEEE*, 93(2):293–312, 2005.

[10] Iain Dunning, Swati Gupta, and John Silberholz. What works best when? a systematic evaluation of heuristics for max-cut and qubo. 2015.

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

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

[13] Anna Grosswendt and Heiko Roeglin. Improved analysis of complete linkage clustering. In *European Symposium of Algorithms*, volume 23, pages 656–667. Springer, 2015.

[14] Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 123–134, 2016.

[15] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and computation*, 100(1):78–150, 1992.

[16] Fredrik D Johansson, Ankani Chattoraj, Chiranjib Bhattacharyya, and Devdatt Dubhashi. Weighted theta functions and embeddings with applications to max-cut, clustering and summarization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1018–1026, 2015.

[17] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[18] Manuel Laguna, Abraham Duarte, and Rafael Martí. Hybridizing the cross-entropy method: An application to the max-cut problem. *Computers & Operations Research*, 36(2):487–498, 2009.

[19] DJ Laughhunn. Quadratic binary programming with application to capital-budgeting problems. *Operations Research*, 18(3):454–461, 1970.

[20] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM (JACM)*, 56(4):22, 2009.

[21] Ryan O'Donnell and Yi Wu. An optimal SDP algorithm for max-cut, and equally optimal long code tests. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 335–344, 2008.

[22] AT Phillips and JB Rosen. A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization*, 4(2):229–241, 1994.

[23] John R Rice. The algorithm selection problem. *Advances in computers*, 15:65–118, 1976.

[24] Sagi Snir and Satish Rao. Using max cut to enhance rooted trees consistency. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):323–333, 2006.

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

[26] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[27] Jun Wang, Tony Jebara, and Shih-Fu Chang. Semi-supervised learning using greedy max-cut. *Journal of Machine Learning Research*, 14(Mar):771–800, 2013.

[28] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University press, 2011.

[29] Christoph Witzgall. Mathematical methods of site selection for electronic message systems (EMS). *NASA STI/Recon Technical Report N*, 76:18321, 1975.

[30] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, pages 565–606, 2008.

[31] Chihiro Yoshimura, Masanao Yamaoka, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, Ken-ichi Kawarabayashi, and Hiroyuki Mizuno. Uncertain behaviours of integrated circuits improve computational performance. *Scientific reports*, 5, 2015.

[32] Uri Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to max cut and other problems. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 679–687, 1999.

A Proofs from Section 3

Proof of Theorem 2. Theorem 2 follows directly from the following lemma.

Lemma 10. *Suppose that m is sufficiently large to ensure that with probability at least $1-\delta$ over a draw of m samples $\left(A^{(i)}, \vec{Z}^{(i)}\right) \sim \mathcal{D} \times \mathcal{Z}$, for all s-linear functions ϕ_s,*

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

Then with probability at least $1-\delta$, if $\hat{s}$ maximizes $\frac{1}{m}\sum_{i=1}^{m} \mathtt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right)$ and s^* maximizes $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_s\left(A\right)\right]$, then $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_{s^*}\left(A\right)\right] - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_{\hat{s}}\left(A\right)\right] < \epsilon$.

Proof. Notice that since $\mathcal{D} \times \mathcal{Z}$ is a product distribution, we have that

$$\mathop{\mathbb{E}}_{(A,\vec{Z})\sim\mathcal{D}\times\mathcal{Z}} \left[\mathtt{slin}_s\left(A, \vec{Z}\right)\right] = \mathop{\mathbb{E}}_{A\sim\mathcal{D}} \left[\mathop{\mathbb{E}}_{\vec{Z}\sim\mathcal{Z}} \left[\mathtt{slin}_s\left(A, \vec{Z}\right)\right]\right] = \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_s\left(A\right)\right],$$

so we know that with probability at least $1-\delta$, for all s-linear functions ϕ_s,

$$\left| \frac{1}{m} \sum_{i=1}^{m} \mathtt{slin}_s\left(A^{(i)}, \vec{Z}^{(i)}\right) - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_s\left(A\right)\right] \right| < \frac{\epsilon}{2}.$$

By assumption, we know that with probability at least $1-\delta$,

$$\left| \frac{1}{m} \sum_{i=1}^{m} \mathtt{slin}_{\hat{s}}\left(A^{(i)}, \vec{Z}^{(i)}\right) - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_{\hat{s}}\left(A\right)\right] \right| < \frac{\epsilon}{2}$$

and

$$\left| \frac{1}{m} \sum_{i=1}^{m} \mathtt{slin}_{s^*}\left(A^{(i)}, \vec{Z}^{(i)}\right) - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_{s^*}\left(A\right)\right] \right| < \frac{\epsilon}{2},$$

which means that $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_{s^*}\left(A\right)\right] - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\mathtt{slin}_{\hat{s}}\left(A\right)\right] < \epsilon$. □

□

Proof of Lemma 3. In order to prove that the pseudo dimension of $\mathcal{H}_{slin}$ is at least $c \log n$ for some c, we must present a set $\mathcal{S} = \left\{\left(G^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(G^{(m)}, \vec{Z}^{(m)}\right)\right\}$ of $m = c \log n$ graphs and projection vectors that can be shattered by $\mathcal{H}_{slin}$. In other words, there exist m witnesses $r_1, \ldots, r_m$ and $2^m = n^c$ s values $H = \{s_1, \ldots, s_{n^c}\}$ such that for all $T \subseteq [m]$, there exists $s_T \in H$ such that if $j \in T$, then $\mathtt{slin}_{S_T}\left(G^{(j)}, \vec{Z}^{(j)}\right) > r_j$ and if $j \notin T$, then $\mathtt{slin}_{S_T}\left(G^{(j)}, \vec{Z}^{(j)}\right) \leq r_j$.

To build $\mathcal{S}$, we will use the same graph G for all $G^{(j)}$ and we will vary $\vec{Z}^{(j)}$. We set G to be the graph composed of $\lfloor n/4 \rfloor$ disjoint copies of K_4. If $n = 4$, then a simple calculation confirms that an optimal max-cut SDP embedding of G is

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} -1/3 \\ 2\sqrt{2}/3 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} -1/3 \\ -\sqrt{2}/3 \\ \sqrt{2/3} \\ 0 \end{pmatrix}, \begin{pmatrix} -1/3 \\ -\sqrt{2}/3 \\ -\sqrt{2/3} \\ 0 \end{pmatrix} \right\}.$$

Therefore, for $n > 4$, an optimal embedding is the set of n vectors $SDP(G)$ such that for all $i \in \{0, \ldots, \lfloor n/4 \rfloor - 1\}$,

$$\vec{e}_{4i+1}, -\frac{1}{3}\vec{e}_{4i+1} + \frac{2\sqrt{2}}{3}\vec{e}_{4i+2}, -\frac{1}{3}\vec{e}_{4i+1} - \frac{\sqrt{2}}{3}\vec{e}_{4i+2} + \sqrt{\frac{2}{3}}\vec{e}_{4i+3}, -\frac{1}{3}\vec{e}_{4i+1} - \frac{\sqrt{2}}{3}\vec{e}_{4i+2} - \sqrt{\frac{2}{3}}\vec{e}_{4i+3}$$

are elements $SDP(G)$.

We now define the set of m vectors $\vec{Z}^{(j)}$. First, we set $\vec{Z}^{(1)}$ to be the vector

$$\vec{Z}^{(1)} = \left(7^0, 5\cdot 7^0, 5\cdot 7^0, 7^0, 7^1, 5\cdot 7^1, 5\cdot 7^1, 7^1, 7^2, 5\cdot 7^2, 5\cdot 7^2, 7^2, 7^3, 5\cdot 7^3, 5\cdot 7^3, 7^3, \ldots\right).$$

In other words, it is the concatenation the vector $7^i(1,5,5,1)$ for all $i > 0$. Next, $\vec{Z}^{(2)}$ is defined as

$$\vec{Z}^{(2)} = \left(7^0, 5\cdot 7^0, 5\cdot 7^0, 7^0, 0, 0, 0, 0, 7^2, 5\cdot 7^2, 5\cdot 7^2, 7^2, 0, 0, 0, 0, \ldots\right),$$

so $\vec{Z}^{(2)}$ is the same as $\vec{Z}^{(1)}$ for all even powers of 7, and otherwise its entries are 0. In a similar vein,

$$\vec{Z}^{(3)} = \left(7^0, 5\cdot 7^0, 5\cdot 7^0, 7^0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7^4, 5\cdot 7^4, 5\cdot 7^4, 7^4, \ldots\right).$$

To pin down this pattern, we set $\vec{Z}^{(j)}$ to be the same as $\vec{Z}^{(1)}$ for all entries of the form $7^{i2^{j-1}}(1,5,5,1)$ for $i \geq 0$, and otherwise its entries are 0.

We set the following positive, increasing constants which will appear throughout the remaining analysis:

$$\begin{aligned}
a &= (1,0,0,0)\cdot(1,5,5,1) &&= 1\\
b &= (-1/3, -\sqrt{2}/3, \sqrt{2/3}, 0)\cdot(1,5,5,1) &&= 5\sqrt{2/3} - \tfrac{5\sqrt{2}+1}{3}\\
c &= (-1/3, 2\sqrt{2}/3, 0, 0)\cdot(1,5,5,1) &&= \tfrac{10\sqrt{2}-1}{3}\\
d &= \left|(-1/3, -\sqrt{2}/3, \sqrt{2/3}, 0)\cdot(1,5,5,1)\right| &&= 5\sqrt{2/3} + \tfrac{5\sqrt{2}+1}{3}.
\end{aligned}$$

We also set $\tilde{c} = b + c + bc - d - bd - cd$ and we claim that the witnesses

$$\begin{aligned}
r_1 &= \frac{1}{2} - \frac{1}{3n}\left(\frac{b}{c^2} - 1\right)\\
r_j &= \frac{1}{2} - \frac{\tilde{c}}{3n7^{2^{j-1}-2}d^2} \qquad\qquad j > 1
\end{aligned}$$

are sufficient to prove that this set is shatterable, and we will spend the remainder of the proof showing that this is true.

Now, the domain of $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ can be split into intervals on which it has a simple, fixed form. These intervals begin at 1 and have the form $\left[7^{i2^{j-1}}, 7^{(i+1)2^{j-1}}\right)$, for $i \geq 0$. It is straightforward matter of calculations to check that for $s \in \left[7^{i2^{j-1}}, 7^{(i+1)2^{j-1}}\right)$,

$$\texttt{slin}_{G,\vec{Z}^{(j)}}(s) = \begin{cases}
\frac{1}{2} - \frac{1}{3n}\left(\frac{1}{s^2}\left[\tilde{c}\sum_{k=0}^{i-1} 7^{2k2^{j-1}}\right] + \frac{7^{2i2^{j-1}}}{s} - 1\right) & \text{if } s \in \left[7^{i2^{j-1}}, 7^{i2^{j-1}}b\right)\\
\frac{1}{2} - \frac{1}{3n}\left(\frac{1}{s^2}\left[\tilde{c}\sum_{k=0}^{i-1} 7^{2k2^{j-1}} + 7^{2i2^{j-1}}b\right] - 1\right) & \text{if } s \in \left[7^{i2^{j-1}}b, 7^{i2^{j-1}}c\right)\\
\frac{1}{2} - \frac{1}{3n}\left(\frac{1}{s^2}\left[\tilde{c}\sum_{k=0}^{i-1} 7^{2k2^{j-1}} + 7^{2i2^{j-1}}(b + c + bc)\right] - \frac{7^{2i2^{j-1}}(1+b+c)}{s}\right) & \text{if } s \in \left[7^{i2^{j-1}}c, 7^{i2^{j-1}}d\right)\\
\frac{1}{2} - \frac{1}{3n}\left(\frac{1}{s^2}\left[\tilde{c}\sum_{k=0}^{i} 7^{2k2^{j-1}}\right]\right) & \text{if } s \in \left[7^{i2^{j-1}}d, 7^{(i+1)2^{j-1}}\right)
\end{cases}$$

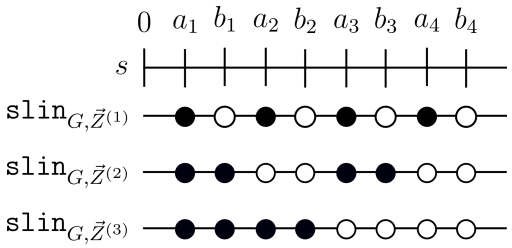

Figure 8: Depiction of $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ as s increases from 0. A black dot means that $\texttt{slin}_{G,\vec{Z}^{(j)}}(s) \geq r_j$ and a white dot means that $\texttt{slin}_{G,\vec{Z}^{(j)}}(s) < r_j$. Here, $a_i = 7^{i-1}c$ and $b_i = 7^{i-1}d$.

(We note here that the power of 7 pattern was chosen so that these intervals are well defined, since $7^i d < 7^{i+1}$.)

We call the following increasing sequence of numbers *points of interest*, which we use to prove that this set is shattered: $\left\{7^0c, 7^0d, 7^1c, 7^1d, 7^2c, 7^2d, \ldots, 7^ic, 7^id, \ldots\right\}$

We make two claims about these points of interest:

1. $\texttt{slin}_{G,\vec{Z}^{(1)}}(s)$ is above its witness whenever $s = 7^ic$ and it is below its witness whenever $s = 7^id$ for $i \geq 0$.
2. Let $j > 1$ and consider $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$. There are 2^j points of interest per interval

$$\left[7^{i2^{j-1}}, 7^{(i+1)2^{j-1}}\right).$$

   On the first half of these points of interest, $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is greater than its witness and on the second half, $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is less than its witness.

These claims are illustrated by the dots in Figure 8. Together, these claims imply that $\mathcal{S}$ can be shattered because for any vector $\vec{b} \in \{0,1\}^m$, there exists a point of interest s such that $\texttt{slin}_s(\mathcal{S})$ induces the binary labeling $\vec{b}$ on $\mathcal{S}$.

The first claim is true because

$$\begin{aligned}\texttt{slin}_{G,\vec{Z}^{(1)}}\left(7^ic\right) &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i}c^2}\left[\tilde{c}\sum_{k=0}^{i-1}7^{2k} + 7^{2i}b\right] - 1\right) \\ &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i}c^2}\left[\tilde{c}\cdot\frac{7^{2i}-1}{7^2-1} + 7^{2i}b\right] - 1\right),\end{aligned}$$

which is an increasing function of i, so it is minimized when $i = 0$, where

$$\texttt{slin}_{G,\vec{Z}^{(1)}}\left(7^0c\right) = \frac{1}{2} - \frac{1}{3n}\left(\frac{b}{c^2} - 1\right) = r_1$$

so $\texttt{slin}_{G,\vec{Z}^{(1)}}\left(7^ic\right)$ is always at least its witness. Further,

$$\begin{aligned}\texttt{slin}_{G,\vec{Z}^{(1)}}\left(7^id\right) &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i}d^2}\left[\tilde{c}\sum_{k=0}^{i}7^{2k}\right]\right) \\ &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i}d^2}\left[\tilde{c}\cdot\frac{7^{2(i+1)}-1}{48}\right]\right)),\end{aligned}$$

which is again an increasing function in i, with a limit of

$$\frac{1}{2} - \frac{49\tilde{c}}{144nd^2} < r_1.$$

Therefore, $\texttt{slin}_{G,\vec{Z}^{(1)}}\left(7^i d\right)$ is always less than its witness, and we may conclude that the first claim is always true.

For the second claim, notice that

$$7^{i2^{j-1}}c < 7^{i2^{j-1}}d < 7^{i2^{j-1}+1}c < 7^{i2^{j-1}+1}d < 7^{i2^{j-1}+2}c \cdots < 7^{i2^{j-1}+2^{j-1}}c = 7^{(i+1)2^{j-1}}c,$$

so there are 2^j points of interest per interval $\left[7^{i2^{j-1}}c, 7^{(i+1)2^{j-1}}c\right)$, as claimed. The first two points of interest, $7^{i2^{j-1}}c$ and $7^{i2^{j-1}}d$, fall in an interval where $\texttt{slin}_{G,\vec{Z}^{(j)}}$ is decreasing in s. Therefore, it is minimized when $s = 7^{i2^{j-1}}d$, where

$$\begin{aligned}\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}}d\right) &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i2^{j-1}}d^2}\left[\tilde{c}\sum_{k=0}^{i} 7^{2k2^{j-1}}\right]\right) \\ &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i2^{j-1}}d^2}\left[\tilde{c}\cdot\frac{7^{(i+1)2^j}-1}{7^{2^j}-1}\right]\right).\end{aligned}$$

Simple calculations show that $\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}}d\right)$ is an increasing function in i, so it is minimized when $i = 0$, where $\texttt{slin}_{G,\vec{Z}^{(j)}}(d) = \frac{1}{2} - \frac{\tilde{c}}{3nd^2} > r_j$, as desired.

The remaining points of interest fall in the interval $\left[7^{i2^{j-1}}d, 7^{(i+1)2^{j-1}}\right)$, so $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ has the form $\frac{1}{2} - \frac{1}{3n}\left(\frac{1}{s^2}\left[\tilde{c}\sum_{k=0}^{i} 7^{2k2^{j-1}}\right]\right)$. This segment of the function has a negative derivative, so it is decreasing.

If $j = 2$, then the points of interest we already considered, $7^{i2^{j-1}}c$ and $7^{i2^{j-1}}d$, make up half of the 2^j points of interest in the interval $\left[7^{i2^{j-1}}c, 7^{(i+1)2^{j-1}}c\right)$. Therefore, we only need to show that when s equals $7^{i2^{j-1}+1}c$ and $7^{i2^{j-1}+1}d$, then $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is less than its witness. As we saw, $\texttt{slin}_{G,\vec{Z}^{(j)}}$ is decreasing on this segment, so it is enough to show that $\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}+1}c\right)$ is less than its witness. To this end,

$$\begin{aligned}\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}+1}c\right) &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i2^{j-1}+2}c^2}\left[\tilde{c}\sum_{k=0}^{i} 7^{2k2^{j-1}}\right]\right) \\ &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2i2^{j-1}+2}c^2}\left[\tilde{c}\cdot\frac{7^{(i+1)2^j}-1}{7^{2^j}-1}\right]\right).\end{aligned}$$

This is an increasing function of i with a limit of $\frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^2c^2}\left[\tilde{c}\cdot\frac{7^{2^j}}{7^{2^j}-1}\right]\right) < r_j$ when $j = 2$. Therefore, when s equals $7^{i2^{j-1}+1}b$ and $7^{i2^{j-1}+1}c$, then $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is less than its witness.

Finally, if $j > 2$, since $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is decreasing on the interval $\left[7^{i2^{j-1}}c, 7^{(i+1)2^{j-1}}\right)$, we must only check that at the $\left(2^{j-1}-1\right)^{th}$ point of interest $\left(7^{i2^{j-1}+2^{j-2}-1}d\right)$, $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is greater than its witness and at the $\left(2^{j-1}+1\right)^{th}$ point of interest $\left(7^{i2^{j-1}+2^{j-2}}c\right)$, $\texttt{slin}_{G,\vec{Z}^{(j)}}(s)$ is less than its witness. To this end,

$$\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}+2^{j-2}-1}d\right) = \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{i2^j+2^{j-1}-2}d^2}\left[\tilde{c}\sum_{k=0}^{i} 7^{2k2^{j-1}}\right]\right)$$

$$= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{i2^j+2^{j-1}-2}d^2}\left[\tilde{c}\cdot\frac{7^{(i+1)2^j}-1}{7^{2^j}-1}\right]\right).$$

This function is increasing in i, so it is minimized when $i = 0$, where

$$\begin{aligned}\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{2^{j-2}-1}d\right) &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{2^{j-1}-2}d^2}\left[\tilde{c}\cdot\frac{7^{2^j}-1}{7^{2^j}-1}\right]\right)\\ &= \frac{1}{2} - \frac{1}{3n}\left(\frac{\tilde{c}}{7^{2^{j-1}-2}d^2}\right) = r_j.\end{aligned}$$

Therefore, $\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}+2^{j-2}-1}d\right) \geq r_j$ for all i.

Next,

$$\begin{aligned}\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}+2^{j-2}}c\right) &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{i2^j+2^{j-1}}c^2}\left[\tilde{c}\sum_{k=0}^{i}7^{2k2^{j-1}}\right]\right)\\ &= \frac{1}{2} - \frac{1}{3n}\left(\frac{1}{7^{i2^j+2^{j-1}}c^2}\left[\tilde{c}\cdot\frac{7^{(i+1)2^j}-1}{7^{2^j}-1}\right]\right).\end{aligned}$$

which is an increasing function in i, with a limit of

$$\frac{1}{2} - \frac{\tilde{c}7^{2^{j-1}}}{3nc^2\left(7^{2^j}-1\right)}$$

as i tends toward infinity. Therefore,

$$\texttt{slin}_{G,\vec{Z}^{(j)}}\left(7^{i2^{j-1}+2^{j-2}}\right) \leq \frac{1}{2} - \frac{\tilde{c}7^{2^{j-1}}}{3nc^2\left(7^{2^j}-1\right)} < r_j$$

for all i, so the second claim holds. □

B More algorithm classes for MaxQP

B.1 $\tilde{\epsilon}$-discretized functions for max-cut

The class of $\tilde{\epsilon}$-discretized rounding functions are a finite yet rich class of functions for the RPR^2 paradigm. They were introduced by O'Donnell and Wu as a tool for characterizing the *SDP gap curve* for the max-cut problem, which we define shortly in order to describe O'Donnell and Wu's guarantees for RPR^2 using $\tilde{\epsilon}$-discretized rounding functions. However, we first define the *max-cut SDP value* of a graph G to be

$$\mathrm{Sdp}(G) = \max_{g:V\to B_n}\left\{\sum_{(v_i,v_j)\in E} w_{ij}\left(\frac{1}{2} - \frac{1}{2}g(v_i)\cdot g(v_j)\right)\right\},$$

where $n = |V|$ and B_n denotes $\{x \in \mathbb{R}^n \mid ||x|| \leq 1\}$.

Now, suppose that a graph G has an SDP value $\mathrm{Sdp}(G) \geq c$. The SDP gap curve $\mathrm{Gap}_{SDP}(c)$ is a function that measures the smallest optimal max-cut value among all graphs such that $\mathrm{Sdp}(G) \geq c$. In other words, given that $\mathrm{Sdp}(G) \geq c$, we are guaranteed that the optimal max-cut value of G is at least $\mathrm{Gap}_{SDP}(c)$. Formally,

Definition 2. *For $\frac{1}{2} \leq s \leq c \leq 1$, we call the pair (c, s) an* SDP gap *if there exists a graph G with Sdp(G) $\geq c$ and Opt(G) $\leq s$. We define the* SDP gap curve *by*

$$Gap_{SDP}(c) = \inf\{s \mid (c, s) \text{ is an SDP gap}\}.$$

O'Donnell and Wu prove that if G is a graph such that $\mathrm{Sdp}(G) \geq c$, if one runs RPR^2 iteratively with all $\tilde{\epsilon}$-discretized rounding functions, then with high probability, at least one will result in a cut with value $\mathrm{Gap}_{SDP}(c) - \tilde{\epsilon}$. We now formally state the definition of an $\tilde{\epsilon}$-discretized rounding function as well as O'Donnell and Wu's algorithm guarantees.

Definition 3. *Given $\tilde{\epsilon} > 0$, let $\mathcal{I}_{\tilde{\epsilon}}$ denote the partition of $\mathbb{R} \setminus \{0\}$ into intervals,*

$$\mathcal{I}_{\tilde{\epsilon}} = \left\{\pm(-\infty, -B], \pm(-B, -B + \tilde{\epsilon}^2], \pm(-B + \tilde{\epsilon}^2, -B + 2\tilde{\epsilon}^2], \ldots, \pm(-2\tilde{\epsilon}^2, -\tilde{\epsilon}^2], \pm(-\tilde{\epsilon}^2, \tilde{\epsilon}^2)\right\},$$

where $B = B(\tilde{\epsilon})$ is the smallest integer multiple of $\tilde{\epsilon}^2$ exceeding $\sqrt{2\ln(1/\tilde{\epsilon})}$. We say that a function $r : \mathbb{R} \to [-1, 1]$ is $\tilde{\epsilon}$-discretized *if the following hold:*

1. *r is identically -1 on $(-\infty, -B]$, 0 at 0, and identically 1 on $[B, \infty)$.*
2. *r's values on the finite intervals in $\mathcal{I}_{\tilde{\epsilon}}$ are from the set $\tilde{\epsilon}\mathbb{Z} \cap (-1, 1)$.*

Note that there are $2^{O(1/\tilde{\epsilon}^2)}$ $\tilde{\epsilon}$-discretized functions.

Theorem 12 (Corollary 5.4 in [21])**.** *There is an algorithm which, given any graph G with Sdp(G) $\geq$ c and any $\tilde{\epsilon} > 0$, runs in time poly($|V|$)$2^{O(1/\tilde{\epsilon}^2)}$ and with high probability outputs a proper cut in G with value at least $Gap_{SDP}(c) - \tilde{\epsilon}$.*

Namely, the algorithm alluded to in Theorem 12 takes as input a graph, runs RPR^2 using all $\tilde{\epsilon}$-discretized rounding functions, and returns the cut with the maximum value. We define $\mathtt{cut}_{\tilde{\epsilon}}(G)$ to be the value of the resulting cut.

It is well-known that the pseudo-dimension of a finite function class $\mathcal{F}$ has pseudo-dimension $\log|\mathcal{F}|$. This immediately implies the following theorem.

Theorem 13. *Given an input sample of size $m = O\left(\frac{1}{\epsilon^2}\left(\frac{1}{\tilde{\epsilon}^2}\log\frac{1}{\epsilon} + \log\frac{1}{\delta}\right)\right)$ there exists an algorithm that (ϵ, δ)-learns the class of $\tilde{\epsilon}$-discretized rounding functions with respect to the cost function $-\mathtt{cut}_{\tilde{\epsilon}}$.*

B.2 Outward rotations

Next we study a class of "outward rotation" based algorithms proposed by Zwick [32]. For the max-cut problem, outward rotations are proven to work better than the random hyperplane technique of Goemans and Williamson [12] on graphs with "light" max-cuts where the max-cut does not constitute a large proportion of the edges. As stated earlier, though Feige and Langberg later showed that there exists a class of rounding functions for which RPR^2 becomes equivalent to outward rotations [11], we will analyze this class as it was originally presented by Zwick [32].

The class of outward rotation algorithms is characterized by an angle $\gamma \in [0, \pi/2]$ varying which results in a range of algorithms between the random hyperplane technique of Goemans and Williamson and the naive approach of outputting a random binary assignment [12]. Unlike RPR^2, the output here is a binary assignment. An outward rotation algorithm in essence extends the optimal SDP embedding in $\mathbb{R}^n$ to $\mathbb{R}^{2n}$. The way this is done can be understood as follows. The original embedding is first carried over to the first n co-ordinates of a $2n$-dimensional space while the remaining co-oordinates are set to zero. Suppose $u_{n+1}, u_{n+2}, \ldots, u_{2n}$ are the orthonormal vectors along each of the last n co-ordinates. Each embedding $\vec{u}_i$ is rotated "out" of the original space, towards u_{n+i} by an angle of γ. After performing these outward rotations, the new embedding is

projected onto a random hyperplane in $\mathbb{R}^{2n}$. The binary assignment is then defined deterministically based on the sign of the projections like in the GW algorithm [12]. Intuitively, the parameter γ determines how far the SDP embedding is used to determine the final projection of v_i as against an arbitrary value drawn from the normal distribution, which is contributed by u_i. We formally define the class below.

Algorithm 6 SDP rounding algorithm using γ-outward rotation

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

1: Solve the SDP (1) for the optimal embedding $U = (\vec{u}_1, \ldots, \vec{u}_n)$ of A.
2: Define a new embedding $\vec{u}'_i$ in $\mathbb{R}^{2n}$ such that the first n co-ordinates correspond to $\vec{u}_i \cos\gamma$ and the following n co-ordinates are set to 0 except the $(n+i)$th co-ordinate which is set to $\sin\gamma$.
3: Choose a random vector $\vec{Z} \in \mathbb{R}^{2n}$ according to the $2n$-dimensional Gaussian distribution.
4: For each decision variable x_i, assign $x_i = \operatorname{sgn}\left(\langle \vec{u}'_i, \vec{Z} \rangle\right)$.

Output: $x_1, \ldots, x_n$.

We will now set up notations similar to Section 3.1. Let $\texttt{owr}_\gamma\left(A, \vec{Z}\right)$ be the value of the binary assignment produced by projecting the SDP embedding of A onto $\vec{Z}$ after rotating it outwardly by γ. That is,

$$\texttt{owr}_\gamma\left(A, \vec{Z}\right) = \sum_{i,j} a_{ij} \operatorname{sgn}\left(\langle \vec{u}'_i, \vec{Z} \rangle\right) \operatorname{sgn}\left(\langle \vec{u}'_j, \vec{Z} \rangle\right).$$

We will use $\texttt{owr}_\gamma(A)$ to denote the expected value of $\texttt{owr}_\gamma\left(A, \vec{Z}\right)$ when $\vec{Z}$ is sampled from $\mathcal{Z}$, the $2n$-dimensional normal distribution. It can be easily seen that a fact very similar to Lemma 10 in Appendix A will apply here. Therefore, we will again use samples of the form $\left(A^{(i)}, \vec{Z}^{(i)}\right)$ for the pseudo-dimension analysis.

Let $\mathcal{H}_{\texttt{owr}} = \left\{\texttt{owr}_\gamma : \mathbb{A} \times \mathbb{R}^{2n} \to [0,1] \mid \gamma \in [0, \pi/2]\right\}$. We first prove in Section B.2.1 that the pseudo-dimension of $\mathcal{H}_{\texttt{owr}}$ is $O(\log n)$. Next, in Section B.2.2 we present an efficient learning algorithm.

B.2.1 The pseudo-dimension of the class of outward rotation based algorithms

We show an upper bound on the pseudo-dimension of the class of outward rotation based algorithms. In the following discussion, we will use the notation $\texttt{owr}_{A,\vec{Z}}(\gamma)$ in order to examine how the value changes as a function of γ for a fixed $(A, \vec{Z})$.

Theorem 14. *Pdim*$(\mathcal{H}_{\texttt{owr}}) = O(\log n)$.

Proof. Suppose $\mathcal{S} = \left\{\left(A^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}\right)\right\}$ is shatterable. This means that there exist m thresholds $\{r_1, \ldots, r_m\} \subset \mathbb{R}$ such that for each $T \subseteq [m]$, there exists a parameter γ_T such that $\texttt{owr}_{A^{(i)},\vec{Z}^{(i)}}(\gamma_T) > r_i$ if and only if $i \in T$.

We claim that for each sample $\left(A^{(i)}, \vec{Z}^{(i)}\right)$, $\texttt{owr}_{A^{(i)},\vec{Z}^{(i)}}(\gamma)$ is a piecewise constant function in γ with at most n values of γ at which it is discontinuous. If this were to be true, it means that there exists n values $a_1, a_2, \ldots a_n$ such that for γ within a given interval in $[0, a_1], (a_1, a_2], \ldots [a_n, 1]$ $\texttt{owr}_{A^{(i)},\vec{Z}^{(i)}}(\gamma)$ is identical and so is the label given by the witness r_i. Therefore, there are at most mn values of γ which define $mn+1$ intervals such that the labels given by the witnesses for the set of m samples is identical within each interval i.e., only at most $mn+1$ distinct labelings of $\mathcal{S}$ are

achievable for any choice of the witnesses. However, since $\mathcal{S}$ is shatterable, we need $2^m < mn + 1$. Thus, $s = O(\log n)$

Now, we only need prove our claim about $\texttt{owr}_{A,\vec{Z}}(\gamma)$ given A and $\gamma = \vec{Z}$. Observe that as γ increases, $\texttt{owr}_{A,\vec{Z}}(\gamma)$ will change only when $\text{sgn}\left(\langle \vec{u}_i', \vec{Z}\rangle\right)$ changes for some v_i. Now, note that $\langle \vec{u}_i', \vec{Z}\rangle = \langle \vec{u}_i, \vec{Z}_{[1,\ldots,n]}\rangle \cos\gamma + z_{n+i} \sin\gamma$ where $\vec{Z}_{[1,\ldots,n]}$ is the projection of $\vec{Z}$ over the first n co-ordindates. Clearly, $\langle \vec{u}_i', \vec{Z}\rangle$ is a monotone function in $\gamma \in [0, \pi/2]$ and attains zero at

$$\gamma = \tan^{-1}\left(-\frac{\langle \vec{u}_i, \vec{Z}_{[1,\ldots,n]}\rangle}{z_{n+i}}\right).$$

This implies that for each $i \in [n]$, $\text{sgn}\left(\langle \vec{u}_i', \vec{Z}\rangle\right)$ changes at most once within $[0, \pi/2]$. Therefore, $\texttt{owr}_{A,\vec{Z}}(\gamma)$ is a piecewise constant function with at most n discontinuities. □

B.2.2 A learning algorithm

We now present Algorithm 7 that efficiently learns the best value of γ for outward rotation with respect to samples drawn from $\mathcal{D} \times \mathcal{Z}$.

Algorithm 7 An algorithm for finding the empirical value maximizing γ

Input: Sample $\mathcal{S} = \left\{\left(A^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}\right)\right\}$

1: Solve for $\left\{U^{(1)}, \ldots, U^{(m)}\right\}$ the optimal SDP embeddings for $A^{(1)}, \ldots, A^{(m)}$, where $U^{(i)} = \left(\vec{u}_1^{(i)}, \ldots, \vec{u}_n^{(i)}\right)$.

2: Let $T = \left\{\gamma_1, \ldots, \gamma_{|T|}\right\}$ be the set of all values $\gamma \in [0, \pi/2]$ such that there exists a pair of indices $i \in [n], j \in [m]$ with $\tan^{-1}\left(-\frac{\langle \vec{u}^{(j)}, \vec{Z}^{(j)}_{[1,\ldots,n]}\rangle}{z^{(j)}_{n+i}}\right) = \gamma$.

3: Let $\hat{\gamma} = \underset{\gamma \in T \cup \{\pi/2\}}{\text{argmax}} \left\{\frac{1}{m}\sum_{i=1}^{m} \texttt{owr}_\gamma\left(A^{(i)}, \vec{Z}^{(i)}\right)\right\}$.

Output: $\hat{\gamma}$

Lemma 11. *Algorithm 7 produces the value $\hat{\gamma}$ which maximizes $\frac{1}{m}\sum_{i=1}^{m} \texttt{owr}_{A^{(i)},\vec{Z}^{(i)}}(\gamma)$ given the sample $\mathcal{S} = \left\{\left(A^{(1)}, \vec{Z}^{(1)}\right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}\right)\right\}$. Algorithm 7 has running time polynomial in m and n.*

Proof. Recall from the proof of Theorem 14 that T defines intervals over $[0, \pi/2]$ within each of which the behavior of any γ is constant across all samples in $\mathcal{S}$. Therefore, we only need to examine the performance of a single value of γ within each interval to exhaustively evaluate all possibilities, and single out the best one.

Also observe that since there are only $O(mn)$ values in T (in Step 2) and since computing the binary assignment on a set of m instances for a particular value of γ takes polynomial time in m and n, Step 3 should also take only polynomial time in m and n. □

Together with Theorem 2 and Theorem 14, Lemma 11 implies the following theorem.

Theorem 15. *Given an input sample of size $m = O\left(\frac{1}{\epsilon^2}\left(\log n \log\frac{1}{\epsilon} + \log\frac{1}{\delta}\right)\right)$ drawn from $(\mathcal{D}\times\mathcal{Z})^m$, Algorithm 7 (ϵ, δ)-learns the class of outward rotation algorithm with respect to the cost function $-\texttt{owr}_\gamma$.*

B.3 A general analysis of RPR^2 algorithms

In Sections 3.1 and B.2, we investigated two specific classes of RPR^2 algorithms. We will now present an analysis that can be applied to a wide range of classes of RPR^2 algorithms including that of s-linear functions and outward rotations. In particular, we show that for most classes of "sigmoid-like" rounding functions, the pseudo-dimension of RPR^2 is $\Theta(\log n)$.

As a first step towards this goal, we define what it means for a class of rounding functions to be "sigmoid-like" (Section B.3.1). Next, in order to both generalize and simplify the sample complexity analysis, we provide an alternative randomized procedure to RPR^2, which we call *Randomized Projection Randomized Thresholding* (RPRT), which produces a binary assignment rather than a fractional assignment (Section B.3.2). We prove that RPRT is equivalent to RPR^2 by showing that in expectation, the assignment produced by RPRT has the same value as the assignment produced by RPR^2 on an arbitrary problem instance. We work in the RPRT framework to bound the sample complexity required to (ϵ, δ)-learn the best rounding function in a fixed class for either RPR^2 or RPRT. We prove this by showing that the pseudo-dimension of RPRT with a sigmoid-like rounding function is $\Theta(\log n)$ (Section B.3.3). Finally, in Section B.3.4, we present an algorithm which (ϵ, δ)-learns the best rounding function in a fixed class.

B.3.1 A generic class of sigmoid-like rounding functions

We say that a class of functions $\mathcal{F}$ is "sigmoid-like" if the functions in the class are parameterized by a single constant $s \in \mathbb{R}^+$ and there exists a baseline function $f_1 \in \mathcal{F}$ such that for every function $f_s \in \mathcal{F}$, $f_s(x) := f_1(sx)$. Clearly, such a representation is rich enough to encompass classes of most common sigmoid-like functions including s-linear functions or any piece-wise linear function. In order for a class of sigmoid-like functions to qualify as SDP rounding functions, we additionally require that each function is non-decreasing and has a limit of -1 as x approaches $-\infty$, and has a limit of 1 as x approaches ∞. In particular,

Definition 4. *A class of real-valued functions $\mathcal{F} = \{f_s \mid s \in \mathbb{R}^+\}$ consists of sigmoid-like rounding functions if there exists a baseline function $f_1(x) : \mathbb{R} \to [-1, 1]$ such that*

1. *$f_1(x)$ is non-decreasing,*
2. *$\lim_{x\to\infty} f_1(x) = 1$ and $\lim_{x\to-\infty} f_1(x) = -1$,*

and for any function $f_s \in \mathcal{F}$, we can write $f_s(x) = f_1(sx)$.

Before we define RPRT, we make the following observation about sigmoid-like functions, as it will be useful for our algorithm design and sample complexity analysis. In particular, observe that for each s, $F_s(x) := \frac{f_s(x)+1}{2}$ is a cumulative density function associated with some probability distribution. This is because F_s is non-decreasing and its limits are 0 and 1 as x tends to $-\infty$ and $+\infty$, respectively. We denote the probability density function associated with F_s by p_s.

B.3.2 Randomized Projection Randomized Thresholding

RPRT differs from RPR^2 primarily in that it produces a binary assignment rather than a fractional assignment. In essence, RPRT simply samples a binary assignment from the distribution over binary assignments that RPR^2 effectively outputs. Like RPR^2, RPRT first projects the embedding of an IQP instance A onto a random vector $\vec{Z}$. However, RPRT then draws a random threshold for each variable and assigns the variable to either 1 or -1 depending on whether the directed distance of the projection (multiplied by $||\vec{Z}||$) is less than or greater than its threshold. The distribution from which these thresholds are picked are designed to mimic RPR^2 in expectation.

Algorithm 8 RPRT-based SDP rounding algorithm with rounding function $f_s \in \mathcal{F}$

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

1: Solve the SDP (1) for the optimal embedding $U = (\vec{u}_1, \ldots, \vec{u}_n)$ of A.
2: Choose a random vector $\vec{Z} \in \mathbb{R}^n$ according to the n-dimensional Gaussian distribution.
3: Draw $\vec{Q} = (q_1, q_2, \ldots, q_n) \sim (p_1)^n$ where p_1 is the probability density function corresponding to $f_1(x)$.
4: For each decision variable x_i, output the assignment $x_i = \operatorname{sgn}\left(q_i - s\langle \vec{u}_i, \vec{Z}\rangle\right)$.

Output: $x_1, \ldots, x_n$.

Now we show that the expected value of the binary assignment produced by RPRT is equal to that of RPR^2 for a given rounding function f_s. In the following discussion, we define $\texttt{rprt}_s\left(A, \vec{Z}, \vec{Q}\right)$ to be the deterministic value of the binary assignment produced by RPRT using the rounding function f_s, given the values of both $\vec{Z}$ and $\vec{Q}$. Similarly, we define $\texttt{rprt}_s\left(A, \vec{Z}\right)$ to be the expected value of the binary assignment produced by RPRT given the value of $\vec{Z}$. In other words, $\texttt{rprt}_s\left(A, \vec{Z}\right) = \mathbb{E}_{\vec{Q} \sim (p_1)^n}\left[\texttt{rprt}_s\left(A, \vec{Z}, \vec{Q}\right)\right]$. Finally, we define $\texttt{rprt}_s(A)$ to be the expected value of the binary assignment produced by RPRT and by $\texttt{rpr2}_s(A)$ the value of the binary assignment produced by RPR^2. In other words,

$$\texttt{rprt}_s(A) = \mathbb{E}_{\vec{Q} \sim (p_1)^n, \vec{Z} \sim \mathcal{Z}}\left[\texttt{rprt}_s\left(A, \vec{Z}, \vec{Q}\right)\right] \text{ and } \texttt{rpr2}_s(A) = \mathbb{E}_{\vec{Z} \sim \mathcal{Z}}\left[\texttt{rpr2}_s\left(A, \vec{Z}\right)\right]$$

Theorem 16. *Given a* MaxQP *problem with input matrix A and a rounding function f_s, the expected value of the binary assignment produced by RPRT equals the value of the fractional assignment produced by RPR^2, i.e.* $\texttt{rprt}_s(A) = \texttt{rpr2}_s(A)$.

Proof. Let x_i and x'_i denote the binary and fractional assignments produced by RPR^2 and RPRT for a given $\vec{Z}$ and $\vec{Q}$ respectively. We claim that for all i, $\mathbb{E}_{\vec{Q} \sim (p_1)^n}[x'_i] = x_i$. If we can prove this, then it implies that $\texttt{rprt}_s(G, Z) = \texttt{rpr2}_s(G, Z)$. To prove this, we will make use of the fact that given $\vec{Z}$, and hence given the projection $\langle \vec{u}_i, \vec{Z}\rangle$, x'_i and x'_j are independent random variables. Therefore, the expectation of $x'_i x'_j$ over all random draws of $\vec{Q}$ can be written as the product of their individual expected values.

$$\begin{aligned}
\texttt{rprt}_s(A, \vec{Z}) &= \mathop{\mathbb{E}}_{\vec{Q} \sim (p_1)^n}\left[\sum_{i,j} a_{ij} x'_i x'_j\right] \\
&= \sum_{i,j} a_{ij} \mathbb{E}_{\vec{Q} \sim (p_1)^n}[x'_i] \mathbb{E}_{\vec{Q} \sim (p_1)^n}[x'_j] \\
&= \sum_{i,j} a_{ij} x_i x_j = \texttt{rpr2}_s(A, \vec{Z}).
\end{aligned}$$

Once we have $\texttt{rprt}_s(A, \vec{Z}) = \texttt{rpr2}_s(A, \vec{Z})$, it is clear that $\texttt{rprt}_s(A) = \texttt{rpr2}_s(A)$ as $\vec{Z}$ is sampled from the same distribution in both the algorithms.

Now we only need to show that for a given $\vec{Z}$, $\mathbb{E}_{\vec{Q} \sim (p_1)^n}[x'_i] = x_i$ i.e., for a given $\vec{Z}$, for any variable, the expected binary assignment of the randomized thresholding step is equal to the fractional assignment of RPR^2 for that variable. Let us examine the expected binary assignment of RPRT for a particular variable x_i. The probability that x_i is assigned $+1$ is $Pr_{q_i \sim p_1}[q_i < s\langle \vec{u}_i, \vec{Z}\rangle]$.

Since f_1 is the cumulative density function of p_1, this is equal to $f_1(s\langle\vec{u}_i, \vec{Z}\rangle)$. However, due to the way we defined f_s, this value is in fact equal to $f_s(\langle\vec{u}_i, \vec{Z}\rangle)$. We complete the proof by noting that $f_s(\langle\vec{u}_i, \vec{Z}\rangle)$ is precisely the fractional assignment of RPR^2 for the choice of the rounding function f_s. □

Now, taking advantage of this equivalence between RPR^2 and RPRT for a given class of sigmoid-like functions, we will show that any learning algorithm that (ϵ, δ)-learns the best rounding functions with respect to RPRT also (ϵ, δ)-learns the best rounding functions with respect to RPR^2. Recall from Section 3.1 that in the analysis for pseudo-dimension, we incorporated $\vec{Z}$ in the sample set $\mathcal{S}$. In the same vein, we will now also incorporate $\vec{Q}$ in our sample set. In other words, we will provide to our learning algorithm the set of samples, $\left\{\left(A^{(1)}, \vec{Z}^{(1)}, \vec{Q}^{(1)}\right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}, \vec{Q}^{(m)}\right)\right\} \sim (\mathcal{D} \times \mathcal{Z} \times (p_1)^n)^m$. In order to prove that this indeed works, we now state Theorem 17 and Lemma 12 which parallel Theorem 2 and Lemma 10 respectively.

Theorem 17. *Suppose $\mathcal{F} = \{f_s | s > 0\}$ is a class of sigmoid-like rounding functions. Let $\mathcal{H}_{\texttt{rprt}} = \{\texttt{rprt}_s : \mathbb{A} \times \mathbb{R}^n \times \mathbb{R}^n \to [0,1] | s > 0\}$ and let $d_{\mathcal{H}_{\texttt{rprt}}}$ be the pseudo-dimension of $\mathcal{H}_{\texttt{rprt}}$. Suppose that $L_{\texttt{rprt}}$ is an algorithm that takes as input m samples $\left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right) \sim \mathcal{D} \times \mathcal{Z} \times (p_1)^n$, where $m = O\left(\frac{1}{\epsilon^2}(d_{\mathcal{H}_{\texttt{rprt}}} \log \frac{1}{\epsilon} + \log \frac{1}{\delta})\right)$, and returns the parameter $\hat{s}$ that maximizes*

$$\frac{1}{m}\sum_{i=1}^{m} \texttt{rprt}_s\left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right).$$

Then $L_{\texttt{rprt}}(\epsilon, \delta)$-learns the class of rounding functions $\mathcal{F}$ with respect to the cost function $-\texttt{rpr2}_s$ and is computationally efficient.

The proof for Theorem 17 follows from the following Lemma that is similar to Lemma 10.

Lemma 12. *Suppose that m is sufficiently large to ensure that with probability at least $1-\delta$ over a draw of m samples $\left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right) \sim \mathcal{D} \times \mathcal{Z} \times (p_1)^n$, for all functions $f_s \in \mathcal{F}$,*

$$\left|\frac{1}{m}\sum_{i=1}^{m} \texttt{rprt}_s\left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right) - \mathop{\mathbb{E}}_{(A,\vec{Z},\vec{Q}^{(i)}) \sim \mathcal{D}\times\mathcal{Z}\times(p_1)^n}\left[\texttt{rprt}_s\left(A, \vec{Z}, \vec{Q}\right)\right]\right| < \frac{\epsilon}{2}.$$

Then with probability at least $1-\delta$, if $\hat{s}$ maximizes $\left\{\frac{1}{m}\sum_{i=1}^{m} \texttt{rprt}_s\left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right)\right\}$, and s^* maximizes $\left\{\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_s(A)\right]\right\}$, then $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_{s^*}(A)\right] - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_{\hat{s}}(A)\right] < \epsilon$.

Proof of Lemma 12. Since $\mathcal{D} \times \mathcal{Z} \times (p_1)^n$ is a product distribution, we have that

$$\mathop{\mathbb{E}}_{(A,\vec{Z},\vec{Q})\sim\mathcal{D}\times\mathcal{Z}\times(p_1)^n}\left[\texttt{rprt}_s\left(A, \vec{Z}, \vec{Q}\right)\right] = \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rprt}_s(A)\right].$$

But from Theorem 16 we know that $\texttt{rprt}_s(A) = \texttt{rpr2}_s(A)$, which implies that

$$\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rprt}_s(A)\right] = \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_s(A)\right].$$

Hence, we can restate the assumption in the theorem statement as follows. With probability at least $1-\delta$, for all functions f_s,

$$\left|\frac{1}{m}\sum_{i=1}^{m} \texttt{rprt}_s\left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right) - \mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_s(A)\right]\right| < \frac{\epsilon}{2}.$$

Since this true for $\hat{s}$ and s^*, we can say that with probability $1-\delta$,

$$\left|\frac{1}{m}\sum_{i=1}^{m}\texttt{rprt}_{\hat{s}}\left(A^{(i)},\vec{Z}^{(i)},\vec{Q}^{(i)}\right)-\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_{\hat{s}}\left(A\right)\right]\right|<\frac{\epsilon}{2}$$

and

$$\left|\frac{1}{m}\sum_{i=1}^{m}\texttt{rprt}_{s^*}\left(A^{(i)},\vec{Z}^{(i)},\vec{Q}^{(i)}\right)-\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_{s^*}\left(A\right)\right]\right|<\frac{\epsilon}{2}.$$

Now, by definition of $\hat{s}$ we have that,

$$\frac{1}{m}\sum_{i=1}^{m}\texttt{rprt}_{\hat{s}}\left(A^{(i)},\vec{Z}^{(i)},\vec{Q}^{(i)}\right)\leq\frac{1}{m}\sum_{i=1}^{m}\texttt{rprt}_{s^*}\left(A^{(i)},\vec{Z}^{(i)},\vec{Q}^{(i)}\right).$$

Combining the previous three inequalities, we get that $\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_{s^*}\left(A\right)\right]-\mathop{\mathbb{E}}_{A\sim\mathcal{D}}\left[\texttt{rpr2}_{\hat{s}}\left(A\right)\right]<\epsilon$. □

B.3.3 The pseudo-dimension of a general class of RPRT algorithms

The main result of this section is a tight bound on the pseudo-dimension of any general class of RPRT algorithms.

Theorem 18. *Let* $\mathcal{H}_{\texttt{rprt}}=\{\texttt{rprt}_s:\mathbb{A}\times\mathbb{R}^n\times\mathbb{R}^n\to[0,1]\mid\ s>0\}$. *Pdim*$(\mathcal{H}_{\texttt{rprt}})=\Theta(\log n)$.

This follows from Lemma 13 and Lemma 14 where we provide matching upper and lower bounds on Pdim$(\mathcal{H}_{\texttt{rprt}})$.

Lemma 13. *Pdim*$(\mathcal{H}_{\texttt{rprt}})=O(\log n)$.

Proof. Suppose $\mathcal{S}=\left\{\left(A^{(1)},\vec{Z}^{(1)},\vec{Q}^{(1)}\right),\ldots,\left(A^{(m)},\vec{Z}^{(m)},\vec{Q}^{(m)}\right)\right\}$ is shatterable. Then there exist m thresholds $\{r_1,\ldots,r_m\}\subset\mathbb{R}$ such that for all $T\subseteq[m]$, there exists a parameters s_T such that $val'_{s_T}\left(A^{(i)},\vec{Z}^{(i)}\right)>r_i$ if and only if $i\in T$.

We first claim that for a given sample $\left(A^{(i)},\vec{Z}^{(i)},\vec{Q}^{(i)}\right)$ the domain of s i.e., $(0,\infty)$ can be partitioned into at most $n+1$ intervals $(0,a_1^{(i)}],(a_1^{(i)},a_2^{(i)}],\ldots(a_n^{(i)},\infty)$ such that the binary assignment produced by RPRT on the sample is identical across all parameter settings of s within a given interval. Once this is proved, we can then consider a partition of the domain of s into at most $mn+1$ intervals based on the points $\bigcup_{i=1}^{m}\{a_1^{(i)},\ldots,a_n^{(i)}\}$. Now, we know that for any value of s within a given interval of these $mn+1$ intervals, the labeling induced by the witnesses on $\mathcal{S}$ is the same. Therefore, there are at most $mn+1$ possible labelings that can be produced by any choice of the witnesses over any set of m samples. However, since we picked a shatterable instance, it must be that $2^m\leq mn+1$ i.e., $m=O(\log n)$.

Now to complete the proof, we need to examine the behavior of RPRT on a single sample $\left(A^{(i)},\vec{Z}^{(i)},\vec{Q}^{(i)}\right)$ for different configurations of s. Observe that as we increase s from 0 to ∞ keeping all else constant, we expect RPRT to produce a different binary assignment only when the assigned value changes for some vertex v_i. However, this happens only when $q_i=s\langle g(v_i),\vec{Z}\rangle$ i.e., when $s=q_i/\langle g(v_i),\vec{Z}\rangle$. Since there are only n vertices, and at most one value of s at which the algorithm changes behavior, we expect the algorithm to change its behavior at most n times as s increases. This proves our claim.

□

We now prove a lower bound on the pseudo-dimension of $\mathcal{H}_{\texttt{rprt}}$.

Lemma 14. *Pdim*$(\mathcal{H}_{\texttt{rprt}}) = \Omega(\log n)$.

Proof. In order to prove that the pseudo-dimension of $\mathcal{H}_{\texttt{rprt}}$ is at least $c \log n$ for some constant c, we must devise a set of samples $\mathcal{S} = \left\{ \left(G^{(1)}, \vec{Z}^{(1)}, \vec{Q}^{(i)} \right), \ldots, \left(G^{(m)}, \vec{Z}^{(m)}, \vec{Q}^{(i)} \right) \right\}$ of size $m = c \log n$ that can be shattered by $\mathcal{H}_{\texttt{rprt}}$. This means that we should be able to find m witnesses $r_1, \ldots, r_m$ and $2^m = n^c$ s values $H = \{s_1, \ldots, s_{n^c}\}$ such that for all $T \subseteq [m]$, there exists $s_T \in H$ such that if $i \in T$, then $\texttt{rprt}_{s_T}\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)} \right) > r_i$ and if $i \notin T$, then $\texttt{rprt}_{s_T}\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)} \right) \leq r_i$.

In our solution, we will use the same $\mathcal{S}$ as designed in the proof for Lemma 3. That is, all $G^{(i)}$ are identical and each consists of $n/4$ disjoint K_4 graphs over vertex sets $V_k = \{v_{4k+1}, v_{4k+2}, v_{4k+3}, v_{4k+4}\}$ for $k = 0, 1, \ldots \lfloor n/4 \rfloor - 4$. We will later pick $\vec{Z}^{(i)}$ and $\vec{Q}^{(i)}$ such that as the value of s increases, we change the cuts on $V_1, V_2, \ldots$ in that order and the change in the cuts is alternatingly better and worse. By appropriately choosing different values of $\vec{Z}^{(i)}$, we can carefully place the intervals in which these oscillations occur across all samples so as to be able to shatter a sample of size $\Omega(\log n)$.

In order to define $\vec{Z}^{(i)}$ and $\vec{Q}^{(i)}$, we make use of the following increasing sequence defined recursively: $c_1 = 8$ and $c_i = c_{i-1}^2$. Note that we can also write $c_i = 8^{2^{i-1}}$ but we will use the notation c_i for the sake of convenience. Let $a_{ik} = \frac{1}{c_i^k}$. We will now define $\vec{Z}^{(i)}$ in terms of a_{ik} as follows. For even k, we define

$$\left(\vec{Z}^{(i)}_{4k+1}, \vec{Z}^{(i)}_{4k+2}, \vec{Z}^{(i)}_{4k+3}, \vec{Z}^{(i)}_{4k+4} \right) = \left(\frac{1}{2} \cdot a_{ik}, a_{ik}, \frac{3}{2} \cdot a_{ik}, a_{ik} \right)$$

and for odd k we define,

$$\left(\vec{Z}^{(i)}_{4k+1}, \vec{Z}^{(i)}_{4k+2}, \vec{Z}^{(i)}_{4k+3}, \vec{Z}^{(i)}_{4k+4} \right) = (a_{ik}, -a_{ik}, -a_{ik}, a_{ik})$$

The rationale behind choosing the above value for $\vec{Z}^{(i)}$ will become evident as we proceed with the proof. First, by a simple calculation we can confirm that the directed distance of the projections of the vertices on $\vec{Z}^{(i)}$ lie in certain intervals as stated below.

For even k,

$$\langle g^{(i)}(v_j), \vec{Z}^{(i)} \rangle \in \begin{cases} \left[\frac{1}{2}a_{ik}, 2a_{ik}\right] & j = 4k+1, \\ \left[\frac{1}{2}a_{ik}, 2a_{ik}\right] & j = 4k+2, \\ \left[\frac{1}{2}a_{ik}, 2a_{ik}\right] & j = 4k+3, \\ \left[-2a_{ik}, -\frac{1}{2}a_{ik}\right] & j = 4k+4. \end{cases}$$

For odd k,

$$\langle g^{(i)}(v_j), \vec{Z}^{(i)} \rangle \in \begin{cases} \left[\frac{1}{2}a_{ik}, 2a_{ik}\right] & j = 4k+1, \\ \left[-2a_{ik}, -\frac{1}{2}a_{ik}\right] & j = 4k+2, \\ \left[-2a_{ik}, -\frac{1}{2}a_{ik}\right] & j = 4k+3, \\ \left[\frac{1}{2}a_{ik}, 2a_{ik}\right] & j = 4k+4. \end{cases}$$

As for the choice of $\vec{Q}^{(i)}$, we pick the same value for all i. In particular, we choose $\vec{Q}^{(i)}$ as follows. For even k,

$$\left(\vec{Q}^{(i)}_{4k+1}, \vec{Q}^{(i)}_{4k+2}, \vec{Q}^{(i)}_{4k+3}, \vec{Q}^{(i)}_{4k+4} \right) = (-1, +1, +1, -1)\,,$$

and for odd k,

$$\left(\vec{Q}^{(i)}_{4k+1}, \vec{Q}^{(i)}_{4k+2}, \vec{Q}^{(i)}_{4k+3}, \vec{Q}^{(i)}_{4k+4}\right) = (-1,-1,-1,+1)\,,$$

Again, our choice of $\vec{Q}^{(i)}$ will make sense as we proceed with the proof. We are now ready to analyze how $\texttt{rprt}_s\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right)$ varies as a function of s. We will show that $\texttt{rprt}_s\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right)$ oscillates above and below a threshold r_i, $\lfloor n/4 \rfloor/2$ times and these oscillations are spaced in such a manner that we can pick 2^m values of s that can shatter $\mathcal{S}$.

First, let us examine the values of s at which we expect the behavior of $\texttt{rprt}_s\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right)$ to change. We know that this can only consist of values of s equal to $1/\langle g^{(i)}(v_j), \vec{Z}^{(i)}\rangle$ for some j. Based on this fact, we observe that for the vertices in V_k the values of s at which the cut changes lie in $\left[\frac{1}{2}c_i^k, 2\cdot c_i^k\right]$. Thus, note that the intervals of s in which the cut changes for $V_1, V_2, \ldots$ are spaced far apart in increasing order because $2c_i^k < \frac{1}{2}c_i^{k+1}$ given $c_i > 4$.

Our analysis will henceforth focus on the values of s outside these intervals. We now claim that there exists values $b_{\max}$ and $b_{\min}$ such that the value of the cut is only one of these values in between the above intervals. In particular, the value is equal to $b_{\max}$ between $\left[0.5c_i^k, 2\cdot c_i^k\right]$ and $\left[0.5c_i^{k+1}, 2\cdot c_i^{k+1}\right]$ for odd k and is equal to $b_{\min}$ for even k. By simple calculation, it can be verified that the cut (S, S^c) assigned to V_k varies as follows. For even k,

$$S = \begin{cases} \{v_{4k+1}, v_{4k+4}\} & s < 0.5c_i^k, \\ \{v_{4k+1}, v_{4k+2}, v_{4k+3}\} & s > 2\cdot c_i^k, \end{cases}$$

and for odd k,

$$S = \begin{cases} \{v_{4k+1}, v_{4k+2}, v_{4k+3}\} & s < 0.5c_i^k, \\ \{v_{4k+1}, v_{4k+4}\} & s > 2\cdot c_i^k. \end{cases}$$

Now, when the cut is defined by $S = \{v_{4k+1}, v_{4k+4}\}$ is $2/3$. On the other hand when $S = \{v_{4k+1}, v_{4k+2}, v_{4k+3}\}$, the value of the cut is $1/2$. Thus we can make a crucial observation here: as s crosses the interval $\left[\frac{1}{2}c_i^k, 2\cdot c_i^k\right]$ for even k, the net value of the cut over the whole graph decreases, and for odd k it increases.

More precisely, we claim that $\texttt{rprt}_s\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right)$ as a function of s outside these intervals takes one of two values $b_{\min}$ and $b_{\max}$. To state these values explicitly, let us define a set of variables b_k such that for even k, $b_k = 2/3$ and for odd k, $b_k = 1/2$. Observe that when $s < 1/2$, the value of the cut is $\frac{1}{\lfloor n/4 \rfloor}\sum_{k=0}^{\lfloor n/4 \rfloor - 1} b_k$ as every odd graph contributes a value of $1/2$ and every even graph contributes a value of $2/3$ to the total cut. We will denote this value by $b_{\max}$. Now when $s \in (2, \frac{1}{2}c_i)$, the value then decreases to $\frac{1}{\lfloor n/4 \rfloor}\left(\sum_{k=0}^{\lfloor n/4 \rfloor - 1} b_k - 2/3 + 1/2\right)$ which we will call $b_{\min}$. We can extend this observation as follows to any interval $I_{i,k} = (2\cdot c_i^k, \frac{1}{2}c_i^{k+1})$

$$\texttt{rprt}_s\left(G^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)}\right) = \begin{cases} b_{\min} & s \in I_{i,k} \text{ for even } k \le \lfloor n/4 \rfloor - 1, \\ b_{\max} & s \in I_{i,k} \text{ for odd } k \le \lfloor n/4 \rfloor - 1, \end{cases}$$

Now we only need to choose r_i to be $(b_{\min} + b_{\max})/2$. To complete the proof, we need to show that the oscillations with respect to these witnesses for all the samples are spaced in a manner that we can pick 2^m different s covering all possible oscillations for $m = \Omega(\log n)$.

This becomes clear from two observations. First, $I_{i,2k} \cup I_{i,2k+1} \subset I_{i+1,k}$ because $c_i = 2c_{i-1}^2$. Secondly, values of s in $I_{i,2k}$ and $I_{i,2k+1}$ induce different labelings on the ith sample.

Now consider $m = \left\lfloor \frac{\log(\lfloor n/4 \rfloor)}{3} \right\rfloor$ samples. It turns out that $c_m^2/2 \leq c_1^{\lfloor n/4 \rfloor}$. That is, $I_{m,0} \cup I_{m,1}$ contains $I_{1,0} \cup \ldots I_{1,\lfloor n/4 \rfloor - 1}$. We then claim that the intervals $I_{1,k}$ each induce a different labeling on $\mathcal{S}$ with respect to the witnesses. To see why this is true, observe that since $I_{i,k} \subset I_{i+1,\lfloor k/2 \rfloor}$, the labeling induced by $s \in I_{1,k}$ is defined by which terms of the sequence $k_1 = k, k_{i+1} = \lfloor k_i/2 \rfloor$ are odd. This however is in fact the binary equivalent of k. Since the binary equivalent for a given k is unique, the labeling induced by $I_{1,k}$ is unique. Thus, we have $\Omega(\log n)$ samples that can be shattered.

□

B.3.4 A learning algorithm

We now present a learning algorithm(Algorithm 9) which (ϵ, δ)-learns the best rounding function with respect to $\mathcal{D}$ from a class of sigmoid-like rounding functions.

Algorithm 9 An algorithm for finding an empirical value maximizing rounding function f_s

Input: Sample $\mathcal{S} = \left\{ \left(A^{(1)}, \vec{Z}^{(1)}, \vec{Q}^{(1)} \right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}, \vec{Q}^{(m)} \right) \right\}$

1: Solve for $\left\{ X^{(1)}, \ldots, X^{(m)} \right\}$ the optimal SDP embeddings of $A^{(1)}, \ldots, A^{(m)}$, where $X^{(i)} = \left(\vec{x}_1^{(i)}, \ldots, \vec{x}_n^{(i)} \right)$.

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

3: Let $\hat{s} = \underset{s \in T \cup \{ s_{|T|} + 1 \}}{\operatorname{argmax}} \left\{ \frac{1}{m} \sum_{i=1}^{m} \texttt{rprt}_s \left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)} \right) \right\}$.

Output: $\hat{s}$

In particular, we prove the following guarantee regarding Algorithm 9's performance.

Lemma 15. *Algorithm 2 produces the value $\hat{s}$ which maximizes $\frac{1}{m} \sum_{i=1}^{m} \texttt{rpr2}_s \left(A^{(i)}, \vec{Z}^{(i)}, \vec{Q}^{(i)} \right)$ given the sample $\mathcal{S} = \left\{ \left(A^{(1)}, \vec{Z}^{(1)}, \vec{Q}^{(1)} \right), \ldots, \left(A^{(m)}, \vec{Z}^{(m)}, \vec{Q}^{(m)} \right) \right\}$. Algorithm 9 has running time polynomial in m and n.*

Proof. The correctness of the algorithm is evident from the proof for Lemma 13. The algorithm identifies the n values of s at which the behavior of RPRT changes and proceeds to exhaustively evaluate all the possible binary assignments which are only polynomially many. Therefore the algorithm successfully finds the best value of s in polynomial time. □

Corollary 1. *Given an input sample of size $m = O\left(\frac{1}{\epsilon^2} \left(\log n \log \frac{1}{\epsilon} + \log \frac{1}{\delta} \right) \right)$ drawn from $(\mathcal{D} \times \mathcal{Z} \times (p_1)^n)^m$, Algorithm 9 (ϵ, δ)-learns the class of rounding functions $\mathcal{F}$ with respect to the cost function $-\texttt{rpr2}_s$ and is computationally efficient.*

C Proofs from Section 4

Proof of Theorem 5. We give a general proof for all three values of b. We will point out a few places in the proof where the details for $b = 1, 2, 3$ are different, but the general structure of the argument is the same. For each value of b, we construct a single clustering instance $\mathcal{V} = (V, d)$ that has the desired property; the distribution $\mathcal{D}$ is merely the single clustering instance with probability 1.

Consider some permissible value of α, denoted α^*. Set $k = 4$ and $n = 210$. The clustering instance consists of two well-separated 'gadgets' of two clusters each. The class $\mathcal{A}_b$ results in different 2-clusterings of the first gadget depending on whether $\alpha \leq \alpha^*$ or not. Similarly, $\mathcal{A}_b$ results in different 2-clusterings of the second gadget depending on whether $\alpha \geq \alpha^*$ or not. By ensuring that for the first gadget $\alpha \leq \alpha^*$ results in the lowest cost 2-clustering, and for the second gadget $\alpha \geq \alpha^*$ results in the lowest cost 2-clustering, we ensure that $\alpha = \alpha^*$ is the optimal parameter overall.

The first gadget is as follows. We define five points a_1, b_1, c_1, x_1 and y_1. For the sake of convenience, we will group the remaining points into four sets A_1, B_1, X_1, and Y_1 each containing 25 points. We set the distances as follows: $d(a_1, b_1) = d(x_1, y_1) = 1$, $d(a_1, c_1) = 1.1$, and $d(b_1, c_1) = 1.2$. For $a \in A_1 \cup B_1$, $d(c_1, a) = 1.51$ and $d(a_1, a) = d(b_1, a) = 1.6$. For $x \in X_1 \cup Y_1$, $d(x_1, x) = d(y_1, x) = 1.6$. For $a \in A_1$, $b \in B_1$, $x \in X_1$, and $y \in Y_1$, $d(a, b) = d(x, y) = 1.6$. We also define special points $x_1^* \in X_1$ and $y_1^* \in Y_1$, which have the same distances as the rest of the points in X_1 and Y_1 respectively, except that $d(x_1, x_1^*) = 1.51$ and $d(y_1, y_1^*) = 1.51$. If two points p and q belong to the same set (A_1, B_1, X_1, or Y_1), then $d(p, q) = 1.5$.

The distances $d(x_1, c_1)$ and $d(y_1, c_1)$ are defined in terms of b and α^*, but they will always be between 1.1 and 1.2. For $b = 1$, we set $d(x_1, c_1) = d(y_1, c_1) = 1.2 - .1 \cdot \alpha^*$. For $b = 2$ and $b = 3$, $d(x_1, c_1) = d(y_1, c_1) = ((1.1^{\alpha^*} + 1.2^{\alpha^*})/2)^{\frac{1}{\alpha^*}}$.

So far, all of the distances we have defined are in $[1, 2]$, therefore they trivially satisfy the triangle inequality. We set all of the rest of the distances to be the maximum distances allowed under the triangle inequality. Therefore, the triangle inequality holds over the entire metric.

Now, let us analyze the merges caused by $\mathcal{A}_b(\alpha)$ for various values of α. Regardless of the values of α and b, since the distances between the first five points are the smallest, merges will occur over these initially. In particular, regardless of α and b, a_1 is merged with b_1, and x_1 with y_1. Next, by a simple calculation, if $\alpha \leq \alpha^*$, then c_1 merges with $a_1 \cup b_1$. If $\alpha > \alpha^*$, then c_1 merges with $x_1 \cup y_1$. Denote the set containing a_1 and b_1 by A_1', and denote the set containing x_1 and y_1 by X_1' (one of these sets will also contain c_1). Between A_1' and X_1', the minimum distance is $\geq 1.1 + 1.1 \geq 2.2$. All other subsequent merges (except for the very last merge) will involve all distances smaller than 2.2, so we never need to consider A_1' merging to X_1'.

The next smallest distances are all 1.5, so all points in A_1 will merge together, and similarly for B_1, X_1, and Y_1. At this point, the algorithm has created six sets: A_1', X_1', A_1, B_1, X_1, and Y_1. We claim that if $\alpha \leq \alpha^*$, A_1' will merge to A_1 and B_1, and X_1' will merge to X_1 and Y_1. This is because the maximum distance between sets in each of these merges is 1.6, whereas the minimum distance between $\{A_1', A_1, B_1\}$ and $\{X_1', X_1, Y_1\}$ is ≥ 2.2. Therefore, for all three values of b, the claim holds true.

Next we claim that the 2-clustering cost of gadget 1 will be lowest for clusters $A_1' \cup A_1 \cup B_1\}$ and $X_1' \cup X_1 \cup Y_1$ and when $c_1 \in A_1'$, i.e., when $\alpha \leq \alpha^*$. Clearly, since the distances within $A_1' \cup A_1 \cup B_1$ and $X_1' \cup X_1 \cup Y_1$ are much less than the distances across these sets, the best 2-clustering is $A_1' \cup A_1 \cup B_1$ and $X_1' \cup X_1 \cup Y_1$ (with all points at distance ≤ 1.6 to their center). We proved this will be a pruning of the tree when $\alpha \leq \alpha^*$. Therefore, we must argue the cost of this 2-clustering is lowest when $c_1 \in A_1'$. The idea is that c_1 can act as a very good center for $A_1' \cup A_1 \cup B_1$. But if $c_1 \in X_1'$, then the best center for $A_1' \cup A_1 \cup B_1$ will be an arbitrary point in $A_1 \cup B_1$. The cost in the first case is $1.51^p \cdot 50 + 1.1^p + 1.2^p$. The cost in the second case is $1.5^p \cdot 24 + 1.6^p \cdot 27$.

For $X_1' \cup X_1 \cup Y_1$, the center does not change depending on α (x_1^* and y_1^* tie for the best center), so the only difference in the cost is whether or not to include c_1. If $\alpha \leq \alpha^*$, then the cost is $1.5^p \cdot 24 + 1.51^p + 1.6^p \cdot 26$, otherwise the cost is $1.5^p \cdot 24 + 1.51^p + 1.6^p \cdot 26 + (1.6 + 1.2 - 0.1\alpha^*)^p$.

Putting it all together, if $\alpha \leq \alpha^*$, the cost is $1.51^p \cdot 50 + 1.1^p + 1.2^p + 1.5^p \cdot 24 + 1.51^p + 1.6^p \cdot 26$. Otherwise the cost is $1.5^p \cdot 48 + 1.51^p + 1.6^p \cdot 53 + (1.6 + 1.2 - 0.1\alpha^*)^p$. Subtracting off like terms, we conclude that the first case is always smaller because $1.51^p \cdot 49 + 1.1^p + 1.2^p < 1.5^p \cdot 24 + 1.6^p \cdot 26 + (1.6 + 1.2 - 0.1\alpha^*)^p$ for all $p \geq 1$.

Next, we will construct the second gadget arbitrarily far away from the first gadget. The second gadget is very similar to the first. There are points $a_2, b_2, c_2, x_2, y_2, x_2^*, y_2^*$ and sets to A_2, B_2, X_2, Y_2. $d(a_2, b_2) = d(x_2, y_2) = 1$, $d(x_2, c_2) = 1.1$, $d(y_2, c_2) = 1.2$, and for $b = 1$, $d(a_2, c_2) = d(b_2, c_2) = 1.2 - .1 \cdot \alpha^*$. For $b = 2$ or $b = 3$, $d(a_2, c_2) = d(b_2, c_2) = ((1.1^{\alpha^*} + 1.2^{\alpha^*})/2)^{\frac{1}{\alpha^*}}$. The rest of the distances are the same as in gadget 1. Then c_2 joins $\{a_2, b_2\}$ if $\alpha \geq \alpha^*$, not $\alpha \leq \alpha^*$. The rest of the argument is identical. So the conclusion we reach, is that the cost for the second gadget is much lower if $\alpha \geq \alpha^*$.

Therefore, the final cost of the 4-clustering is minimized when $\alpha = \alpha^*$, and the proof is complete. □

Now, we will analyze the pseudo-dimension of $\mathcal{H}_{\mathcal{A}_2,\Phi}$. Recall the argument in Lemma 5 for the upper bound of the pseudo-dimension of $\mathcal{H}_{\mathcal{A}_1,\Phi}$. Here, we relied on the linearity of $\mathcal{A}_1$'s merge equation to prove that for any eight points, there is exactly one value of α such that $\alpha d(p, q) + (1 - \alpha)d(p', q') = \alpha d(x, y) + (1 - \alpha)d(x', y')$. Now we will use a consequence of Rolle's Theorem (Theorem 19) that tells us that there is at most one value of α such that $((d(p, q))^\alpha + d(p', q')^\alpha)^{1/\alpha} = ((d(x, y))^\alpha + d(x', y')^\alpha)^{1/\alpha}$.

Theorem 19 (ex. [25])**.** *Let f be a polynomial-exponential sum of the form $f(x) = \sum_{i=1}^{N} a_i b_i^x$, where $b_i > 0$, $a_i \in \mathbb{R}$, and at least one a_i is non-zero. The number of roots of f is upper bounded by N.*

Lemma 16. *For any Φ, $Pdim(\mathcal{H}_{\mathcal{A}_2}, \Phi) = O(\log n)$.*

Proof. Suppose $\mathcal{S} = \{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\}$ is a set of clustering instances that can be shattered by $\mathcal{H}_{\mathcal{A}_2}$ using the witnesses $r_1, \ldots, r_s$. We must show that $m = O(\log n)$. For each value of $\alpha \in [0, 1]$, the algorithm $\mathcal{A}_2(\alpha)$ induces a binary labeling on each $\mathcal{V}^{(i)}$, based on whether $\Phi_{\mathcal{A}_2(\alpha)}(\mathcal{V}^{(i)}) \leq r_i$ or not.

Our argument will now parallel that of Lemma 5. Recall that our objective is to understand the behavior of $\mathcal{A}_2(\alpha)$ over m instances. In particular, as α varies over $\mathbb{R}$ we want to count the number of times the algorithm outputs a different merge tree on one of these instances. As we did in Lemma 5, for some instance $\mathcal{V}$ we will consider two pairs of sets A, B and X, Y that can be potentially merged. The decision to merge one pair before the other is determined by the sign of $d^\alpha(p, q) + d^\alpha(p', q') - d^\alpha(x, y) + d^\alpha(x', y')$. This expression, as before, is determined by a set of 8 points $p, p' \in A$, $q, q' \in B$, $x, x' \in X$ and $y, y' \in Y$ chosen independent of α.

Now, from Theorem 19, we have that the sign of the above expression as a function of α flips 4 times across $\mathbb{R}$. Since the expression is defined by exactly 8 points, iterating over all pairs (A, B) and (X, Y) we can list only $O(n^8)$ such unique expressions, each of which correspond to $O(1)$ values of α at which the corresponding decision flips. Thus, we can divide $\mathbb{R}$ into $O(n^8)$ intervals over each of which the output of $\Phi_{\mathcal{A}_2,\mathcal{V}}(\alpha)$ is fixed. In fact, we can divide $[0, 1]$ into $O(mn^8)$ intervals over each of which $\Phi_{\mathcal{A}_2,\mathcal{V}^{(i)}}(\alpha)$, and therefore the labeling induced by the witnesses, is fixed for all $i \in [m]$. This means that $\mathcal{H}_{\mathcal{A}_2}$ can achieve only $O(mn^8)$ binary labelings, which is at least 2^m since $\mathcal{S}$ is shatterable, so $m = O(\log n)$. □

Now we proceed to proving Lemma 6 which states that the pseudo-dimension of $\mathcal{H}_{\mathcal{A}_1}$ and $\mathcal{H}_{\mathcal{A}_1}$ is $\Omega(\log n)$. We first prove this lemma for the center-based objective cost denoted by $\Phi^{(p)}$ for $p \in [1, \infty) \cup \{\infty\}$. We later note how this can be extended cluster purity based cost.

We first prove the following useful statement which helps us construct general examples with desirable properties. In particular, the following lemma guarantees that given a sequence of values of α of size $O(n)$, it is possible to construct an instance $\mathcal{V}$ such that the cost of the output of $\mathcal{A}_1(\alpha)$ on $\mathcal{V}$ as a function of α, that is $\Phi^{(p)}_{\mathcal{A}_1,\mathcal{V}}(\alpha)$, oscillates above and below some threshold as α moves along the sequence of intervals (α_i, α_{i+1}). Given this powerful guarantee, we can then pick appropriate sequences of α and generate a sample set of $\Omega(\log n)$ instances that correspond to cost functions that oscillate in a manner that helps us pick $\Omega(n)$ values of s that shatters the samples.

Lemma 17. *Given $n \in \mathbb{N}$ and given a sequence of $n' \leq \lfloor n/7 \rfloor$ α's such that $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_{n'} < \alpha_{n'+1} = .7$, there exists a real valued witness $r > 0$ and a clustering instance $\mathcal{V} = (V, d)$, $|V| = n$, such that for $0 \leq i \leq n'/2 - 1$, $\Phi^{(p)}_{\mathcal{A}_1(\alpha)}(\mathcal{V}) < r$ for $\alpha \in (\alpha_{2i}, \alpha_{2i+1})$, and $\Phi^{(p)}_{\mathcal{A}_1(\alpha)}(\mathcal{V}) > r$ for $\alpha \in (\alpha_{2i+1}, \alpha_{2i+2})$, for $k = 2$.*

Proof. Here is a high level description of our construction $\mathcal{V} = (V, d)$. There will be two "main" points, a and a' in V. The rest of the points are defined in groups of 6: $(x_i, y_i, z_i, x'_i, y'_i, z'_i)$, for $1 \leq i \leq (n-2)/6$. We will define the distances d such that initially for all $\mathcal{A}_b(\alpha)$, x_i merges to y_i to form the set A_i, and x'_i merges to y'_i to form the set A'_i. As for (z_i, z'_i), depending on whether $\alpha < \alpha_i$ or not, $\mathcal{A}_b(\alpha)$ merges the points z_i and z'_i with the sets A_i and A'_i respectively or vice versa. Thus, there are $(n-2)/6$ values of α such that $\mathcal{A}_b(\alpha)$ has a unique behavior in the merge step. Finally, for all α, sets A_i merge to $\{a\}$, and sets A'_i merge to $\{a'\}$. Let $A = \{a\} \cup \bigcup_i A_i$ and $A' = \{a'\} \cup \bigcup_i A'_i$. Thus, there will be $(n-2)/6$ intervals (α_i, α_{i+1}) for which $\mathcal{A}_b(\alpha)$ returns a unique partition $\{A, A'\}$. By carefully setting the distances, we cause the cost $\Phi^{(p)}(\{A, A'\})$ to oscillate above and below a specified value r along these intervals.

First of all, in order for d to be a metric, we set all distances in $[1, 2]$ so that the triangle inequality is trivially satisfied. In particular, the following are the distances of the pairs of points within each group for $1 \leq i \leq (n-2)/6$.

$$\begin{aligned}
&d(x_i, y_i) = d(x'_i, y'_i) = 1,\\
&d(x_i, z_i) = 1.3,\ d(y_i, z_i) = 1.4,\\
&d(x'_i, z_i) = d(y'_i, z_i) = 1.4 - .1 \cdot \alpha_i,\\
&d(x_i, x'_i) = d(y_i, y'_i) = 2.
\end{aligned}$$

We set the distances to z'_i as follows (see Figure 9).

$$\begin{aligned}
&d(x_i, z'_i) = d(y_i, z'_i) = d(x'_i, z'_i) = d(y'_i, z'_i) = 1.41,\\
&d(z_i, z'_i) = 2.
\end{aligned}$$

Then the first merges will be x_i to y_i and x'_i to y'_i, no matter what α is set to be (when each point is a singleton set, each pair of points with the minimum distance in the metric will merge). Next, z_i will either merge to A_i or A'_i based on the following equation:

$$\begin{aligned}
& \alpha \cdot 1.3 + (1-\alpha) \cdot 1.4 \lessgtr \alpha \cdot (1.4 - .1 \cdot \alpha_i) + (1-\alpha)(1.4 - .1 \cdot \alpha_i)\\
\implies \quad & 1.4 - .1 \cdot \alpha \lessgtr 1.4 - .1 \cdot \alpha_i\\
\implies \quad & \alpha_i \lessgtr \alpha
\end{aligned}$$

If $\alpha < \alpha_i$, then z_i merges to A'_i, otherwise it will merge to A_i. Notice that the merge expression for A_i to A'_i could be as small as $\alpha \cdot 1.3 + (1-\alpha) \cdot 2 = 2 - .7 \cdot \alpha$, but we do not want this

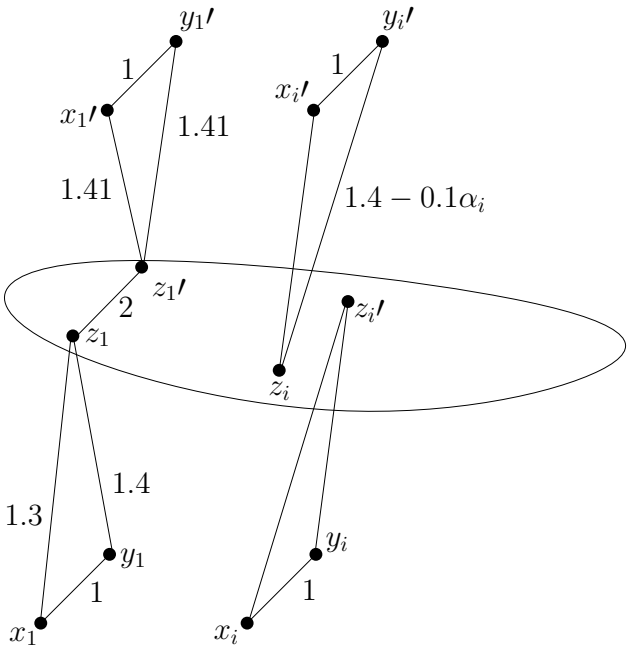

Figure 9: The clustering instance used in Lemma 17

merge to occur. If we ensure all subsequent merges have maximum distance less than 1.5, then A_i will not merge to A_i' (until A and A' merge in the very final step) as long as $\alpha < .7$, because $\alpha \cdot 1.5 + (1-\alpha) \cdot 1.5 = 1.5 < 2 - .7 \cdot .7$.

These distances ensure z_i' merges after z_i regardless of the value of α, since z_i is closer than z_i' to x_i, x_i', y_i, and y_i'. Furthermore, z_i' will merge to the opposite set of z_i, since we set $d(z_i, z_i') = 2$. The merge expression for z_i' to merge to the opposite set is $\alpha \cdot 1.41 + (1-\alpha) \cdot 1.41$, while the merge expression to the same set is $\geq \alpha \cdot 1.41 + (1-\alpha) \cdot 2$.

Now we set the distances to a and a' as follows.

$$d(a, x_i) = d(a, y_i) = d(a', x_i') = d(a', y_i') = 1.42,$$
$$d(a, x_i') = d(a, y_i') = d(a', x_i) = d(a', y_i') = 2.$$

We also set all distances between A_i and A_j' to be 2, for all i and j, and all distances between A_i and A_j to be 1.5, for all $i \neq j$. We will set the distances from a and a' to z_i and z_i' later, but they will all fall between 1.45 and 1.5. By construction, every set A_i will merge to the current superset containing $\{a\}$, because the merge expression is $\alpha \cdot 1.42 + (1-\alpha)1.5$, and *any other possible merge* will have value $\geq \alpha \cdot 1.3 + (1-\alpha) \cdot 2$, which is larger for $\alpha < .7$. Similarly, all A_i' sets will merge to $\{a'\}$.

Therefore, the final two sets in the linkage tree are A and A'. Given $1 \leq i \leq (n-2)/6$, by construction, for $\alpha \in (\alpha_i, \alpha_{i+1})$, $\{z_1, \ldots, z_i, z_{i+1}', \ldots z_{(n-2)/6}'\} \subseteq A$ and $\{z_1', \ldots, z_i', z_{i+1}, \ldots z_{(n-2)/6}\} \subseteq A'$.

Finally, we set the distances between a, a', z_i, and z_i' to ensure the cost function oscillates.

$$\begin{aligned} \forall i, \quad & d(a, z_i') = d(a', z_i) = 1.46 \\ \forall 1 \leq j \leq (n-2)/12, \quad & d(a, z_{2j-1}) = d(a', z_{2j}') = 1.47, \\ & \text{and } d(a, z_{2j}) = d(a', z_{2j+1}') = (2 \cdot 1.46^p - 1.47^p)^{1/p}. \end{aligned}$$

Now we calculate the 2-clustering cost of (A, A') for α's in different ranges. Regardless of α, all partitions will pay $\sum_i (d(a,x_i)^p + d(a,y_i)^p + d(a',x_i')^p + d(a',y_i')^p) = (n-2)/6 \cdot (4 \cdot 1.42^p)$,but the distances for z_i and z_i' differ. For $\alpha \in (\alpha_0, \alpha_1)$, all of the z's pay 1.46^p, so the cost is $(n-2)/6 \cdot (4 \cdot 1.42^p + 2 \cdot 1.46^p)$. Denote this value by r_{low}.

When $\alpha \in (\alpha_1, \alpha_2)$, the only values that change are z_1 and z_1', which adds $d(a,z_1) + d(a',z_1') - d(a,z_1') - d(a',z_1) = 2 \cdot (1.47^p - 1.46^p) > 0$ to the cost (the inequality is always true for $p \in [1, \infty]$). Denote $r_{low} + 2 \cdot (1.47^p - 1.46^p)$ by r_{high}. When $\alpha \in (\alpha_2, \alpha_3)$, the values of z_2 and z_2' change, and the cost changes by $d(a,z_2) + d(a',z_2') - d(a,z_2') - d(a',z_2) = 2 \cdot ((2 \cdot 1.46^p - 1.47^p) - 1.46^p) = -2 \cdot (1.47^p - 1.46^p)$, decreasing it back to r_{low}.

In general, the cost for $\alpha \in (\alpha_i, \alpha_{i+1})$ is $r_{low} + \sum_{1 \le j \le i} (-1)^{i+1} \cdot 2(1.47^p - 1.46^p) = r_{low} + (1.47^p - 1.46^p) + (-1)^{i+1} \cdot (1.47^p - 1.46^p)$. If $\alpha \in (\alpha_{2j}, \alpha_{2j+1})$, then the cost is r_{low}, and if $\alpha \in (\alpha_{2j+1}, \alpha_{2j+2})$, the cost is r_{high}. We set $r = (r_{low} + r_{high})/2$, and conclude that the cost function oscillates above and below r as specified in the lemma statement.

The pruning step will clearly pick (A, A') as the optimal clustering, since the only centers with more than 3 points at distance < 1.5 are a and a', and (A, A') are the clusters in which the most points can have a and a' as centers. This argument proved the case where $n' = (n-2)/6$. If $n' < (n-2)/6$, then we set $d(a, z_i) = d(a', z_i') = 1.46$ for all $i > n'$, which ensures the cost function oscillates exactly n' times. This completes the proof. □

Now we can prove Lemma 6.

Proof of Lemma 6. First, we prove the claim for $\mathcal{H}_{\mathcal{A}_1, \Phi^{(p)}}$ by constructing a set of samples $\mathcal{S} = \{\mathcal{V}^{(1)}, \ldots, \mathcal{V}^{(m)}\}$ where $m = \log n - 3$ that can be shattered by $\mathcal{H}_{\mathcal{A}_1, \Phi^{(p)}}$. That is, we should be able to choose $2^m = n/8$ different values of α such that there exists some witnesses $r_1, \ldots, r_m$ with respect to which $\Phi^{(p)}_{\mathcal{A}_1(\alpha)}(\cdot)$ induces all possible labelings on $\mathcal{S}$.

Choose a sequence of 2^m distinct α's arbitrarily in the range $(0, .7)$. We will index the terms of this sequence using the notation $\alpha_{\mathbf{x}}$ for all $\mathbf{x} \in \{0,1\}^m$, such that $\alpha_{\mathbf{x}} < \alpha_{\mathbf{y}}$ iff $\mathbf{x}_1\mathbf{x}_2 \ldots \mathbf{x}_m < \mathbf{y}_1\mathbf{y}_2\mathbf{y}_m$. Then the α's satisfy

$$0 < \alpha_{[0\ \ldots\ 0\ 0]} < \alpha_{[0\ \ldots\ 0\ 1]} < \alpha_{[0\ \ldots\ 1\ 0]} < \cdots < \alpha_{[1\ \ldots\ 1\ 1]} < .7.$$

Given $\mathbf{x}$, denote by $n(\mathbf{x})$ the vector corresponding to $\mathbf{x}_1\mathbf{x}_2 \ldots \mathbf{x}_s + 1$, therefore, $\alpha_{n(\mathbf{x})}$ is the smallest α greater than $\alpha_{\mathbf{x}}$.

Now, the crucial step is that we will use Lemma 17 to define our examples $\mathcal{V}^{(1)}, \ldots V^{(m)}$ and witnesses $r_1, \ldots r_m$ so that when $\alpha \in (\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$ the labeling induced by the witnesses on $\mathcal{S}$ corresponds to the vector $\mathbf{x}$. This means that for $\alpha \in (\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$ the cost function $\Phi^{(p)}_{\mathcal{A}_1(\alpha)}(\mathcal{V}^{(i)})$ must be greater than r_i if the ith term in $\mathbf{x}$ is 1, and less than r_i otherwise. Since there are only $2^m = \frac{n}{8}$ $\mathbf{x}$'s, it implies that for any sample $\mathcal{V}^{(i)}$ there at most $n/8$ values of α at which we want its cost to flip above/below r_i. We can we can accomplish this using Lemma 17 by choosing $\alpha_{\mathbf{x}}$'s for which $\mathcal{V}^{(i)}$ is supposed to switch labels. In this manner, we pick each $\mathcal{V}^{(i)}i$ and r_i thus creating a sample of size $\Omega(\log n)$ that is shattered by $\mathcal{H}_{\mathcal{A}_1, \Phi^{(p)}}$.

It is straightforward to modify this proof to work for $\mathcal{A}_2$. by applying Lemma 17 adapted for $\mathcal{A}_2$ holds. In particular, given $n \in \mathbb{N}$, $p \in [1, \infty]$, and given a set of $n' \le \lfloor n/7 \rfloor$ α's such that $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_{n'} < \alpha_{n'+1} = .7$, then there exists a real valued witness $r > 0$ and a clustering instance (V, d), $|V| = n$, which oscillates up to $n/7$ times as α increases in $(1, 3)$. The only major change is to set

$$d(x_i', z_i) = d(y_i', z_i) = ((1.3_i^\alpha + 1.4_i^\alpha)/2)^{\frac{1}{\alpha_i}}.$$

Then we may use the same proof as $\mathcal{A}_1$ to show that $\text{Pdim}(\mathcal{H}_{\mathcal{A}_2}, \Phi^{(p)}) = \Omega(\log n)$. □

Note C.1. *Lemma 6 assumes that the pruning step fixes a partition, and then the optimal centers can be chosen for each cluster in the partition, but points may not switch clusters even if they are closer to the center in another cluster. This is desirable, for instance, in applications which much have a balanced partition.*

If it is desired that the pruning step only outputs the optimal centers, and then the clusters are determined by the Voronoi partition of the centers, we modify the proof as follows. We introduce $2n'$ more points into the clustering instance: $c_1, \ldots, c_{n'}$, and $c'_1, \ldots, c'_{n'}$. Each c_i will merge to cluster A, and each c'_i will merge to cluster A'. We set the distances so that c_i and c'_i will be the best centers for A and A' when $\alpha \in (\alpha_i, \alpha_{i+1})$. The distances are also set up so that the cost of the Voronoi tiling induced by c_{2i} and c'_{2i} is r_{low}, and the cost for c_{2i+1} and c'_{2i+1} is r_{high}. This is sufficient for the argument to go through.

Furthermore, the lower bound holds even if the cost function is the symmetric distance to the ground truth clustering. For this proof, let $A \cup \bigcup_i \{z_{2i}, z'_{2i+1}\}$ and $A' \cup \bigcup_i \{z_{2i+1}, z'_{2i}\}$ be the ground truth clustering. Then in each interval as α increases, the cost function switches between having $(n-2)/3$ errors and having $(n-2)/3 - 2$ errors.

In this section, we give a proof of Lemma 8. We start with a helper lemma.

Lemma 18. *Given n, and setting $N = \lfloor (n-8)/2 \rfloor$, then there exists a clustering instance $\mathcal{V} = (V, d)$ of size $|V| = n$ and a set of $2^N + 2$ α's for which α-linkage creates a unique merge tree.*

Proof. Here is the outline of our construction. At the start, two specific pairs of points will always merge first: p_a merges with q_a, and p_b merges with q_b. The sets $\{p_a, q_a\}$ and $\{p_b, q_b\}$ will stay separated until the last few merge operations. Throughout the analysis, at any point in the merging procedure, we denote the current superset containing $\{p_a, q_a\}$ by A, and we similarly denote the superset of $\{p_b, q_b\}$ by B. The next points to merge will come in pairs, (p_i, q_i) for $1 \le i \le N$. We construct the distances so that p_i and q_i will always merge before p_j and q_j, for $i < j$. Furthermore, for all i, $\{p_i\}$ will first merge to A or B, and then $\{q_i\}$ will merge to the opposite set as p_i. Let these two merges be called 'round i', for $1 \le i \le N$. Finally, there will be a set C_A of size $N + 2$ which merges together and then merges to A, and similarly a set C_B which merges to B.

Thus, in our construction, the only freedom is whether p_i merges to A or to B, for all i. This is 2^N combinations total. The crux of the proof is to show there exists an α for each of these behaviors.

We attack the problem as follows. In round 1, the following equation specifies whether p_1 merges to A or B:

$$\frac{1}{2}(d(p_a, p_1)^\alpha + d(q_a, p_1)^\alpha) \lesseqgtr \frac{1}{2}(d(p_b, p_1)^\alpha + d(q_b, p_1)^\alpha)$$

If the LHS is smaller, then p_1 merges to A, otherwise B. By carefully setting the distances, we will ensure there exists a value α' which is the only solution to the equation in the range $(1, 3)$. Then p_1 merges to A for all $\alpha \in (1, \alpha')$, and B for all $\alpha \in (\alpha', 3)$. For now, assume it is easy to force q_1 to merge to the opposite set.

In round 2, there are two equations:

$$\frac{1}{3}(d(p_a, p_2)^\alpha + d(q_a, p_2)^\alpha + d(p_1, p_2)^\alpha) \lesseqgtr \frac{1}{3}(d(p_b, p_2)^\alpha + d(q_b, p_2)^\alpha + d(q_1, p_2)^\alpha),$$
$$\frac{1}{3}(d(p_a, p_2)^\alpha + d(q_a, p_2)^\alpha + d(q_1, p_2)^\alpha) \lesseqgtr \frac{1}{3}(d(p_b, p_2)^\alpha + d(q_b, p_2)^\alpha + d(p_1, p_2)^\alpha).$$

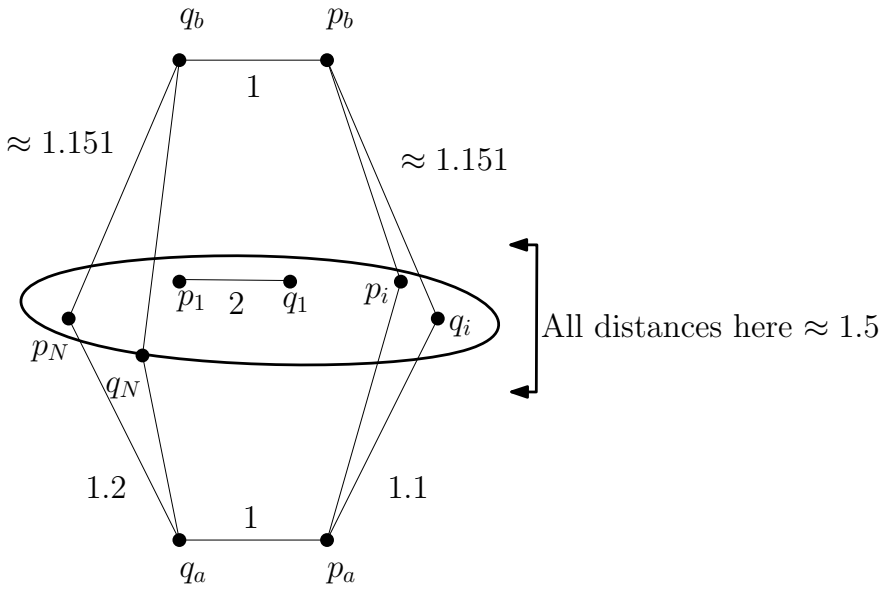

Figure 10: The clustering instance used in Lemma 8

The first equation specifies where p_2 merges in the case when $p_1 \in A$, and the second equation is the case when $p_1 \in B$. So we must ensure there exists a specific $\alpha_{[-1]} \in (1, \alpha')$ which solves equation 1, and $\alpha_{[1]} \in (\alpha', 3)$ which solves equation 2, and these are the only solutions in the corresponding intervals.

In general, round i has 2^{i-1} equations corresponding to the 2^{i-1} possible states for the partially constructed tree. For each state, there is a specific α interval which will cause the algorithm to reach that state. We must ensure that the equation has exactly one solution in that interval. If we achieve this simultaneously for every equation, then the next round will have $2 \cdot 2^{i-1}$ states. See Figure 10 for a schematic of the clustering instance.

For $1 \le i \le N$, given $\mathbf{x} \in \{-1, 1\}^{i-1}$, let $E_{\mathbf{x}}$ denote the equation in round i which determines where p_i merges, in the case where for all $1 \le j < i$, p_j merged to A if $x_j = -1$, or B if $x_j = 1$ (and let E' denote the single equation for round 1). Let $\alpha_{\mathbf{x}} \in (1, 3)$ denote the solution to $E_{\mathbf{x}} = 0$. Then we need to show the α's are well-defined and follow a specific ordering, shown in Figure 11. This ordering is completely specified by two conditions: *(1)* $\alpha_{[\mathbf{x}\ -1]} < \alpha_{[\mathbf{x}]} < \alpha_{[\mathbf{x}\ 1]}$ and *(2)* $\alpha_{[\mathbf{x}\ -1\ \mathbf{y}]} < \alpha_{[\mathbf{x}\ 1\ \mathbf{z}]}$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \bigcup_{i<N} \{-1, 1\}^i$ and $|\mathbf{y}| = |\mathbf{z}|$.

Now we show how to set up the distances to achieve all of these properties. To enhance readability, we start with an example for $n = 10$, and then move to the general construction.

We give the construction round by round. All distances are in $[1, 2]$ which ensures the triangle inequality is always satisfied. Set $d(p_a, q_a) = d(p_b, q_b) = 1$, and set all pairwise distances between $\{p_a, q_a\}$ and $\{p_b, q_b\}$ to 2. We also set $d(p_i, q_i) = 2$ for all $i \ne j$.

Here are the distances for the first round.

$$d(p_a, p_1) = d(p_a, q_1) = 1.1,\ d(q_a, p_1) = d(q_a, q_1) = 1.2,$$
$$d(p_b, p_1) = d(p_b, q_1) = d(q_b, p_1) = d(q_b, q_1) = \sqrt{(1.1^2 + 1.2^2)/2} \approx 1.151.$$

Say we break ties by lexicographic order. Then p_1 will merge first, and E' is the following.

$$\frac{1}{2}(1.1^\alpha + 1.2^\alpha) \lesseqgtr \frac{1}{2}(1.151^\alpha + 1.151^\alpha).$$

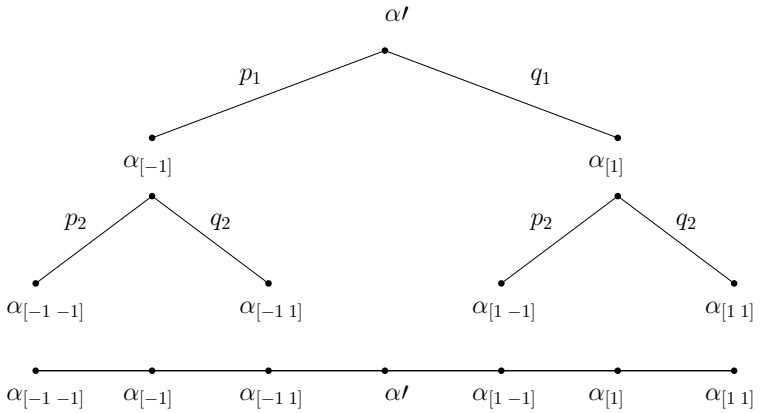

Figure 11: A schematic for the α intervals. Each edge denotes whether to merge p_i to A or q_i to A.

The unique solution in $(1, 3)$ is $\alpha' = 2$. p_1 merges to A for $\alpha \in (1, 2)$, and B for $\alpha \in (2, 3)$.

The merge equations for q_1 are

$$\frac{1}{3}(1.1^\alpha + 1.2^\alpha + 2^\alpha) \lessgtr \frac{1}{2}(1.151^\alpha + 1.151^\alpha) \text{ if } p_1 \in A, \text{ or}$$
$$\frac{1}{2}(1.1^\alpha + 1.2^\alpha) \lessgtr \frac{1}{3}(1.151^\alpha + 1.151^\alpha + 2^\alpha) \text{ otherwise.}$$

As long as $\alpha \in (1, 3)$, q_1 will merge to the opposite cluster as p_1. This is because we set $d(p_1, q_1)$ to be significantly larger than the other relevant distances.

We set the round two distances as follows.

$$d(p_a, p_2) = d(p_a, q_2) = 1.1, \quad d(q_a, p_2) = d(q_a, q_2) = 1.2,$$
$$d(p_b, p_2) = d(p_b, q_2) = d(q_b, p_2) = d(q_b, q_2) = \sqrt{(1.1^2 + 1.2^2)/2} \approx 1.151,$$
$$d(p_1, p_2) = d(p_1, q_2) = 1.5 + 10^{-4}, \quad d(q_1, p_2) = d(q_1, q_2) = 1.5 - 10^{-4}.$$

Note the distances from $\{p_a, q_a, p_b, q_b\}$ are the same to p_1 as to p_2. Since we break ties lexicographically, this ensures p_2 and q_2 merge in round 2. Alternatively, we can add tiny perturbations to the distances such that they do not affect our analysis, but ensure the correct merge orders regardless of the tiebreaking rule. The values were picked so that the p_a, q_a, p_b, q_b distances will have the most influence in the merge equations. If it were just these four distances, then the value of α at equality would be 2 again. But in this round, the p_1 and q_1 distances show up in the equation. We set $d(p_1, p_2), d(q_1, p_2) \approx 1.5$ but with small 'offsets' so that the values of α are on either side of $\alpha' = 2$. Equations $E_{[-1]}$ and $E_{[1]}$ are the following.

$$\frac{1}{3}(1.1^\alpha + 1.2^\alpha + 1.5001^\alpha = \frac{1}{3}(2 \cdot 1.151^\alpha + 1.4999^\alpha),$$
$$\frac{1}{3}(1.1^\alpha + 1.2^\alpha + 1.4999^\alpha) = \frac{1}{3}(2 \cdot 1.151^\alpha + 1.5001^\alpha).$$

Then $\alpha_{[-1]} \approx 1.884$ and $\alpha_{[1]} \approx 2.124$.

As in the previous round, it is straightforward to check all four merge equations for q_2 send q_2 to the opposite cluster as p_2, as long as $\alpha < 3$. So far, our four intervals of α which lead to distinct behavior are $(1, 1.884)$, $(1.884, 2)$, $(2, 2.124)$, and $(2.124, 3)$.

Now we specify the distances for the third round, the final round in our example. Again, we set the distances from $\{p_a, q_a, p_b, q_b\}$ the same way as in previous rounds. The new distances are as follows.

$$d(p_1,p_3) = d(p_1,q_3) = 1.5 + 10^{-4}, \quad d(q_1,p_3) = d(q_1,q_3) = 1.5 + 10^{-4},$$
$$d(p_2,p_3) = d(p_2,q_3) = 1.5 + 10^{-6}, \quad d(q_2,p_3) = d(q_2,q_3) = 1.5 - 10^{-6}.$$

Again, if it were just the first four points, the value of α would be 2. The distances for $\{p_1, q_1\}$ differ by a small offset, and the distances for $\{p_2, q_2\}$ differ by an even smaller offset, which causes the latter distances to have less influence in the merge equations. This forces the α's into the correct intervals. In general, the offset value will decrease in higher and higher rounds.

The equations $E_{[-1\ -1]}$, $E_{[-1\ 1]}$, $E_{[1\ -1]}$, and $E_{[1\ 1]}$ are as follows:

$$\frac{1}{4}(1.1^\alpha + 1.2^\alpha + (1.5+10^{-4})^\alpha + (1.5+10^{-6})^\alpha) = \frac{1}{4}(2\cdot 1.151^\alpha + (1.5-10^{-4})^\alpha + (1.5-10^{-6})^\alpha),$$
$$\frac{1}{4}(1.1^\alpha + 1.2^\alpha + (1.5+10^{-4})^\alpha + (1.5-10^{-6})^\alpha) = \frac{1}{4}(2\cdot 1.151^\alpha + (1.5-10^{-4})^\alpha + (1.5+10^{-6})^\alpha),$$
$$\frac{1}{4}(1.1^\alpha + 1.2^\alpha + (1.5-10^{-4})^\alpha + (1.5+10^{-6})^\alpha) = \frac{1}{4}(2\cdot 1.151^\alpha + (1.5+10^{-4})^\alpha + (1.5-10^{-6})^\alpha),$$
$$\frac{1}{4}(1.1^\alpha + 1.2^\alpha + (1.5-10^{-4})^\alpha + (1.5-10^{-6})^\alpha) = \frac{1}{4}(2\cdot 1.151^\alpha + (1.5+10^{-4})^\alpha + (1.5+10^{-6})^\alpha).$$

Solving for the α's, we obtain $\alpha_{[-1\ -1]} \approx 1.882$, $\alpha_{[-1\ 1]} \approx 1.885$, $\alpha_{[1\ -1]} \approx 2.123$, $\alpha_{[1\ 1]} \approx 2.125$. Furthermore, solving the equations for q_3, we find that it will merge to the opposite cluster for $\alpha < 3$. Therefore, we have 8 different ranges corresponding to the 8 equations.

This example suggests a general argument by induction which follows the same intuition. In each round, the new distances have less and less influence in the merge equations, ensuring the α's stay in the correct ranges to double the number of behaviors.

In our argument, we will utilize the following fact (true by elementary calculus).

Fact 1. *For all $0 \le z \le .01$ and $\alpha \in (1,3)$, the following are true about $g(z,\alpha) = (1.5-z)^\alpha - (1.5+z)^\alpha$ and $h(z,\alpha) = (1.1-z)^\alpha + (1.1+z)^\alpha - 2\cdot(((1.1-z)^\alpha + (1.1+z)^\alpha)/2)^{\frac{\alpha}{2}}$.*

1. *For $z > 0$, $g(z,\alpha) < 0$,*
2. *for a fixed z, g is nonincreasing in α,*
3. *for a fixed α, g is nonincreasing in z,*
4. *$h(0,\alpha) = 0$ and h is nondecreasing in z.*

Here are the details for the general construction. All distances will be between 1 and 2 so that the triangle inequality is satisfied. Given N, for all i,

$$\begin{aligned}
& d(p_a,q_a) = d(p_b,q_b) = 1, \\
& d(p_a,q_a) = d(p_a,q_b) = d(p_b,q_a) = d(p_b,q_b) = 2, \\
\forall i \le N,\ & d(p_a,p_i) = d(p_a,q_i) = 1.1 - q,\ d(q_a,p_i) = d(q_a,q_i) = 1.1 + q, \\
& d(p_b,p_i) = d(p_b,q_i) = d(q_b,p_i) = d(q_b,q_i) = \sqrt{\frac{1}{2}((1.1-q)^2 + (1.1+q)^2)}, \\
& d(p_i,q_i) = 2, \\
\forall 1 \le j < i \le N,\ & d(p_i,p_j) = d(p_i,q_j) = 1.5 + o_j \\
& d(q_i,p_j) = d(q_i,q_j) = 1.5 - o_j.
\end{aligned}$$

where q and o_j are offset values in $(0, .01)$ which we will specify later. Then for $\alpha \in (1, 3)$, the following are true.

- The first two merges are p_a to q_a and p_b to q_b,
- $\{p_i\}$ and $\{q_i\}$ will always prefer merging to A or B instead of merging to another singleton $\{p_j\}$ or $\{q_j\}$.

After the first two merges occur, all p_i and q_i are tied to first merge to A or B. For convenience, we specify the tiebreaking order as $\{p_1, q_1, \ldots, p_N, q_N\}$. Alternatively, at the end we can make tiny perturbations to the distances so that tiebreaking does not occur.

Next, we choose the value for q, which must be small enough to ensure that q_i always merges to the opposite cluster as p_i. Consider

$$\begin{aligned} h(\alpha, q, o_1, \ldots, o_N, \mathbf{x}) =& \frac{N+2}{N+3}\left((1.1+q)^\alpha + (1.1-q)^\alpha + \sum_{i<N} \mathbf{x}_i(1.5+o_i)^\alpha + 1.5^\alpha\right) \\ & - 2\cdot(((1.1+q)^2 + (1.1-q)^2)/2)^{\frac{\alpha}{2}} - \sum_{i<N} \mathbf{x}_i(1.5+o_i)^\alpha. \end{aligned}$$

If this equation is positive for all $\mathbf{x} \in \{-1, 1\}^{N-1}$, then q_N will always merge to the opposite cluster as p_N (and q_i will always merge to the opposite cluster as p_i, which we can similarly show by setting $o_j = 0$ in h for all $j > i$).

Note

$$h(\alpha, 0, 0, \ldots, 0, \mathbf{x}) = \frac{N+2}{N+3}\left(2\cdot 1.1^\alpha + (N+1)\cdot 1.5^\alpha\right) - 2\cdot 1.1^\alpha - N\cdot 1.5^\alpha > 0$$

for all $\mathbf{x}$ and all $\alpha \in (1, 3)$. Fact 1 implies there exists a $0 < q^* < .01$ such that $h(\alpha, q, 0, \ldots, 0, \mathbf{x})$ stays positive. Similarly, there exists a cutoff value $\delta > 0$ such that for all $0 < o_1, \ldots, o_N < \delta$, $\alpha \in (1, 3)$, and $\mathbf{x} \in \{-1, 1\}^{N-1}$, $h(\alpha, q^*, o_1, \ldots, o_k, \mathbf{x}) > 0$. Therefore, as long as we set all the offsets o_i less than δ, the merges will be as follows:

1. p_a merges to q_a and p_b merges to q_b.
2. For $1 \ldots, N$, p_i merges to A or B, and q_i merges to the opposite cluster. Then q_N will always merge to the opposite cluster as p_N.

Now we show that there are 2^N intervals for $\alpha \in (1, 3)$ which give unique behavior. Recall for $\mathbf{x} \in \bigcup_{i<N}\{-1, 1\}^i$, $E_{\mathbf{x}}$ is defined as

$$(1.1-q^*)^\alpha + (1.1+q^*)^\alpha - 2\cdot(\frac{1}{2}((1.1-q^*)^2 + (1.1+q^*)^2))^{\frac{\alpha}{2}} + \sum_{i<N} \mathbf{x}_i((1.5-o_i)^\alpha - (1.5+o_i)^\alpha).$$

For brevity, we denote

$$d = (\frac{1}{2}((1.1-q^*)^2 + (1.1+q^*)^2))^{\frac{1}{2}}.$$

We show the αs are correctly ordered by proving the following three statements with induction. The first statement is sufficient to order the αs, and the second two will help to prove the first.

1. There exist $0 < o_1, \ldots, o_N < \delta$ such that if we solve $E_{\mathbf{x}} = 0$ for $\alpha_{\mathbf{x}}$ for all $\mathbf{x} \in \bigcup_{i<N}\{-1, 1\}^i$, then the α's satisfy $\alpha_{[\mathbf{x}\ -1]} < \alpha_{[\mathbf{x}]} < \alpha_{[\mathbf{x}\ 1]}$ and for all $i < N$, $\alpha_{[\mathbf{x}\ 1]} < \alpha_{[\mathbf{y}\ -1]}$ for $\mathbf{x}, \mathbf{y} \in \{-1, 1\}^i$ and $\mathbf{x}_1 \ldots \mathbf{x}_i < \mathbf{y}_1 \ldots \mathbf{y}_i$.
2. For all $k' \leq N$ and $\alpha \in (1, 3)$,

$$(1.5+o_{k'})^\alpha - (1.5-o_{k'})^\alpha + \sum_{k'<i<N} ((1.5-o_i)^\alpha - (1.5+o_i)^\alpha) > 0.$$

3.

$$(1.1-q^*)^3+(1.1+q^*)^3-2\cdot d^3+\sum_{i<N}((1.5-o_i)^3-(1.5+o_i)^3)>0, \text{ and}$$
$$(1.1-q^*)+(1.1+q^*)-2\cdot d+\sum_{i<N}((1.5+o_i)-(1.5-o_i))<0.$$

We proved the base case in our earlier example for $n=10$. Assume for $k\leq N$, there exist $0<o_1,\ldots,o_k<\delta$ which satisfy the three properties. We first prove the inductive step for the second and third statements.

By inductive hypothesis, we know for all $k'\leq k$ and $\alpha\in(1,3)$,

$$(1.5+o_{k'})^\alpha-(1.5-o_{k'})^\alpha+\sum_{k'<i\leq k}((1.5-o_i)^\alpha-(1.5+o_i)^\alpha)>0,$$

Since there are finite integral values of $k'\leq k$, and the expression is >0 for all values of k', then there exists an $\epsilon>0$ such that the expression is $\geq\epsilon$ for all values of k'. Then we define z_a such that $(1.5+z_a)^\alpha-(1.5-z_a)^\alpha<\frac{\epsilon}{2}$ for $\alpha\in(1,3)$. Then for all $0<z<z_a$, $k'\leq k+1$, and $\alpha\in(1,3)$,

$$(1.5+o_{k'})^\alpha-(1.5-o_{k'})^\alpha+\sum_{k'<i\leq k+1}((1.5-o_i)^\alpha-(1.5+o_i)^\alpha)>0.$$

So as long as we set $0<o_{k+1}<z_a$, the inductive step of the second property will be fulfilled. Now we move to the third property. We have the following from the inductive hypothesis:

$$(1.1-q^*)^3+(1.1+q^*)^3-2\cdot d^3+\sum_{i\leq k'}((1.5-o_i)^3-(1.5+o_i)^3)>0,$$
$$(1.1-q^*)+(1.1+q^*)-2\cdot d+\sum_{i\leq k'}((1.5+o_i)-(1.5-o_i))<0.$$

We may similarly find z_b such that for all $0<o_{k+1}<z_b$,

$$(1.1-q^*)^3+(1.1+q^*)^3-2\cdot d^3+\sum_{i\leq k+1}((1.5-o_i)^3-(1.5+o_i)^3)>0,$$
$$(1.1-q^*)+(1.1+q^*)-2\cdot d+\sum_{i\leq k+1}((1.5+o_i)-(1.5-o_i))<0.$$

Now we move to proving the inductive step of the first property. Given $\mathbf{x}\in\{-1,1\}^k$, let $p(\mathbf{x}),n(\mathbf{x})\in\{-1,1\}^k$ denote the vectors which sit on either side of $\alpha_{\mathbf{x}}$ in the ordering, i.e., $\alpha_{\mathbf{x}}$ is the only $\alpha_{\mathbf{y}}$ in the range $(\alpha_{p(\mathbf{x})},\alpha_{n(\mathbf{x})})$ such that $|\mathbf{y}|=k$. If $\mathbf{x}=[1\ \ldots\ 1]$, then set $\alpha_{n(\mathbf{x})}=3$, and if $\mathbf{x}=[0\ \ldots\ 0]$, set $\alpha_{p(\mathbf{x})}=1$. Define

$$f(\alpha,\mathbf{x},z)=E_{\mathbf{x}}+(1.5-z)^\alpha-(1.5+z)^\alpha.$$

By inductive hypothesis, we have that $f(\alpha_{\mathbf{x}},\mathbf{x},0)=0$. We must show there exists $z_{\mathbf{x}}$ such that for all $0\leq z\leq z_{\mathbf{x}}$, $f(\alpha_{\mathbf{x}},\mathbf{x},z)<0$ and $f(\alpha_{n(\mathbf{x})},\mathbf{x},z)>0$. This will imply that if we choose $0<o_{k+1}<z_{\mathbf{x}}$, then $\alpha_{[\mathbf{x}\ 1]}\in(\alpha_{\mathbf{x}},\alpha_{n(\mathbf{x})})$.

Case 1: $\mathbf{x} \neq [1 \dots 1]$. Since $f(\alpha_{\mathbf{x}}, \mathbf{x}, 0) = 0$, and by Fact 1, then for all $0 < z < .01$, $f(\alpha_{\mathbf{x}}, \mathbf{x}, z) < 0$. Now denote i^* as the greatest index such that $\mathbf{x}_{i^*} = -1$. Then $n(\mathbf{x}) = [\mathbf{x}_1 \dots \mathbf{x}_{i^*-1}\ 1\ -1 \cdots -1]$. By statement 1 of the inductive hypothesis ($\alpha_{n(\mathbf{x})}$ is a root of $E_{n(\mathbf{x})} = 0$),

$$(1.1 - q^*)^{\alpha_{n(\mathbf{x})}} + (1.1 + q^*)^{\alpha_{n(\mathbf{x})}} - 2 \cdot d^{\alpha_{n(\mathbf{x})}} + \sum_{i \leq k} (n(\mathbf{x})_i (1.5 - o_i)^{\alpha_{n(\mathbf{x})}} - n(\mathbf{x})_i (1.5 + o_i)^{\alpha_{n(\mathbf{x})}}) = 0$$

From statement 2 of the inductive hypothesis, we know that

$$(1.5 - o_{i^*})^{\alpha_{n(\mathbf{x})}} - (1.5 + o_{i^*})^{\alpha_{n(\mathbf{x})}} + \sum_{i^* < i \leq k} ((1.5 + o_i)^{\alpha_{n(\mathbf{x})}} - (1.5 - o_i)^{\alpha_{n(\mathbf{x})}}) < 0.$$

It follows that

$$(1.1 - q^*)^{\alpha_{n(\mathbf{x})}} + (1.1 + q^*)^{\alpha_{n(\mathbf{x})}} - 2 \cdot d^{\alpha_{n(\mathbf{x})}} + \sum_{i < i^*} (n(\mathbf{x})_i (1.5 - o_i)^{\alpha_{n(\mathbf{x})}} - n(\mathbf{x})_i (1.5 + o_i)^{\alpha_{n(\mathbf{x})}}) > 0,$$

and furthermore,

$$(1.1 - q^*)^{\alpha_{n(\mathbf{x})}} + (1.1 + q^*)^{\alpha_{n(\mathbf{x})}} - 2 \cdot d^{\alpha_{n(\mathbf{x})}} + \sum_{i < i^*} (\mathbf{x}_i (1.5 - o_i)^{\alpha_{n(\mathbf{x})}} - \mathbf{x}_i (1.5 + o_i)^{\alpha_{n(\mathbf{x})}}) > 0.$$

Therefore, $f(\alpha_{n(\mathbf{x})}, 0) > 0$, so denote $f(\alpha_{n(\mathbf{x})}, 0) = \epsilon > 0$. Then because of Fact 1, there exists $z_{\mathbf{x}}$ such that $\forall 0 < z < z_{\mathbf{x}}$, $f(\alpha_{n(\mathbf{x})}, z) > 0$.

Case 2: $\mathbf{x} = [1 \dots 1]$. Since $f(\alpha_{\mathbf{x}}, 0) = 0$, and by Fact 1, then for all $0 < z < .01$, $f(\alpha_{\mathbf{x}}, z) < 0$. By property 3 of the inductive hypothesis, we have

$$(1.1 - q^*)^3 + (1.1 + q^*)^3 - 2 \cdot d^3 + \sum_{i \leq k} ((1.5 - o_i)^3 - (1.5 + o_i)^3) > 0,$$

so say this expression is equal to some $\epsilon > 0$. Then from Fact 1, there exists $z_{\mathbf{x}}$ such that for all $0 < z < z_{\mathbf{x}}$, $0 < (1.5 + z)^3 - (1.5 - z)^3 < \frac{\epsilon}{2}$. Combining these, we have $f(3, z) > 0$ for all $0 < z < z_{\mathbf{x}}$.

To recap, in both cases we showed there exists $z_{\mathbf{x}}$ such that for all $0 < z < \min(.01, z_{\mathbf{x}})$, $f(\alpha_{\mathbf{x}}, z) < 0$ and $f(\alpha_{n(\mathbf{x})}, z) > 0$. We may perform a similar analysis on a related function f', defined as $f'(\alpha, \mathbf{x}, z) = E_{\mathbf{x}} + (1.5 + z)^{\alpha} - (1.5 - z)^{\alpha}$ to show there exists $z'_{\mathbf{x}}$ such that for all $0 < z < z'_{\mathbf{x}}$, $f'(\alpha_{p(\mathbf{x})}, z) < 0$ and $f'(\alpha_{\mathbf{x}}, z) > 0$. We perform this analysis over all $\mathbf{x} \in \{-1, 1\}^k$.

Finally, we set $o_{k+1} = \min_{\mathbf{x}}(z_{\mathbf{x}}, z'_{\mathbf{x}}, z_a, z_b, .01)$. Given $\mathbf{x} \in \{-1, 1\}^k$, since $f(\alpha_{\mathbf{x}}, o_{k+1}) < 0$ and $f(\alpha_{n(\mathbf{x})}, o_{k+1}) > 0$, there must exist a root $\alpha_{[\mathbf{x}\ 1]} \in (\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$ (and by Fact 1, the function is monotone in α in the short interval $(\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$, so there is exactly one root). Similarly, there must exist a root $\alpha_{[\mathbf{x}\ -1]} \in (\alpha_{p(\mathbf{x})}, \alpha_{\mathbf{x}})$. Then we have shown $\alpha_{[\mathbf{x}\ -1]}$ and $\alpha_{[\mathbf{x}\ 1]}$ are roots of $E_{[\mathbf{x}\ -1]}$ and $E_{[\mathbf{x}\ 1]}$, respectively. By construction, $\alpha_{[\mathbf{x}\ -1]} < \alpha_{\mathbf{x}} < \alpha_{[\mathbf{x}\ 1]}$, so condition 1 is satisfied. Now we need to show condition 2 is satisfied. Given $\mathbf{x}, \mathbf{y} \in \{-1, 1\}^k$, let k' be the largest number for which $\mathbf{x}_i = \mathbf{y}_i$, $\forall i \leq k'$. Let $\mathbf{z} = \mathbf{x}_{[1 \dots k']} = \mathbf{y}_{[1 \dots k']}$. Then by the inductive hypothesis,

$$\alpha_{\mathbf{x}} < \alpha_{n(\mathbf{x})} \leq \alpha_{\mathbf{z}} \leq \alpha_{p(\mathbf{y})} < \alpha_{\mathbf{y}}.$$

It follows that

$$\alpha_{[\mathbf{x}\ -1]} < \alpha_{[\mathbf{x}\ 1]} < \alpha_{\mathbf{z}} < \alpha_{[\mathbf{y}\ -1]} < \alpha_{[\mathbf{y}\ 1]},$$

proving condition 2. This completes the induction.

□

Now we are ready to prove Lemma 8.

Proof of Lemma 8. Given n, and setting $N = \lfloor (n-8)/4 \rfloor$, we will show there exists a clustering instance (V, d) of size $|V| = n$, a witness r, and a set of $2^N + 2$ α's $1 = \alpha_0 < \alpha_1 < \cdots < \alpha_{2^N} < \alpha_{2^N+1} = 3$, such that $\Phi^{(p)}_{\mathcal{A}_3(\alpha)}(\mathcal{V})$ oscillates above and below r between each interval (α_i, α_{i+1}).

We start by using the construction from Lemma 8, which gives a clustering instance with $2N+8$ points and $2^N + 2$ values of α for which α-linkage creates a unique merge tree. The next part is to add $2N$ more points and define a witness r so that the cost function alternates above and below r along each neighboring α interval, for a total of 2^N oscillations. Finally, we will finish off the proof in a manner similar to Lemma 6.

Starting with the clustering instance (V, d) from Lemma 8, we add two sets of points, C_A and C_B, which do not interfere with the previous merges, and ensure the cost functions alternates. Let $C_A = \{c_a, c'_a, a_1, a_2, \ldots, a_N\}$ and $C_B = \{c_b, c'_b, b_1, b_2, \ldots, b_N\}$. All distances between two points in C_A are 1, and similarly for C_B. All distances between a point in C_A and a point in C_B are 2. The distances between $C_A \cup C_B$ and $A \cup B$ are as follows (we defined the sets A and B in Lemma 8).

$$
\begin{aligned}
&d(p_a, c_a) = d(p_a, c'_a) = d(q_a, c_a) = d(q_a, c'_a) = 1.51,\\
&d(p_b, c_b) = d(p_b, c'_b) = d(q_b, c_b) = d(q_b, c'_b) = 1.51,\\
&d(p_a, c_b) = d(p_a, c'_b) = d(q_a, c_b) = d(q_a, c'_b) = 2,\\
&d(p_b, c_a) = d(p_b, c'_a) = d(q_b, c_a) = d(q_b, c'_a) = 2,\\
&d(p_a, c) = d(q_a, c) = d(p_b, c) = d(q_b, c) = 2 \ \forall c \in C_A \cup C_B \setminus \{c_a, c'_a, c_b, c'_b\},\\
&d(c, p_i) = d(c, q_i) = 1.51 \ \forall 1 \le i \le N-1 \text{ and } c \in C_A \cup C_B.
\end{aligned}
$$

We will specify the distances between $\{c_a, c'_a, c_b, c'_b\}$ and $\{p_N, q_N\}$ soon, but they will be in $[1.6, 2]$. So at the start of the merge procedure, all points in C_A merge together, and all points in C_B merge together. Then all merges from Lemma 8 take place, because all relevant distances are smaller than 1.51. We end up with four sets: A, B, C_A, and C_B. The pairs (A, B) and (C_A, C_B) are dominated by distances of length 2, so the merges (C_A, A) and (C_B, B) will occur, which dominate (C_A, B) and (C_B, A) because of the distances between $\{p_a, q_a, p_b, q_b\}$ and $\{c_a, c'_a, c_b, c'_b\}$. The final merge to occur will be $(C_A \cup A, C_B \cup B)$, however, the 2-median pruning step will clearly pick the 2-clustering $C_A \cup A$, $C_B \cup B$, since no other clustering in the tree has almost all distances ≤ 1.51. Then by construction, c_a or c'_a will be the best center for $C_A \cup A$, which beat p_a and q_a because $1.51 \cdot (2N) < 1.1 \cdot N + 2 \cdot N = 1.55 \cdot (2N)$. Similarly, c_b or c'_b will be the best center for $C_B \cup B$. Note that centers $\{c_a, c'_a\}$ and $\{c_b, c'_b\}$ currently give equivalent 2-median costs. Denote this cost by r' (i.e., the cost before we set the distances to p_N and q_N).

Now we set the final distances as follows.

$$
\begin{aligned}
&d(c_a, p_N) = d(c_b, q_N) = 1.6,\\
&d(c'_a, p_N) = d(c'_b, q_N) = 1.7,\\
&d(c'_a, q_N) = d(c'_b, p_N) = 1.8,\\
&d(c_a, q_N) = d(c_b, p_N) = 1.9.
\end{aligned}
$$

If $p_N \in A$ and $q_N \in B$, then c_a and c_b will be the best centers, achieving cost $r' + 3.2$ for $(C_A \cup A, C_B \cup B)$. If $p_N \in B$ and $q_N \in A$, then c'_a and c'_b will be the best centers, achieving cost $r' + 3.6$ for $(C_A \cup A, C_B \cup B)$.

The distances are also constructed so that in the variant where the pruning outputs the optimal centers, and then all points are allowed to move to their closest center, the cost still oscillates. First note that no points other than p_N and q_N are affected, since $d(c_a, p_i) = d(c_a, q_i)$ for $i < N$, and similarly for c_b. Then p_N will move to the cluster with c_a or c'_a, and q_N will move to the cluster with c_b or c'_b. If p_N was originally in A, then the cost is $r' + 3.2$, otherwise the cost is $r' + 3.4$.

In either scenario, we set $r = r' + 3.3$. Then we have ensured for all $\mathbf{x} \in \{-1, 1\}^{N-1}$, the cost for $\alpha \in (\alpha_{p(\mathbf{x})}, \alpha_{\mathbf{x}})$ is $< r$, and the cost for $\alpha \in (\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$ is $> r$.

We have finished our construction of a clustering instance whose cost function alternates 2^N times as α increases. To finish the proof, we will show there exists a set $S = \{V_1, \ldots, V_s\}$ of size $s = N = \lfloor (n-8)/4 \rfloor \in \Omega(n)$ that is shattered by $\mathcal{A}$. Such a set has 2^N orderings total. For V_1, we use the construction which alternates 2^N times. For V_2, we use the same construction, but we eliminate (p_N, q_N) so that there are only $N-1$ rounds (the extra two points can be added to C_A and C_B to preserve $|V_2| = n$). Then V_2's cost will alternate $\frac{1}{2} \cdot 2^N$ times, between the intervals $(\alpha_{p(\mathbf{x})}, \alpha_{\mathbf{x}})$ and $(\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$, for $\mathbf{x} \in \{-1, 1\}^{N-2}$. So V_2 oscillates every other time V_1 oscillates, as α increases. In general, V_i will be the construction with only $N - i + 1$ rounds, oscillating $2^{\frac{N}{2^{i-1}}}$ times, and each oscillation occurs every other time V_{i-1} oscillates. This ensures for every $\mathbf{x} \in \{-1, 1\}^{N-1}$, $(\alpha_{p(\mathbf{x})}, \alpha_{\mathbf{x}})$ and $(\alpha_{\mathbf{x}}, \alpha_{n(\mathbf{x})})$ will have unique labelings, for a total of 2^N labelings. This completes the proof. □

Note C.2. *As in Lemma 6, this lower bound holds even if the cost function is the symmetric distance to the ground truth clustering. Merely let p_N and q_N belong to different ground truth clusters, but for all $i < N$, p_i and q_i belong to the same ground truth cluster. Since in each adjacent α interval, p_N and q_N switch clusters, this shows the symmetric distance to the ground truth clustering oscillates between every interval.*

C.1 Analyzing $\mathcal{A}_3$ over restricted classes of clustering instances

Below we consider restricted classes of clustering instances and improve on the pseudo-dimension bounds of $\mathcal{H}_{\mathcal{A}_3}$ as compared to Lemma 7. In particular, we consider the class $\mathbb{V}_\beta$ that consists of clustering instances in which the distances take one of at most β ($\beta \in \mathbb{N}$) real values. A natural example would be one in which all distances are integers and are less than some value H. Here, $\beta = H$. For this case, we show a tight bound.

Theorem 20. *For any objective function Φ, let*

$$\mathcal{H}_{\mathcal{A}_3,\Phi} = \left\{ \Phi_{\mathcal{A}_3(\alpha)} : \mathbb{V}_\beta \to \mathbb{R}_{\geq 0} \mid \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\}.$$

Then $Pdim(\mathcal{H}_{\mathcal{A}_3}, \Phi) = O\left(\min(\beta \log n, n)\right)$.

Proof. The proof for the upper bound follows a similar line of reasoning as that of Lemma 7. For a particular instance $\mathcal{V}$, let $\{d_1, d_2, \ldots, d_\beta\}$ denote the set of β values that its distances can take. The linkage criterion for merging A, B can be expressed as $\frac{1}{|A||B|} \sum_{i=1}^{\beta} a_i d_i^\alpha$ where, each a_i can take one of at most $O(n^2)$ values corresponding to the number of pairs of points at a distance d_i. Therefore, iterating over all pairs of subsets (A, B) like we did in the proof of Lemma 7, we can only list at most $O\left((n^2)^{\beta+1}\right)$ potential linkage criteria. Therefore, the set of all pairs of subsets (A, B) and (X, Y) will induce at most $O\left((n^2)^{2(\beta+1)}\right)$ unique comparisons between two linkage criteria. By the same argument as in the proof of Lemma 7, since each such comparison has $O(n^2)$ roots, if $\mathcal{S} \in \mathbb{V}^m$ is a shatterable set, then $2^m \leq n^2((n^2)^{2(\beta+1)})$, which means that $m = O(\beta \log n)$. □

Theorem 21. *For any objective function $\Phi^{(p)}$, $Pdim(\mathcal{H}_{\mathcal{A}_3,\Phi^{(p)}}) = \Omega\left(\min(\beta, n)\right)$.*

Proof. For the lower bound, we use a similar line of reasoning as in Lemma 8. In each round of the construction in Lemma 8, only a constant number of distinct edge lengths are added. I.e. the offsets o_i define new distances $1.5+o_i$ and $1.5-o_i$ per round, but the set of the rest of the distances has constant size. Therefore, we can easily modify the proof to construct a clustering instance with $\Omega(m)$ rounds using m distinct distances. This instance will have $2^{\Omega(\beta)}$ distinct behaviors depending on α, however this reasoning is only consistent for $\beta = o(n)$. For $\beta = \omega(n)$, we may inherit the lower bound from Lemma 8, so the final pseudo-dimension lower bound is $\Omega(\min(\beta, n))$. □

C.2 An Interpolation

So far, the linkage criteria was based on the distances between either two pairs of points, or every single pair between two sets considered for merging. Now we provide an interpolation between these two extremes. In particular, we define a linkage criterion which uses σ different distances between the sets for comparison. In particular, for any two sets A and B, we define an abstract rule to pick σ pairs of points (p_i, q_i) from $A \times B$. For example, a natural choice would be to pick the $(\sigma-1)$-th quantiles of the set of distances between points in A and B along with the maximum and minimum distances. On picking these points, we define the criterion as a function of these σ distances as follows.

$$\mathcal{A}_{1,\sigma} = \left\{ \sum_{i=1}^{\sigma} \alpha_i d(p_i, q_i) \mid \vec{\alpha} = (\alpha_1, \ldots, \alpha_\sigma) \in \mathbb{R}^\sigma \right\}, \tag{3}$$

$$\mathcal{A}_{2,\sigma} = \left\{ \left(\sum_{i=1}^{\sigma} d(p, q))^\alpha \right)^{1/\alpha} \mid \alpha \in \mathbb{R} \cup \{\infty, -\infty\} \right\}. \tag{4}$$

Observe that $\mathcal{A}_{1,\sigma}$ has multiple parameters unlike any of the classes of algorithms we have discussed. Therefore, the analysis for $\mathcal{A}_{1,\sigma}$ is considerably different from the rest as shown below. We use notations similar to the previous sections.

Theorem 22. *For any Φ, let*

$$\mathcal{H}_{\mathcal{A}_{1,\sigma},\Phi} = \left\{ \Phi_{\mathcal{A}_{1,\sigma}(\vec{\alpha})} : \mathbb{V} \to \mathbb{R}_{\geq 0} \mid \vec{\alpha} \in \mathbb{R}^\sigma \right\}.$$

The pseudo-dimension of $Pdim(\mathcal{H}_{\mathcal{A}_{1,\sigma},\Phi}) = O(\sigma^2 \log n)$.

Proof. The proof parallels that of Lemma 5. Consider two pairs of sets A, B and X, Y that can be potentially merged. Regardless of the parameters chosen, we know that the linkage criterion first chooses σ pairs of points $(p_i, q_i) \in A \times B$ and $(x_i, y_i) \in X \times Y$. Now, the decision to merge A, B before X, Y or vice versa is determined by the sign of $\sum_{i=1}^{\sigma} \alpha_i d(p_i, q_i) - \sum_{i=1}^{\sigma} \alpha_i d(x_i, y_i)$. If this expression evaluates to zero, we will break ties arbitrarily but consistently.

Observe that for a given set of values for $d(p_i, q_i)$ and $d(x_i, y_i)$, the above expression is either 0 for all α, or is equal to zero for the hyperplane passing through the origin and normal to $(d(p_1, q_1) - d(x_1, y_1), \ldots, d(p_\sigma, q_\sigma) - d(x_\sigma, y_\sigma))$ in the parameter space $\mathbb{R}^\sigma$. This hyperplane divides the parameter space into two half-spaces each of which correspond to merging one pair of sets before the other.

Next we note that, for a given problem instance, as we iterate over all pairs of sets (A, B) and (X, Y), we can list only $O\left(n^{4\sigma}\right)$ possible choices for the hyperplane as there are only so many 4σ-tuples of points p_i, q_i, x_i, y_i. Thus, for m problem instances, we can list at most $O\left(mn^{4\sigma}\right)$

different hyperplanes in $\mathbb{R}^\sigma$. These hyperplanes, can partition the parameter space into at most $O\left(\left(mn^{4\sigma}\right)^\sigma\right)$ regions such that all the parameter settings in a given region correspond to identical merge trees and hence identical costs/labeling induced by the witnesses. By an argument similar to the proof for Theorem 6, we can concluce that $m = O(\sigma^2 \log n)$.

□

Theorem 23. *For any Φ, let*

$$\mathcal{H}_{\mathcal{A}_{2,\sigma},\Phi} = \left\{\Phi_{\mathcal{A}_{2,\sigma}(\alpha)} : \mathbb{V} \to \mathbb{R} \mid \alpha \in \mathbb{R}_{\geq 0} \cup \{\infty, -\infty\}\right\}.$$

Then $Pdim(\mathcal{H}_{\mathcal{A}_{2,\sigma}}) = O\left(\min(\sigma \log n, n)\right)$.

Proof. This proof follows the same reasoning as in Lemma 7. The decision of whether to merge any two pairs of sets (A, B) and (X, Y), is determined by the sign of the difference in their linkage criterion which is $\sum_{i=1}^{\sigma} d^\alpha(p_i, q_i) - \sum_{i=1}^{\sigma} d^\alpha(x_i, y_i)$ where the points p_i, q_i, x_i, y_i are fixed for a given pairs of set. We know from Theorem 19 that this expression has only $O(n^2)$ roots. Furthermore, as we iterate over all pairs of sets (A, B) and (X, Y), we can only generate as many such expressions as there are 4σ-tuples of points p_i, q_i, x_i, y_i. In particular, we will list only $O(n^{4\sigma})$ such expression each with $O(n^2)$ roots. In summary, similar to the proof of Lemma 7, we can argue that for a set of samples in $\mathcal{V}^m$ to be shattered, we need that $2^m \leq n^{4\sigma} \cdot n^2$ i.e., $m = O(\sigma \log n)$. □

Theorem 24. *For any $\Phi^{(p)}$ $Pdim(\mathcal{H}_{\mathcal{A}_{2,\sigma},\Phi^{(p)}}) = \Omega(\sigma)$.*

Proof. We use a similar line of reasoning as in Lemma 8. In round i of the construction in Lemma 8, the merges were between a set of size 1 and a set of size $i + 2$. Therefore, we can easily modify the proof to construct a clustering instance with $\Omega(\sigma)$ rounds, and the merge equations will be the same as in Lemma 8. This instance will have $2^{\Omega(\sigma)}$ distinct behaviors depending on α, and so the pseudo-dimension is $\Omega(\sigma)$. □

Algorithm Class	**Linkage rule**	**Pseudo-dimension**	**Runtime of Learning Algorithm**
$\mathcal{A}_1$	$\alpha \min_{p\in A,\ q\in B} d(p,q) + (1-\alpha) \max_{p\in A,\ q\in B} d(p,q)$ for $\alpha \in (0,1)$	$\Theta(\log n)$	$\tilde{O}(n^8 \mathbf{t_{alg}})$
$\mathcal{A}_2$	$\min_{p\in A, q\in B}(d(p,q))^\alpha + \max_{p\in A, q\in B}(d(p,q))^\alpha$ for $\alpha \in \mathbb{R}$	$\Theta(\log n)$	$\tilde{O}(n^8 \mathbf{t_{alg}})$
$\mathcal{A}_3$	$\frac{1}{\lvert A\rvert\lvert B\rvert} \sum_{p\in A, q\in B} d(A,B)^\alpha$ for $\alpha \in \mathbb{R}$	$O(n \log n)$, $\Omega(n)$	$\tilde{O}(n^2 3^{2n} \mathbf{t_{alg}})$
$\mathcal{A}_3$	with β unique distances	$\tilde{\Theta}(\min(\beta, n))$	$\tilde{O}(n^{4(\beta+1)} \mathbf{t_{alg}})$
$\mathcal{A}_{1,\sigma}$	$\sum_{i=1}^{\sigma} \alpha_i d(p_i, q_i)$ for $\alpha_i \in \mathbb{R}$	$O(\sigma^2 \log n)$	
$\mathcal{A}_{2,\sigma}$	$\sum_{i=1}^{\sigma} d(p_i, q_i)^\alpha$ for $\alpha_i \in \mathbb{R}$	$\tilde{\theta}(\sigma)$	$\tilde{O}(n^{2\sigma+2} \mathbf{t_{alg}})$

Table 2: The different classes of algorithms and their corresponding linkage rule, pseudo-dimension, and runtime. $\mathbf{t_{alg}}$ denotes the runtime of the α-linkage algorithm for an arbitrary linkage rule.