Algorithms Above The Noise Floor

by

Ludwig Schmidt

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author . . . .
Department of Electrical Engineering and Computer Science
May 23, 2018

Certified by . . . .
Piotr Indyk
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Algorithms Above The Noise Floor

by

Ludwig Schmidt

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Many success stories in the data sciences share an intriguing computational phenomenon. While the core algorithmic problems might seem intractable at first, simple heuristics or approximation algorithms often perform surprisingly well in practice. Common examples include optimizing non-convex functions or optimizing over non-convex sets. In theory, such problems are usually NP-hard. But in practice, they are often solved sufficiently well for applications in machine learning and statistics. Even when a problem is convex, we often settle for sub-optimal solutions returned by inexact methods like stochastic gradient descent. And in nearest neighbor search, a variety of approximation algorithms works remarkably well despite the "curse of dimensionality".

In this thesis, we study this phenomenon in the context of three fundamental algorithmic problems arising in the data sciences.

- In constrained optimization, we show that it is possible to optimize over a wide range of non-convex sets up to the statistical noise floor.
- In unconstrained optimization, we prove that important convex problems already require approximation if we want to find a solution quickly.
- In nearest neighbor search, we show that approximation guarantees can explain much of the good performance observed in practice.

The overarching theme is that the computational hardness of many problems emerges only below the inherent "noise floor" of real data. Hence computational hardness of these problems does not prevent us from finding answers that perform well from a statistical perspective. This offers an explanation for why algorithmic problems in the data sciences often turn out to be easier than expected.

Thesis Supervisor: Piotr Indyk
Title: Professor of Electrical Engineering and Computer Science

To my parents

Acknowledgments

First and foremost, I want to thank my advisor Piotr Indyk. Starting graduate school without much research experience, I was initially quite uncertain which direction to pursue. Deciding to work with Piotr turned out to be an excellent choice in many ways. Piotr was always supportive of my interests at the intersection of theory and practice and introduced me to many problems of this flavor. The direction he put me on turned out to be a perfect combination of my background in combinatorial algorithms with my interests in machine learning. Along the way, I learned a lot from Piotr's quick insights and vast knowledge of algorithms. Piotr was always supportive of my career and generously kept me free from worries about funding. Finally, Piotr is a remarkably kind person and truly cares about his students, which made his research group a pleasure to be in.

Next, I want to thank Chinmay Hegde. Chin came to MIT as a postdoc at the beginning of my second year and effectively became my co-advisor. Chin taught me many topics in signal processing, which was invaluable to get started with my research. Over the years, we developed both a very fruitful research collaboration and a good friendship. Almost all of my early papers are co-authored with Chin, and my work with Piotr and Chin forms the core of this thesis.

In the middle of my PhD, I benefited from the guidance of four mentors. Ben Recht and Martin Wainwright were very welcoming when they hosted me at Berkeley in Spring 2016. The following two summers (and part-time in between), I was lucky to intern with Yoram Singer and Kunal Talwar at Google Brain. Talking to Ben, Kunal, Martin, and Yoram broadened my perspective beyond the core algorithmic problems I had worked on and prompted me to think more deeply about statistics and machine learning. This had a lasting (and very positive!) effect on my research and significantly influenced this thesis.

The final year of my PhD was not entirely planned and arguably correlated with the fact that I began working with Aleksander Mądry. It turned out that Aleksander was about to launch a research program that aligned very well with my interests and led me to many fascinating problems (which more than compensate for the repeatedly delayed thesis defense!). Moreover, Aleksander has been a great mentor over the past year and I am very grateful for his support.

Besides Aleksander and Piotr, my third committee member is Costis Daskalakis, who was also part of my RQE committee. I would like to thank Costis for his input, encouraging words, and being flexible with the timeline of this thesis.

A very important part of my PhD are my collaborators. Over the past seven years, I was fortunate to join many colleagues in research projects. Listing all of them (more than 40) would go beyond the limits of these acknowledgements, but I am very grateful to each of them. You taught me many topics through our collaborations and I truly enjoyed the company. Thank you!

However, I can't get around mentioning two student collaborators. Jerry Li has been the perfect companion for my ventures into learning theory. I learned a lot from Jerry's math background, and I hope I didn't do too much damage by adding experiments to our papers. The other student who influenced my work significantly is Michael Cohen, who tragically passed away during the last year of my PhD. Talking to Michael has been one of the greatest intellectual privileges during my time at MIT, and also one of the biggest sources of fun conversations. It is impossible to do justice to Michael in a few sentences. I can just say that I truly miss him.

Beyond the academic side of graduate school, I was lucky to be part of many great communities at MIT. The theory group has been an excellent place not only for work but also for companionship (thanks for organizing, G!). I lived in Ashdown House during my entire time at MIT, which was always a welcoming home to come back to outside the department. The MIT Outing Club (MITOC) has been an incredible community for my outdoor activities. My friends at MITOC introduced me to all varieties of hiking and brought me to mountains that I would have never climbed alone. Thank you (among many others) Anthony, Ben, Elliott, Eric, Katherine, Katie, Maddie, Margie, Matthew, Meghan, Michele, MinWah, Nathaniel, Rachel, Richard, Rob, Vlad and Yuval!

Over the past few years, I was fortunate to meet many wonderful friends at MIT, Berkeley, and beyond. I am surely forgetting some important names (sorry!), but I would still like to mention a few friends individually: Adam, Alex, Alison, Arturs, Christos, Fanny, Fredrik, Henry, Madars, Maithra, Matt, Reinhard, Philipp, Ruby, and Ryan: thank you for all the great time together and supporting me throughout grad school!

An important milestone on the way to MIT were my undergrad years at Cambridge. My directors of study at Trinity, Arthur Norman and Sean Holden, gave me plenty of freedom and helped me get into MIT. I also learned a lot from my friends Ada, Freddie, Martin, Rasmus, and Sean, who – together with many others – provided great company over the years. I'd especially like to thank Felix Arends, who was always a few years ahead of me and inspired me to go to Cambridge and then to intern at Google.

While my school days are a while in the past now, it is clear that I would not be here without the early support from that time. My high school teachers Helmut Eckl, Alois Stöhr, and Stefan Winter got me started with math and computer science competitions early, which put me on a trajectory that I continue to benefit from. The most influential competition for me has been the Bundeswettbewerb Informatik (BWInf). Together with the informatics olympiads, it shifted my focus from mathematics to computer science. The BWInf and the German olympiad team are still organized by Wolfgang Pohl, who I am very grateful to. I would also like to thank the "Schülerstudium" team at the University of Passau for giving me an early start with studying computer science. And school would have been boring without my friends: Andreas, Andreas, Henry, Johannes, Julian, Matthias, Max, Simon, Simon, Simon, Stephan (and everyone I can't list here right now) – thank you for all the good times!

Finally, I would like to thank my family. My brother Valentin has kept me grounded over the years and has always been a great source of fun and laughter. My parents, Barbara and Norbert, have supported me in everything I wanted to do, and I could not be more grateful for that. This thesis is dedicated to them.

Contents

Chapter 1

Introduction

Over the past two decades, the reach of data sciences such as machine learning, statistics, and signal processing has grown tremendously. Fueled by our ever-increasing capabilities to collect large amounts of data, statistical algorithms now underlie a broad range of modern technology. Well-known examples include web search, medical imaging, bioinformatics, computer vision, and speech recognition – just to name a few. From a computational perspective, many of these success stories highlight an intriguing overarching theme: while the core algorithmic problems might seem costly to solve or even intractable at first, simple heuristics or approximation algorithms often perform surprisingly well in practice.

Examples of this phenomenon are abundant. In statistical estimation tasks, it is now standard practice to optimize over non-convex sets (e.g., sparse vectors or low-rank matrices), even though such problems are NP-hard in general. When training neural networks or fitting random forests, iterative methods often find good solutions in spite of the non-convex loss functions. Even for convex problems, we commonly resort to stochastic gradient methods that return only coarse answers yet still provide good generalization properties. And when searching through large collections of complex data such as images or videos, approximate data structures quickly return the right answer in spite of the "curse of dimensionality".

This discrepancy between the inexact nature of popular algorithms and their good empirical performance raises a fundamental question:

When and why do approximate algorithms suffice in statistical settings?

There are multiple reasons to study this question. On the theoretical side, we should understand how statistical settings evade fundamental hardness results. After all, the hallmark of a scientific theory is its predictive power. But when it comes to predicting performance in machine learning or statistics, the dominant "worst case" view of algorithms often seems too pessimistic.

On the more applied side, the growing influence of data sciences also means that errors made by our algorithms can have direct negative consequences in the real world. If we want to responsibly utilize these algorithms in scenarios with serious impact, it is important to delineate the regimes where we can safely rely on their answers. Finally, understanding why inexact solutions work well often enables us to develop new algorithms that achieve even better computational or statistical performance.

The goal of this thesis is to explain and extend the success of approximate algorithms in statistical settings. To this end, we investigate three fundamental algorithmic problems in the data sciences. All three vignettes share a common theme: approximate algorithms consistently avoid computational hardness results, but still achieve strong performance *both* in theory and in practice.

At a high level, our results show that the computational hardness of many algorithmic problems only emerges when we compute high-accuracy solutions below the inherent "noise floor" of the data we collected. So from the perspective of the statistical problem we want to solve, the computational hardness of the intermediate algorithmic problem often turns out to be irrelevant. This explains why algorithmic problems in the data sciences can usually be solved sufficiently well in practice, even if a worst case perspective might at first suggest otherwise.

The work in this thesis makes contributions to the following areas:

Constrained optimization. Optimizing over convex sets is a well-understood task with polynomial-time algorithms for a broad range of constraints. In contrast, even simple problems involving non-convex sets are NP-hard. Nevertheless, seminal results in compressive sensing show that statistical settings improve the situation markedly. For certain non-convex sets such as sparse vectors and low-rank matrices, we now have reliable algorithms and a comprehensive understanding of both computational and statistical aspects. But the situation is still less clear for general non-convex sets.

In this thesis, we introduce an extension of projected gradient descent for arbitrary non-convex cones that achieves (near-) optimal statistical accuracy. Moreover, we identify a new notion of *approximately* projecting onto non-convex sets. This notion allows us to optimize over constraints that would otherwise be NP-hard to handle. In addition to their strong theoretical guarantees, our algorithms also improve over the state of the art in numerical experiments.

Unconstrained optimization. Stochastic gradient descent (SGD) is a ubiquitous algorithm in machine learning and underlies many successful classification techniques. On the one hand, research has shown that SGD returns solutions with good statistical properties. On the other hand, SGD relies on approximate gradients and has only modest convergence guarantees as an optimization algorithm. As a result, SGD is often unreliable and requires instance-specific tuning of hyperparameters. Such brittle performance is a common hidden cost of approximate algorithms. So why do we resort to sometimes unreliable algorithms?

To explain our dependence on approximate algorithms, we show that computational limits already emerge when solving important convex problems to high accuracy. Two concrete examples in this thesis are kernel support vector machines (SVM) and kernel ridge regression, which are widely used methods in machine learning. For these problems, a reliable high-accuracy algorithm with subquadratic running time would break a well-established barrier in complexity theory. So our use of inexact algorithms like SGD is probably due to the fact that there is no "silver bullet" algorithm for these problems. There are inherent computational limits when working with powerful data representations such as kernels. This shows that the negative side effects of approximate algorithms are sometimes unavoidable.

Nearest neighbor search. Quickly searching through vast amounts of data plays a key role in important technologies such as web-scale search engines and recommender systems. However, there are (conditional) lower bounds telling us that fast nearest neighbor search should be impossible in general. So what structure in the data do successful algorithms exploit? The Locality Sensitive Hashing (LSH) framework provides a rigorous explanation. But empirically, LSH lost ground to other methods over the past decade.

In this thesis, we bridge this divide between theory and practice and re-establish LSH as a strong baseline. We thereby demonstrate that the LSH theory can explain much of the good performance observed in practice. Moreover, we show empirically that the assumptions in the LSH theory are naturally satisfied when the dataset is derived from a trained embedding such as a neural network. On the practical side, our LSH implementation FALCONN has become a popular library for nearest neighbor search on GitHub.

In the next three sections, we outline our concrete contributions in more detail. At a high level, each part of this thesis corresponds to one of the three algorithmic problems mentioned above.

1.1 Optimizing over non-convex cones

The goal in constrained optimization is to minimize an objective function $f : \mathbb{R}^d \to \mathbb{R}$ over a given *constraint set* $\mathcal{C} \subseteq \mathbb{R}^d$. This fundamental paradigm has found a myriad of applications and has led to influential algorithmic techniques such as the simplex algorithm or interior point methods. The classical theory of optimization shows that for a broad range of *convex* constraint sets, it is possible to solve constrained optimization problems in polynomial time (for instance, linear programs or semidefinite programs) [40, 54, 174, 175]. Building on this work, a large body of research from the past 15 years has focused on applications with *non-convex* constraints. Here, the situation is more subtle. Even simple problems over non-convex sets can be NP-hard, yet are often solved successfully in practice.

One important example in the data sciences are estimation problems. In a linear estimation problem, we want to (approximately) recover an unknown parameter vector $\theta^* \in \mathbb{R}^d$ from a small number of linear observations. We can summarize n such observations in a measurement matrix $X \in \mathbb{R}^{n \times d}$ and a response vector $y \in \mathbb{R}^n$ to get the following data model:

$$y \;=\; X\theta^* + e \;,$$

where $e \in \mathbb{R}^n$ is an unknown noise vector. Since this is a linear model, a natural approach to estimating θ^* is a constrained least squares problem of the form

$$\begin{aligned} &\text{minimize } \|y - X\theta\|_2^2 \\ &\text{subject to } \theta \in \mathcal{C} \;. \end{aligned} \tag{1.1}$$

In this formulation, the constraint set $\mathcal{C}$ encodes our prior knowlege about the unknown parameters θ^* (e.g., a sparsity or rank constraint). While the linear setup might seem simple at first, it already applies to a broad range of problems including magnetic resonance imaging [157], recommender systems [59], and selecting important features in gene expression data

[222]. In these applications, utilizing prior knowledge through carefully chosen constraints is crucial for finding a good estimate from a small number of samples.

From a worst case perspective, Problem (1.1) might seem hopeless. For instance, the least squares problem is already NP-hard for simple non-convex constraints such as $\mathcal{C}$ being the set of sparse vectors [168]. Nevertheless, various algorithms often find good estimates in practice [73, 221]. To explain this phenomenon, the fields of compressive sensing and high-dimensional statistics have developed a rich theory for optimizing over certain non-convex sets [69, 101, 114, 231]. Under regularity assumptions on the matrix X, corresponding algorithms provably find an estimate $\hat{\theta}$ such that $\hat{\theta} \approx \theta^*$ up to the inherent noise in the problem.

At a high level, there are two algorithmic approaches to estimating θ^*:

- *Convex relaxations* relate the non-convex constraint back to a convex counterpart. The statistical guarantees for estimating θ^* are well understood and in many cases optimal. While this approach has been very successful for some non-convex constraints (e.g., sparse vectors and low-rank matrices [59, 60, 90, 186]), it is not clear how to derive convex relaxations for other constraint sets (e.g., graph sparsity or group sparsity [232]).

- *Projected gradient descent* (PGD) forgoes the convex route and directly works with the non-convex set. While this succeeds for well-conditioned problems [50, 80, 131, 170, 177], the PGD approach does not match the sample complexity of convex relaxations in the general case [133]. Moreover, PGD relies on access to an efficient *projection oracle* for the non-convex set $\mathcal{C}$.[1] Unfortunately, the projection problem can also be NP-hard (among others, the culprits are again graph sparsity and group sparsity [33, 118]). In such cases, PGD does not have a polynomial running time.

Methods for sparse vectors and low-rank matrices have been highly successful despite the non-convex constraints. Hence it is important to understand how general this phenomenon is. Specifically, we investigate the following question:

For what non-convex sets can we solve constrained estimation problems efficiently?

In this thesis, we significantly extend the class of non-convex sets for which we can efficiently optimize in a statistical setting. In particular, we make the following three contributions:

Result 1. We introduce a generalized version of projected gradient descent for non-convex sets that achieves the optimal sample complexity for estimation problems with arbitrary conditioning.

This result provides an alternative path to optimal statistical performance that does not rely on convex relaxations. However, the new algorithm still relies on projections for the set $\mathcal{C}$, which can be an NP-hard sub-problem. To ameliorate this issue, we show that our generalization of PGD also works when combined with a weaker notion of projection.

[1] To be precise, projecting an input $\theta^{in} \in \mathbb{R}^d$ to the set $\mathcal{C}$ means solving the problem $\min_{\theta \in \mathcal{C}} \|\theta - \theta^{in}\|_2$.

Result 2. Our generalized PGD algorithm still succeeds when it only has access to approximate projections.

While PGD has been combined with approximate projections before [39, 49, 82, 107, 108, 144, 145, 205], our framework is the first that leads to faster algorithms for a broad range of constraint sets. An important insight is that we need to combine PGD with two complementary notions of approximate projections *simultaneously*.

To instantiate our new PGD framework, we have to provide approximate projections for concrete constraint sets. To this end, we combine ideas from both continuous and discrete optimization. Our faster projections leverage a broad range of techniques from the approximation algorithms literature.

Result 3. We exhibit many constraint sets for which an approximate projection is provably faster than an exact projection.

This result summarizes multiple independent algorithms in this thesis. Among others, we give fast approximate projections for constraint sets defined by low-rank matrices, hierarchical sparsity, graph sparsity, group sparsity, and k-histograms (see Table 2.1 for details). Even in cases where an exact projection is NP-hard (such as graph sparsity and group sparsity), we can still approximately project in *nearly-linear time*.

Our three results show that we can optimize over a diverse collection of non-convex sets as long as we stay above the noise floor of the estimation problem. In particular, approximate projections provide a concrete new example of algorithmic problems that become significantly easier in a statistical setting. Although they solve the intermediate projection problems only approximately, they still provide optimal statistical guarantees in the overall estimation task. Our provable guarantees also show how PGD-like heuristics can succeed in practice even when applied to non-convex sets. Finally, we remark that our algorithms improve over prior work *both* in theory and in experiments.

1.2 Lower bounds for unconstrained optimization

Another large class of algorithmic problems arising in the data sciences are instances of unconstrained optimization. One important framework here is *empirical risk minimization* (ERM) [225]. For a set of labeled data points $(x_1, y_1), \ldots, (x_n, y_n)$ and a loss function L, the goal is to find a minimizer $\hat{\theta}$ of the empirical risk

$$L_n(\theta) \;=\; \sum_{i=1}^{n} L(\theta, x_i, y_i)\;.$$

The loss function L encodes how well the parameters θ fit the data point (x, y). By varying the loss function and the underlying classifier, the ERM paradigm encompasses widely used learning techniques such as logistic regression, support vector machines (SVM), and artificial neural networks [207].

Over the past decade, stochastic gradient descent (SGD) has emerged as a powerful tool for approximately solving ERM problems [52, 188]. The main idea of SGD is to work with *approximate* gradients obtained from only a *subset* of the data points in each iteration (as opposed to the exact gradient from all n data points). Especially for large datasets, a single iteration of SGD can be multiple orders of magnitude faster than a single iteration of gradient descent.

For convex ERM problems (and some non-convex ones), it is now understood that SGD quickly finds solutions that *generalize*, i.e., achieve good classification accuracy on new unseen data [113, 173, 180]. Nevertheless, the approximate gradients used in SGD come at a cost. While they allow the algorithm to make rapid initial progress, SGD only converges slowly to the true minimizer of the empirical risk. This leads to the following issues:

- It is often unclear if a low classification accuracy is a failure of *optimization* (e.g., an inaccurate ERM solution) or *learning* (wrong choice of model or insufficient data).
- Achieving good performance with SGD often requires fine-tuning various parameters such as the step size [239].
- Seemingly benign questions such as step size schedules or adaptive step size selection are still the subject of current research and actively debated in the community [94, 141, 187, 216, 239].

This unreliable behavior of SGD can add significant overhead to the overall training time. It also limits the effective use of machine learning to a small number of expert researchers. In contrast to SGD, an ideal algorithm should reliably and quickly converge to the empirical risk minimizer. The fact that we are stuck with tuning SGD raises a natural question:

What are the fundamental computational limits of empirical risk minimization?

The second part of our thesis addresses this question in both convex and non-convex settings.

Convex ERM. How would an ideal ERM algorithm look like? Formally, our goal is to find an ε-approximate solution to the ERM problem, i.e., an estimate $\hat{\theta}$ that is ε-close to an optimal solution θ^*:

$$L_n(\hat{\theta}) - L_n(\theta^*) \ \leq \ \varepsilon \ .$$

An ideal algorithm would have a time complexity of $O(n \log {}^1\!/\!_\varepsilon)$. This running time implies that the algorithm would return a high-accuracy solution (due to the $\log {}^1\!/\!_\varepsilon$ dependence) with only a small cost per iteration (the linear dependence on n). Note that it is possible to achieve both goals individually. So-called interior point methods have a $\log {}^1\!/\!_\varepsilon$ dependence but usually require expensive iterations with a cost of $O(n^3)$ [40, 54, 174, 175]. In constrast, SGD has cheap iterations but only a slow convergence rate where the accuracy parameter enters as ${}^1\!/\!_\varepsilon$ (without the logarithm) [208]. Is it possible to achieve the best of both worlds?

We utilize results from fine-grained complexity to show that such an ideal algorithm is probably impossible for two important ERM problems. In particular, we build on the *Strong Exponential Time Hypothesis (SETH)*, which can be seen as a natural strengthening of the classical P vs NP conjecture [123, 124, 226].

Result 4. Assuming SETH, no algorithm can solve kernel support vector machines and kernel ridge regression to high accuracy in subquadratic time.

From a practical perspective, a quadratic running time can be burdensome or even prohibitive for datasets with $n = 10^6$ or $n = 10^7$ data points, which are now increasingly common. So unless a new algorithm cleverly exploits extra problem-specific structure, we have to live with the downsides of approximate algorithms for large datasets. This shows that our reliance on approximations in statistical settings is not only a matter of convenience but due to fundamental computational limits. Moreover, our result justifies the use of SGD and other approximation methods (e.g., random features or the Nyström method [179, 234]) for large-scale kernel problems.

Non-convex ERM. Finding the exact minimizer of the empirical risk is NP-hard even for simple non-convex problems [47]. But in practice, SGD has become a surprisingly effective tool for non-convex problems such as training neural networks. While the success of SGD is not yet well understood, can we give evidence that SGD – as currently employed – is in some sense optimal?

When we run SGD to train a large neural network, the main computational burden is computing the gradient of the loss with respect to the network parameters. This is the problem commonly solved by the seminal *backpropagation* algorithm that is now implemented in all major deep learning frameworks [195]. The standard approach is to compute gradients for a *batch* of examples and then to update the network parameters with the average gradient. Since data batches now contain hundreds or even thousands of data points, it is important to understand how quickly we can compute batch gradients [111].

For a neural network with p hidden units, computing the gradient of a single data point via backpropagation takes $O(p)$ time. With a batch of size n, the standard approach is to simply compute the gradient of each data point individually and then to average the result. This has a *rectangular* time complexity of $O(n \cdot p)$. But from an information-theoretic perspective, the algorithm has all the information for computing the batch gradient after reading the input (the network and the data points) in time $O(n + p)$. So is it possible to improve over the rectangular running time? We again leverage the fine-grained complexity toolkit to prove that the rectangular running time is (likely) optimal.

Result 5. Assuming SETH, approximately computing the gradient of a two-layer network with rectified linear units (ReLU non-linearities) requires $O(n \cdot p)$ time.

Our hardness result suggests that it may be impossible to significantly improve over backpropagation for computing gradients in neural networks. As in kernel methods, this perspective points towards an inherent tension between the representational power of a learning approach and the resulting computational hardness. Furthermore, our result provides an explanation for why backpropagation has been an unchanged core element in deep learning since its invention.

1.3 Nearest neighbor search

The third algorithmic problem we focus on is nearest neigbor search (NNS). Given a dataset of d-dimensional points $P = \{p_1, \ldots, p_n\}$ and a distance function $\delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, we first build a data structure that then allows us to answer *nearest neighbor queries* efficiently. Specifically, for a given query point $q \in \mathbb{R}^d$, our goal is to quickly find its nearest neighbor

$$\arg\min_{p \in P} \ \delta(p, q) \ .$$

NNS is a fundamental tool for processing large datasets and is commonly used for exploratory data analysis and near-duplicate detection. Moreover, large-scale recommender systems and search engines rely on NNS as an integral component.

A simple baseline for NNS is a linear scan: when a new query q arrives, simply compute the distance $\delta(p_i, q)$ to all n points without building a data structure beforehand. We then pick the closest point p_i, which can be done in $O(n)$ total time. In fact, this baseline is (nearly) optimal in the general case: there are conditional lower bounds proving that a strongly sub-linear query time is impossible [15, 84, 192, 235]. But as datasets continue to grow, it is increasingly important to design fast algorithms with a *sublinear* query time. Indeed, there is now a long line of work on data structures for NNS that enable fast query times in practice. Popular examples include k-d trees [41], Locality Sensitive Hashing (LSH) [112, 127], product quantization [134], and graph-based approaches [159]. Their impressive empirical performance raises a natural question:

Why do nearest neighbor algorithms work well in practice?

The theory of LSH offers an explanation via *distance gaps*. To be concrete, let us assume that the query point q is c times closer to its nearest neighbor p^* than to all other data points. We can formalize this as

$$\delta(q, p^*) \ \leq \ \frac{1}{c} \cdot \delta(q, p) \qquad \text{for } p \in P \setminus \{p^*\} \ .$$

Intuitively, large distance gaps should lead to an easier nearest neighbor problem since there is more “contrast” between the nearest neighbor and an average data point. Indeed, the seminal hyperplane hash yields a data structure with query time $O(n^{1/c})$ for the widely used angular distance (also known as cosine similarity) [70]. This sublinear query time is responsible for the good empirical performance of the hyperplane hash. It also shows how the linear time hardness of NNS only appears below the “noise floor” given by the distance gaps.

However, more recent NNS techniques have improved significantly and now outperform the hyperplane hash by an order of magnitude on standard benchmarks [23, 43]. So are distance gaps still sufficient to explain the good performance of NNS methods in practice? And even if distance gaps still suffice, it is not clear *why* a dataset would have large distance gaps to begin with. This thesis addresses both of these questions.

Bridging theory and practice of locality-sensitive hashing. Can distance gaps provide faster query times than the $O(n^{1/c})$ achieved by the hyperplane hash? Over the past decade, theoretical research has shown that more advanced hash functions achieve a $O(n^{1/c^2})$ query time [16, 18, 19]. The quadratically better exponent could potentially lead to faster query times in practice as well. But unfortunately, this has not happened so far.

The main bottleneck of the theoretically motivated hash functions is the time it takes to compute the hash function itself. Here, a key feature of the hyperplane hash becomes apparent: computing a single hash function costs only $O(d)$ time, while the theoretically better hash functions have higher time complexities such as $O(d^2)$. Is it possible to combine the good sensitivity (i.e., small query time exponent) of more advanced hash functions with the fast computation time of the hyperplane hash?

Our starting point is the *cross-polytope* (CP) hash originally introduced by Terasawa and Tanaka [219]. The authors conduct insightful experiments demonstrating that the CP hash has a good sensitivity. But they do not provide theoretical proofs for this phenomenon. Moreover, their version of the CP hash also suffers from the slow $O(d^2)$ time to compute a hash function and consequently was not adopted in practice. Our work addresses these shortcomings. A key ingredient are ideas from randomized dimensionality reduction that enable us to compute the CP hash in $O(d \log d)$ time [12].

Result 6. The cross-polytope hash provably achieves a $O(n^{1/c^2})$ query time in the locality-sensitive hashing framework. Empirically, it also improves over the hyperplane hash by a factor of 4× to 10× for datasets of size 10^6 to 10^8.

We converted our research implementation of the CP hash into an open source library called FALCONN [185].[1] When we released FALCONN in 2015, it was the fastest library in the standard NNS benchmark and improved over the prior state-of-the-art by a factor 2× to 3× [43]. Moreover, it was the first LSH-based technique that empirically improved over the hyperplane hash in more than a decade. While a graph-based approach surpassed FALCONN in the following year, our library is still competitive with the state-of-the-art [23]. This demonstrates that distance gaps can indeed provide a good explanation for the empirical success of NNS. Moreover, FALCONN has become a popular library on GitHub and found users in both academia and industry [185].

Understanding distance gaps in learned embeddings. LSH explains how distance gaps enable fast NNS in practice. But where do these distance gaps come from? In the past, feature vectors for images, text, or speaker identities were usually designed by hand. More recently, such hand-crafted embeddings are often replaced by embeddings trained from data. Why do these trained embeddings exhibit good distance gaps? And can we specifically train embeddings for more efficient NNS?

We investigate this question for embeddings derived from artificial neural networks. This is an important class of models for NNS because the aforementioned types of data are now often embedded into vector spaces via neural networks. For our exposition here, it suffices to focus on the *cross-entropy* loss used to train most contemporary networks. After some

[1] The name stands for "Fast Approximate Look-up of COsine Nearest Neighbors".

simplifications, we can write the loss for an example x and class y as

$$L(x, y) \;\approx\; \angle(\phi(x), v_y) \tag{1.2}$$

where $\phi(x) : \mathbb{R}^d \to \mathbb{R}^k$ is the embedding computed by the network and v_y is the weight vector for class y.[1] We write $\angle(\cdot, \cdot)$ to denote the angle between two vectors. From the perspective of NNS, achieving a small angle $\angle(\phi(x), v_y)$ for all points of interest directly corresponds to a good distance gap.

We conduct a range of experiments on different neural networks to understand the angle $\angle(\phi(x), v_y)$ achieved by their embeddings. Indeed, we find that the training process always leads to distance gaps in the resulting embeddings. However, these gaps are sometimes too small for NNS methods to offer a meaningful improvement over a simple linear scan. To address this issue, we introduce changes to the training process that reliably lead to larger distance gaps in the resulting embedding. Overall, we obtain the following result.

Result 7. We can train a neural network embedding of 800,000 videos so that standard NNS methods are 5× faster on the resulting dataset.

In addition to the NNS speed-up, our changes do not negatively affect the classification accuracy (and sometimes lead to higher accuracy). Overall, our experiments explain how distance gaps naturally arise in neural network training because the cross-entropy loss encourages angular distance gaps.

[1] Intuitively, v_y is a prototypical example for class y under the embedding ϕ.

Part I

Optimizing Over Non-Convex Cones

Chapter 2

Projected Gradient Descent For Non-Convex Cones

2.1 Introduction

The goal in constrained optimization is to minimize a (often convex) objective function $f : \mathbb{R}^d \to \mathbb{R}$ over a given *constraint set* $\mathcal{C} \subseteq \mathbb{R}^d$, i.e., to find a point $x \in \mathcal{C}$ that minimizes the function value $f(x)$ among all points in the set $\mathcal{C}$. Formally, we want to solve the problem

$$\begin{array}{c} \text{minimize } f(x) \\ \text{subject to } x \in \mathcal{C} \ . \end{array}$$

This fundamental paradigm has found many applications in the sciences and engineering, including control theory, finance, signal processing, statistics, circuit design, machine learning, etc. [40, 54, 174, 175]. As a result, the underlying techniques such as the Simplex algorithm, interior point methods, and projected gradient descent have become some of the most widely used algorithmic primitives and are available in broad range of commercial solvers and optimization libraries.

A crucial question in constrained optimization is what constraint sets $\mathcal{C}$ allow for efficient algorithms. Over the past few decades, the optimization community has developed a rich classification based on different types of *cones*.[1] The most prominent examples are the non-negative orthant (giving rise to linear programs), the second-order cone (underlying for instance quadratic programs), and the cone of positive semidefinite matrices (which forms the basis of semidefinite programming). While there are more cones for which we have polynomial time solvers (e.g., the exponential cone or the hyperbolicity cone), the overarching theme is that all of these cones are *convex*. This raises the natural question:

Can we efficiently solve optimization problems over non-convex cones?

[1] Recall that a cone is a set that is closed under multiplication with non-negative scalars. So if we have $x \in \mathcal{C}$, then we also have $\alpha \cdot x \in \mathcal{C}$ for any $\alpha \geq 0$.

Constrained estimation problems. Driven by applications in the data sciences, the past ten years have seen a large amount of research devoted to specific non-convex cones. The canonical example here is the set of *sparse* vectors, i.e., vectors with only a small number of non-zero coefficients. Sparsity arises in a broad range of applications. In compressive sensing, the goal is to recover a sparse vector (e.g., the image of a patient's spine) from a small number of linear observations. Examples of linear observations are measurements given by a magnetic resonance imaging (MRI) device [157]. In statistics, we often want to find a small number of important features (e.g., genes relevant to a disease) from a high-dimensional dataset such as gene expression data [222]. In both cases, we can write the problem as a linear model of the form

$$y \;=\; X\theta^* + e$$

where $\theta^* \in \mathbb{R}^d$ is the unknown sparse vector and $X \in \mathbb{R}^{n\times d}$ is the measurement matrix (in signal processing) or design matrix (in statistics). The vector $y \in \mathbb{R}^n$ contains the linear observation corrupted by noise $e \in \mathbb{R}^n$. Given the observations y and the matrix X, the goal is now to find a good estimate $\hat{\theta} \approx \theta^*$ up to the inherent noise level.

Since we are working with a linear model, a natural approach to solve this problem is via *constrained* least squares:

$$\begin{aligned} \text{minimize } & \|y - X\theta\|_2^2 \\ \text{subject to } & \|\theta\|_0 \;\leq\; s \end{aligned}$$

where we write $\|\theta\|_0$ to denote the sparsity or ℓ_0-"norm" of θ (i.e., the number of non-zero coefficients in θ). The parameter s is our desired upper bound on the sparsity of the solution. The resulting estimator $\hat{\theta}$ has essentially optimal statistical performance, but the computational side is not immediately clear.

It is easy to see that sparse vectors form a *non-convex* cone.[1] An unfortunate consequence of the non-convexity is that solving least squares with a sparsity constraint is already NP-hard [168]. At first, this might suggest that we cannot hope for an algorithm with strong guarantees. But in a variety of applications, several methods usually find good estimates $\hat{\theta}$. So what is going on here?

To explain this phenomenon, a rich body of work has investigated regularity conditions for the design matrix X. The most common assumptions take the form of the following pair of inequalities:

$$\ell\|\theta\|_2^2 \;\leq\; \|X\theta\|_2^2 \;\leq\; L\|\theta\|_2^2 \,,$$

which is assumed to hold for all sparse vectors θ and two fixed constants $0 < \ell \leq L$. Under this condition, it is possible to show that widely influential methods such ℓ_1-regularized least squares (also known as the Lasso [221]) and ℓ_1-minimization (Basis Pursuit [73]) provably find estimates $\hat{\theta}$ that are essentially optimal from a statistical point of view.

The step from the non-convex ℓ_0-constraint (sparsity) to the convex ℓ_1-norm has been generalized to a broader theme of *convex relaxations*. Another seminal instantiation of this approach are low-rank matrix problems, where the non-convex rank constraint is relaxed to

[1] A convex set is closed under taking convex combinations of its elements. But the convex combination of two s-sparse vectors with different supports can be up to $2s$-sparse.

the convex nuclear norm [59, 186]. Moreover, there convex relaxation frameworks such as atomic norms and decomposable regularizers [69, 171]. However, it is still unclear which non-convex cones are amenable to the relaxation approach, and whether the convex relaxation route is necessay for optimal statistical guarantees in the first place.

To address these questions, we make two main contributions.

2.1.1 Projected gradient descent for non-convex cones

Our first contribution is an extension of the classical projected gradient descent (PGD) algorithm to the non-convex setting outlined above. Similar to gradient descent, PGD is a first-order method that improves an objective function f iteratively via gradient steps. In order to incorporate the constraint set, PGD assumes that it has access to a *projection oracle* $P_{\mathcal{C}}$ that maps an arbitrary input to the closest point in the constraint set $\mathcal{C}$. With such a projection oracle, the PGD algorithm can be stated as a simple iterative procedure:[1]

```
for $i \leftarrow 1, \ldots, T$ :
    $\theta^{i+1} \leftarrow P_{\mathcal{C}}(\theta^i - \eta \cdot \nabla f(\theta^i))$ .
```

The classical theory for PGD analyzes convergence to the global minimizer for convex sets. More recent work from the past decade has also established estimation results for non-convex sets [50, 80, 102, 131, 170]. However, these results typically rely on stringent assumptions on the (restricted) condition number $\kappa = {}^{L}/_{\ell}$. It is known that non-convex PGD only matches convex relaxations in the regime where κ is very close to 1 [133]. In contrast, the convex relaxation approaches also achieve optimal statistical guarantees for arbitrary κ [231]. Especially in machine learning and statistics, one would instead prefer algorithms that reliably work for arbitrary condition numbers because the design matrix X is typically given by a data generating process we do not control. In contrast, signal processing settings usually have some freedom to co-design the measurement matrix with the estimation algorithm.

The gap between PGD and the convex relaxations is not due to a lack of better understanding – it is a real limitation of the algorithm. If we project onto the non-convex set $\mathcal{C}$ in every iteration, PGD can get stuck in local minima due to the "kinks" of the non-convex constraint set. Moreover, these local minima can correspond to incorrect solutions of the overall estimation problem. So a crucial question is whether we can modify PGD to escape such local minima (indeed, this exactly what happens implicitly in the $\kappa \approx 1$ regime mentioned above: a single gradient step already provides enough signal to escape local minima). One approach in this direction is to project onto significantly larger constraint sets in every iteration, but doing so leads to worse statistical performance since we now require our regularity condition on a much larger set [133].

Our new algorithm. To overcome this limitation posed by non-convex constraints, we introduce an extension of PGD that carefully explores the local neighborhood around the constraint set. This exploration takes the form of a new inner loop in PGD that allows the

[1] For simplicity, we ignore the choice of initialization θ^1 and step size η here.

iterates to leave the constraint set in a controlled way. At a high level, this approach can be formalized as follows:

```
for $i \leftarrow 1, \ldots, T$:                                   ▷ Standard outer PGD loop
    $\tilde{\theta}^{i,1} \leftarrow \theta^i$                     ▷ Inner loop starts at current iterate
    for $j \leftarrow 1, \ldots, T_{\text{inner}}$:
        $\tilde{\theta}^{i,j+1} \leftarrow \tilde{\theta}^{i,j} - \eta \cdot P_{\mathcal{C}}(\nabla f(\tilde{\theta}^{i,j}))$
    $\theta^{i+1} \leftarrow P_{\mathcal{C}}(\tilde{\theta}^{i,T_{\text{inner}}+1})$
```

The crucial trade-off in the inner exploration phase is between the non-convex constraint and our statistical "budget". If we explore over a larger space, it becomes easier to escape the local minima caused by the non-convex kinks in the constraint set. However, exploring over a larger space also means that we require regularity conditions over a larger set, which in turn means a worse statistical rate (more regularity requires more samples). On the other hand, if we explore only over a small region (e.g., only a single gradient step as in vanilla PGD), we risk getting stuck in a local minimum.

By introducing an inner loop inspired by the Frank Wolfe algorithm, we show that we can carefully balance these two desiderata. The resulting guarantee demonstrates that we can match the optimal guarantees of convex relaxations for *any* non-convex cone and any condition number κ. For constraint sets where no convex relaxation is known (see the next subsection), our approach also goes beyond the convex relaxation approaches. Overall, the non-convex PGD perspective allows us to unify and extend a broad range of prior work in this area. Moreover, our provable guarantees justify the heuristic use of PGD with non-convex sets in practice and offer an explanation why this approach can work well in spite of the non-convexity.

2.1.2 Approximate projections

Our algorithm from the previous subsection is a powerful tool for optimizing over non-convex cones – provided we have an efficient projection oracle $P_{\mathcal{C}}$. So have we really made progress? If projecting onto the non-convex set turns out to be an NP-hard problem, our overall algorithm is not efficient because it relies on a projection in every iteration. Hence it is important to understand the complexity of projecting onto our constraint sets of interest.

Recall that we started with the following type of optimization problem (again instantiated for the least squares loss):

$$\arg\max_{\theta \in \mathcal{C}} \|y - X\theta\|_2^2 \, . \tag{2.1}$$

PGD allows us to solve this problem via a sequence of projections. If we denote our current iterate (before the projection) with θ^{in}, computing $P_{\mathcal{C}}(\theta^{in})$ means solving a problem of the form:

$$\arg\max_{\theta \in \mathcal{C}} \|\theta^{in} - \theta\|_2^2 \, . \tag{2.2}$$

Note the main difference between Equations 2.1 and 2.2: we have removed the data matrix X from the optimization problem. While this certainly simplifies the situation, we are still left with a non-convex problem because $\mathcal{C}$ is a non-convex set.

For some constraint sets, the projection problem (2.2) is indeed much easier. For instance, we can project onto the set of sparse vectors with a simple thresholding operation in linear time (in contrast, recall that solving sparse least squares is NP-hard). Similarly, we can project onto the set of low-rank matrices via a singular value decomposition (SVD). However, for more complex constraint sets, the computational hardness often associated with non-convex sets quickly strikes back.

A concrete example is *graph sparsity* [118, 122]. In addition to the "vanilla" sparsity constraint, we now also require that our estimate obeys a graph connectivity constraint. For a given graph defined on our features, such a constraint enforces that the selected features form a connected component in the underlying graph. A concrete example of such a graph are protein-protein interaction networks in computational biology. From the biologist's point of view, graph-sparse solutions are more meaningful because they are more likely to correspond to a true biological process. But from the algorithmist's point of view, projecting onto the set of graph-sparse vectors is unfortunately an NP-hard problem.

Graph sparsity is not the only example of a constraint set that is hard to work with. Another example that has been intensively studied in the statistics literature is *group sparsity*, where the sparse solution consists of a few groups of variables [242]. Again, projecting on the set of group sparse vectors is NP-hard [33]. And even for sets with polynomial-time projections such as low-rank matrices or tree sparsity (also known as hierarchical sparsity), their *superlinear* running time can be a bottleneck in practice. While these problems are not NP-hard, improving the current running times of exact projections would still represent progress on fundament algorithmic problems. So how can we overcome this "projection bottleneck" in PGD? Addressing this challenge is our second contribution in this context.

Tail approximations. When faced with a hard *exact* problem such as that in Equation 2.2, a standard approach in theoretical computer science is to study an *approximate* version of the problem instead. A natural way to relax the projection guarantee is as follows: for an input vector θ^{in}, we are now content with an approximate projection $\theta \in \mathcal{C}$ satisfying the guarantee

$$\|\theta^{in} - \theta\|_2^2 \;\leq\; c_{\mathcal{T}} \cdot \min_{\theta' \in \mathcal{C}} \|\theta^{in} - \theta'\|_2^2 \;, \tag{2.3}$$

where $c_{\mathcal{T}} \geq 1$ is the approximation ratio (for $c_{\mathcal{T}} = 1$, this is the same guarantee as an exact projection). In the following, we will refer to such a notion of projection as a *tail approximation*.

Using an approximate projection of this form is a natural approach for speeding up PGD. After all, PGD is an iterative algorithm where every step contributes only a small improvement to the solution quality. So insisting on an exact projection seems like an overly burdensome intermediate goal. But surprisingly, this intuition is false. As we show in Section 2.5, it is possible to prove that for any $c_{\mathcal{T}} > 1$, there are problem instances where PGD provably fails. While this does not preclude the success of heuristics in practice, ideally we would like to understand whether it is possible to leverage approximation algorithms in a way that still provides strong performance guarantees.

Head approximations. We show that this is possible by utilizing a second, complementary notion of approximation: For an input vector θ^{in}, our goal is now to find a unit vector

$\theta \in \mathcal{C} \cap \mathbb{S}^{d-1}$ such that

$$\langle \theta^{in}, \theta \rangle \;\geq\; c_{\mathcal{H}} \cdot \max_{\theta' \in \mathcal{C} \cap \mathbb{S}^{d-1}} \langle \theta^{in}, \theta' \rangle \tag{2.4}$$

where $0 < c_{\mathcal{H}} \leq 1$ is again the approximation ratio (and for $c_{\mathcal{H}} = 1$, this is also the same guarantee as an exact projection). In this thesis, we refer to this notion of projection as a *head approximation*. It is important to note that head and tail projections offer complementary approximation guarantees. As soon as $c_{\mathcal{H}}$ or $c_{\mathcal{T}}$ are strictly bounded away from 1, a tail approximation cannot be converted to a high-quality head approximation (or vice versa).

By combining head an tail projections in the right way, we can prove that PGD with approximate projections still converges to a statistically optimal solution in a broad range of estimation tasks (more specifically, under the same regularity conditions as in the previous subsection). Our approximation-tolerant extension of PGD is best illustrated in the well-conditioned regime ($\kappa \approx 1$), where we do not need the inner loop of the general algorithm. Denoting a tail approximation oracle with $\mathcal{T}$ and a head approximation oracle with $\mathcal{H}$, the main iterative step of PGD then becomes:

$$\theta^{i+1} \leftarrow \mathcal{T}_{\mathcal{C}}(\theta^i - \eta \cdot \mathcal{H}(\nabla f(\theta^i))) \; .$$

Compared to vanilla PGD, the main difference is that we now apply projections to both the gradient and the current iterate. Underlying this scheme is the insight that a tail approximation is only a useful guarantee when the input vector is already somewhat close to the constraint set. However, applying a full gradient update can lead us far outside the constraint set. To overcome this difficulty, we instead only take a *head-projected* gradient step. Clearly, this allows us to stay relatively close to the constraint set $\mathcal{C}$. Moreover, we can show that the head-projected gradient step still makes enough progress in the function value so that the overall algorithm converges to a good soluton.

Instantiating our framework. While our approximation-tolerant version of PGD is an intriguing framework, at first it is not clear how useful the framework can be. To get an actual algorithm, we must instantiate the head and tail projection guarantees with concrete approximation algorithms for interesting constraint sets. Indeed, prior work on PGD with approximation projections could only instantiate their respective algorithms in restrictive settings [49, 82, 107, 108, 144, 145, 205]. To address this issue for our framework, we build a bridge between the continuous and discrete optimization worlds. By leveraging a wealth of results from the approximation algorithms literature, we construct new approximate projections for a broad range of constraint sets including tree sparsity, graph sparsity, group sparsity, and low-rank matrices (see Table 2.1). Even in cases where an exact projection is NP-hard (such as graph or group sparsity), our approximation algorithms run in *nearly-linear* time.

In all cases, our algorithms significantly improve over the running time of exact projections both in theory and in practice. While our theoretical analysis is not tight enough to preclude a moderate constant-factor increase in the sample complexity, the resulting estimators are still optimal in the $O(\cdot)$ sense. Experimentally, we also observe an interesting phenomenon: despite the use of approximate projections, the empirical sample complexity is essentially the same as for exact projections. Again, our results provide evidence that algorithmic problems become significantly easier in statistical settings. Moreover, they support the use of approximate projections for optimization problems in statistics and machine learning.

Constraint set	Best known time complexity of an **exact projection** (prior work)	Best known time complexity of an **approximate projection** (this thesis)
Sparsity	$O(d)$	—
Low-rank matrices	$O(d^{1.5})$	$\widetilde{O}(r \cdot d)$
Tree sparsity	$O(d^2)$ [64]	$\widetilde{O}(d)$
k-histograms	$O(k^2 \cdot d^{2.5})$	$\widetilde{O}(d^{1+\varepsilon})$
EMD sparsity	—	$\widetilde{O}(d^{1.5})$
Graph sparsity	NP-hard [118]	$\widetilde{O}(d)$
Group sparsity	NP-hard [33]	$\widetilde{O}(d)$

Table (2.1): The time complexity of exact and approximate projections for various constraint sets. In all cases beyond simple sparsity, approximate projections are significantly faster than the best known exact projections.

Further notes on the table: for simplicity, we use the $\widetilde{O}$ notation and omit logarithmic factors. The variable d denotes the size of the input (the dimension of the vector θ^*). For matrix problems, the stated time complexites are for matrices with dimension $\sqrt{d} \times \sqrt{d}$ so that the overall size (number of entries) is also d. The variable r denotes the rank of a matrix. For EMD sparsity, there is no known exact projection or hardness result. The ε in the exponent of the running time for k-histograms is an arbitrary constant greater than 0.

2.1.3 Outline of this part

In this chapter, we describe our core algorithms for optimizing over non-convex cones with approximate projections. Later chapters will then provide approximate projections for concrete constraint sets. The focus in this chapter is on our new two-phase version of projected gradient descent since it is the most general algorithm. We will also provide more specialized versions for the RIP setting ($\kappa \approx 1$) since we use them in later chapters.

2.2 Main result

Before stating our algorithm and its guarantee, we formally define our problem of interest.

2.2.1 Setup

Recall that our goal is to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$ over a given *constraint set* $\mathcal{C} \subseteq \mathbb{R}^d$. Formally:

$$\begin{aligned} &\text{minimize } f(\theta) \\ &\text{subject to } \theta \in \mathcal{C} \ . \end{aligned}$$

When solving such a constrained optimization problem, there are two main questions:

- What properties does the objective function f satisfy?
- How do we access the constraint set $\mathcal{C}$?

Restricted strong convexity and smoothness. Regarding the first question, we assume *restricted* strong convexity and smoothness as is commonly done in statistical estimation problems [9]. In contrast to the classical (global) notions of strong convexity and smoothness, the restricted counterparts only hold over the constraint set (or a suitably relaxed version of the constraint set). This weakened assumption is crucial because the concentration phenomena in statistical settings are only sufficient to guarantee strong convexity and smoothness over a subset of the entire parameter space.

Definition 2.1 (Restricted smoothness)**.** *Let C be a cone. Then a differentiable convex function f has restricted smoothness L over C if, for every point $\theta' \in \mathbb{R}^d$ and every vector $\theta \in C$,*

$$\langle \nabla f(\theta' + \theta) - \nabla f(\theta'), \theta \rangle \leq L \|\theta\|_2^2 \ .$$

Note that for a quadratic function $\frac{1}{2}\|X\theta - y\|_2^2$, this is equivalent to $\|X\theta\|_2^2 \leq L\|\theta\|_2^2$ for every $\theta \in C$.

Definition 2.2 (Restricted strong convexity)**.** *Let C be a cone. Then a differentiable convex function f has restricted strong convexity ℓ over C if, for every point $\theta' \in \mathbb{R}^d$ and every vector $\theta \in C$,*

$$f(\theta' + \theta) \geq f(\theta') + \langle \nabla f(\theta'), \theta \rangle + \frac{\ell}{2}\|\theta\|_2^2 \ .$$

For a quadratic function $\frac{1}{2}\|X\theta - y\|_2^2$, this is equivalent to $\|X\theta\|_2^2 \geq \ell\|\theta\|_2^2$ for every $\theta \in C$.

Approximate projections. Next, we address the second question: how do we access the constraint set $\mathcal{C}$? To this end, we now introduce our notions of "head" and "tail" approximations. These relaxed projections are the only way our algorithm 2Phase-PGD interfaces with the constraint sets.[1]

Definition 2.3 (Head approximation)**.** *Let $C^\star, C_\mathcal{H} \subseteq \mathbb{R}^d$ be two cones and let $c_\mathcal{H} \in \mathbb{R}$. Then an $(C^\star, C_\mathcal{H}, c_\mathcal{H})$-head approximation satisfies the following property. Given any vector*

[1] To avoid confusion: in this thesis, $C^\star$ does not denote a polar cone, but instead the constraint set corresponding to the solution θ^*.

$g \in \mathbb{R}^d$, the head approximation returns a unit vector $\theta \in C_{\mathcal{H}} \cap \mathbb{S}^{d-1}$ such that

$$\langle g, \theta \rangle \;\geq\; c_{\mathcal{H}} \cdot \max_{\theta' \in C^\star \cap \mathbb{S}^{d-1}} \langle g, \theta' \rangle \; .$$

Definition 2.4 (Tail approximation)**.** *Let $C^\star, C_{\mathcal{H}} \subseteq \mathbb{R}^d$ be two cones and let $c_{\mathcal{T}} \in \mathbb{R}$. Then an $(C^\star, C_{\mathcal{T}}, c_{\mathcal{T}})$-tail approximation satisfies the following property. Given any vector $\theta^{in} \in \mathbb{R}^d$, the tail approximation returns a vector $\theta \in C_{\mathcal{T}}$ such that*

$$\|\theta^{in} - \theta\|_2 \;\leq\; c_{\mathcal{T}} \cdot \min_{\theta' \in C^\star} \|\theta^{in} - \theta'\|_2 \; .$$

These definitions are the main link to the later chapters of this thesis. It is important to note that head and tail are two complementary notions of approximate projections. In general, a good head approximation cannot be converted to a good tail approximation (or vice versa). For $c_{\mathcal{H}} = 1$ or $c_{\mathcal{T}} = 1$, the respective definition of an approximate projection is equivalent to an exact one.

In addition to a relaxed approximation guarantee ($c_{\mathcal{H}} < 1$ and $c_{\mathcal{T}} > 1$, respectively), the above definitions also allow for approximation in the output sets $C_{\mathcal{H}}$ and $C_{\mathcal{T}}$. As long as $C_{\mathcal{H}}$ and $C_{\mathcal{T}}$ are comparable to $C^\star$ (say sparsity $2 \cdot s$ instead of sparsity s), this relaxation affects the sample complexity only by constant factors, yet enables polynomially faster algorithms in cases such as graph sparsity.

2.2.2 Main results

Before we proceed to our main result, we briefly introduce some additional notation. For two sets C_1 and C_2, we denote their Minkowski sum by $C_1 + C_2$. For an integer m, we write $m \times C$ for the m-wise Minkowski sum of a set with itself (i.e., $C + C + C + \cdots + C$ a total of m times).

With this notation in place, we now state our main theorem.

Theorem 2.5 (2Phase-PGD)**.** *There is an algorithm* 2Phase-PGD *with the following properties. Assume that* 2Phase-PGD *is given*

- *a $(C^\star, C_{\mathcal{T}}, c_{\mathcal{T}})$-tail approximation such that $\theta^* \in C^\star$, and*
- *a $(C^\star - C_{\mathcal{T}}, C_{\mathcal{H}}, c_{\mathcal{H}})$-head approximation.*

Then let

$$k \;=\; \Theta\left(\frac{(1 + c_{\mathcal{T}})^2 \cdot L_{\mathcal{H}}}{c_{\mathcal{H}}^2 \cdot \ell_{all}}\right)$$

and assume that f has restricted smoothness $L_{\mathcal{H}}$ over $C_{\mathcal{H}}$ and restricted strong convexity ℓ_{all} over the sum of $C^\star$ with the negations of $C_{\mathcal{T}}$ and k copies of C, i.e., $\ell_{all} = \ell_{C^\star - C_{\mathcal{T}} - k \times C_{\mathcal{H}}}$.

For a given $\varepsilon > 0$ and $R > \|\theta^*\|_2$, 2Phase-PGD *then returns an estimate $\hat{\theta}$ such that*

$$\|\hat{\theta} - \theta^*\|_2 \;\leq\; \max\left(64 \cdot \frac{(1 + c_{\mathcal{T}})^2}{\ell_{all}} \max_{\theta' \in (C^\star - C_{\mathcal{T}} - 1/c_{\mathcal{H}}^2 \times C_{\mathcal{H}}) \cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle, \, \varepsilon R\right) \; .$$

The time complexity of 2Phase-PGD *is dominated by* $O(\log 1/\varepsilon)$ *calls to the tail approximation and* $O(k \log 1/\varepsilon)$ *calls to the head approximation and the gradient oracle, respectively.*

Since the theorem above is somewhat technical, we briefly instantiate it in the standard compressive sensing setting to illustrate the bounds. Let $C^\star$, $C_\mathcal{H}$, $C_\mathcal{T}$ each be the set of s-sparse vectors. Since we can exactly project onto s-sparse vectors in linear time, we have $c_\mathcal{H} = c_\mathcal{T} = 1$. Hence our algorithm requires restricted smoothness and strong convexity only over sets that are $O(s)$-sparse, for which we can invoke standard results for RIP matrices [101]. The RIP setting also implies that $\ell_{all} \approx \ell_\mathcal{H} \approx 1$. Together, this yields the standard bound $\|\hat{\theta} - \theta^*\|_2 \leq O(\|e\|_2)$ for a noise vector e after a sufficient number of iterations (so that the εR terms is negligible).

2.2.3 Related work

Due to the vast literature on constrained optimization in statistical settings, we only compare to the most closely related works in detail. In this chapter, we focus on general PGD-style algorithms. Related work for specific constraint sets can be found in the respective chapters.

As mentioned before, there is a long line of work on instantiations of PGD with specific non-convex sets. This includes Iterative Hard Thresholding [50], Compressive Sampling Matching Pursuit [170], and the Singular Value Projection algorithm [131], among others [80, 102]. The work here is clearly inspired by these algorithms. In particular, a guiding question for this thesis was how general the non-convex PGD phenomenon is. Our contributions go beyond these – arguably by now classical – algorithms in three important ways:

- Our algorithm works for *arbitrary condition numbers*, while the aforementioned work is for the well-conditioned regime ($\kappa \approx 1$) that is common in compressed sensing.
- Our algorithm applies for *arbitrary non-convex cones* as opposed to special constraint sets such as sparse vectors or low-rank matrices.
- Our algorithm requires only *approximate projections*, while the aformentioned methods require exact projections. Hence our algorithm can be used in settings where an exact projection is NP-hard.

More recently, Oymak, Recht, and Soltanolkotabi [177] also consider projected gradient descent for general constraint sets that can be non-convex. However, their goals are somewhat different from ours: the authors focus on establishing sharp constants for isotropic measurement setups or data matrices, which corresponds to the well-conditioned regime given by the restricted isometry property. In contrast, our focus is on the regime where the condition number can be arbitrary. Moreover, our algorithm works with approximate projections.

Jain, Tewari, and Kar [133] also consider iterative thresholding methods (i.e., projected gradient descent) in the regime where the condition number can be arbitrary. However, the sample complexity does not match the optimal rates achieved by the Lasso estimator because the statistical rate has a quadratic dependence on the condition number. Our two-phase variant of projected gradient descent addresses this shortcoming. Moreover, our algorithm works with approximate projections and arbitrary conic constraint sets.

2.2.3.1 Approximate projections

Prior to this thesis, several efforts have been made to enable compressive sensing recovery for structured sparse signals with approximate projection oracles. Blumensath [49] discusses a projected Landweber-type method that succeeds even when the projection oracle is approximate. However, the author assumes that the projection oracle provides an *ϵ-additive* tail approximation guarantee. In other words, for any given $\theta^{in} \in \mathbb{R}^d$, their approximation oracle returns a $\theta \in \mathcal{C}$ satisfying:

$$\|\theta^{in} - \theta\|_2 = \min_{\theta' \in \mathcal{C}} \|\theta^{in} - \theta'\|_2 + \varepsilon \tag{2.5}$$

for some parameter $\varepsilon > 0$. Under such conditions, there exists an algorithm that returns an estimate $\hat{\theta}$ within an $O(\epsilon)$-neighborhood of the optimal solution θ^*. However, approximation oracles that achieve low additive approximation guarantees are rather rare. Note that for cones it is always possible the scale the input so that an additive approximation is essentially an optimal projection.

On the other hand, the works [144, 145] assume the existence of a head-approximation oracle similar to our definition and develop corresponding signal recovery algorithms. However, these approaches only provide estimation guarantees with an additive error term that is not necessary statistically. For a standard sparsity constraint, this error term has the form of $O(\|\theta^*_{\overline{\Omega}}\|)$, where Ω is the set of the s largest coefficients in θ^*. Therefore, their result is not directly comparable to our desired estimation guarantee.

Some more recent works have introduced the use of approximate projections for vectors that are sparse in a *redundant dictionary*. Giryes and Elad [107] present a sparse recovery algorithm for redundant dictionaries that succeeds with multiplicative approximation guarantees. However, their framework uses only the tail oracle and therefore is subject to the lower bound that we provide in Section 2.5. In particular, their guarantees make stringent assumptions on the maximum singular values of the sensing matrix X.

Davenport, Needell, and Wakin [82] introduce an algorithm called *Signal Space CoSaMP* (SSCoSaMP), which also assumes the existence of multiplicative approximate oracles. However, the assumptions made on the oracles are restrictive. Interpreted in the sparsity context, the oracles must capture a significant fraction of the optimal support in each iteration, which can be hard to achieve with provable guarantees. More recently, Giryes and Needell [108] propose a version of SSCoSaMP which succeeds with oracles satisfying both multiplicative head- and tail-approximation guarantees. Indeed, an earlier version of our algorithm is closely related to their work. However, our earlier algorithm already requires technically weaker conditions to succeed and our proof techniques are more concise. See Section 2.6.3 for a more detailed discussion on this topic.

2.3 Our algorithm

As mentioned before, our new algorith consists of two main parts. The inner loop of the algorithm makes progress in function value by taking head-projected gradient steps. We then combine these gradient steps in a way similar to the Frank-Wolfe algorithm (conditional gradient descent). Algorithm 1 contains the pseudo code for the inner loop.

The outer loop of our algorithm resembles projected gradient descent. Instead of taking a gradient step, we now run our inner loop. Then we project the current iterate back to the constraint set with a tail approximation. The pseudo code for this part can be found in Algorithm 2. This algorithm provides the guarantees stated in Theorem 2.5.

Algorithm 1 InnerLoop

```
function InnerLoop($\theta^{in}, r, \rho$)
    $k \leftarrow \left\lceil \frac{8L^{\mathcal{H}}}{\rho \cdot c_{\mathcal{H}}^2} \right\rceil$
    $\theta^{(0)} \leftarrow 0$
    for $i \leftarrow 0, \ldots, k-1$ do
        $g^{(i+1)} \leftarrow$ HeadApprox($-\nabla f(\theta^{in} + \theta^{(i)})$)
        $\tilde{\theta}^{(i+1)} \leftarrow \frac{r}{c_{\mathcal{H}}} \cdot g^{(i+1)}$
        $\theta^{(i+1)} \leftarrow \gamma^{(i)} \cdot \tilde{\theta}^{(i+1)} + (1 - \gamma^{(i)}) \cdot \theta^{(i)}$
    end for
    return $\theta^{in} + \theta^{(k)}$
end function
```

Algorithm 2 2Phase-PGD

```
function 2Phase-PGD($R, \varepsilon$)
    $T \leftarrow \left\lceil \log_2 \frac{1}{\varepsilon} \right\rceil$
    $\theta^{(0)} \leftarrow \vec{0}$
    $r^{(0)} \leftarrow R$
    for $t \leftarrow 0, \ldots, T-1$ do
        $\tilde{\theta}^{(t+1)} \leftarrow$ InnerLoop($\theta^{(t)}, r^{(t)}$)
        $\theta^{(t+1)} \leftarrow$ TailApprox($\tilde{\theta}^{(t+1)}$)
        if $\|\theta^{(t+1)} - \theta^{(t)}\|_2 > \frac{3}{2} r^{(t)}$ then return $\hat{\theta} \leftarrow \theta^{(t)}$
        end if
        $r^{(t+1)} \leftarrow \frac{r^{(t)}}{2}$
    end for
    return $\hat{\theta} \leftarrow \theta^{(T)}$
end function
```

2.4 The proof

This section contains the proof of our main algorithm. We begin with a simple consequence of the restricted smoothness condition.

Lemma 2.6 (Restricted smoothness)**.** *Let C be a cone, and let f be a differentiable convex function satisfying restricted smoothness L over C. Let $v = \sum_i v_i$, where for all i, $v_i \in C$ or $-v_i \in C$. Then for any x,*

$$f(x+v) \leq f(x) + \langle \nabla f(x), v \rangle + \frac{L}{2} \left(\sum_i \|v_i\| \right)^2 .$$

Proof. First, note that restricted smoothness over C implies restricted smoothness over $-C$, since one may substitute $x - v$ for x. Next, note that for a single $u \in C$, $f(x+u) \leq f(x) + \langle \nabla f(x), u \rangle + \frac{L}{2}\|u\|^2$ follows from integrating the derivative of the function $g(t) = f(x + tu)$. Finally, v is a convex combination of vectors in C and $-C$ with norm $\sum_i \|v_i\|$, so the result follows from convexity. □

Next, we abstractly define the properties we require for our inner loop. We call an oracle with these guarantees an InnerPhase.

Definition 2.7 (InnerPhase)**.** *Let $\theta^{in} \in C^{in}$ be the input vector in the cone C^{in} and let $\theta^* \in C^*$ be a vector in the cone C^*. Moreover, let $r \in \mathbb{R}$ be such that $\|\theta^{in} - \theta^*\| \leq r$.*

Let C be a cone, let k be a natural number, and let $\rho, c \in \mathbb{R}$. Then given θ^{in} and r, an (C, C^{in}, k, ρ, c)-InnerPhase returns a vector $\theta^{out} \in \mathbb{R}^d$ such that the following properties hold:

- *The output θ^{out} can be decomposed as*

$$\theta^{out} \;=\; b_0 + \sum_{i=1}^{k} b_i$$

where $b_0 \in C^{in}$ and for each i we have $b_i \in C$. Moreover,

$$\sum_{i=1}^{k} \|b_i\|_2 \;\leq\; c \cdot r \; .$$

- *The function value of the output satisfies*

$$f(\theta^{out}) \;\leq\; f(\theta^*) + \rho \cdot r^2 \; .$$

We now prove that our Algorithm 1 is indeed a valid InnerPhase (provided we have access to a head approximation oracle).

Theorem 2.8 (InnerLoop)**.** *There is an algorithm* InnerLoop *with the following properties. Assume* InnerLoop *is given a $(C^* - C^{in}, C_{\mathcal{H}}, c_{\mathcal{H}})$-head approximation such that $\theta^* \in C^*$. Moreover, assume that f obeys restricted smoothness $L^{\mathcal{H}}$ over $C_{\mathcal{H}}$.*

Then for any $\rho > 0$, InnerLoop *is a $(C_{\mathcal{H}}, C^{in}, k, \rho, 1/c_{\mathcal{H}})$-InnerPhase where $k = \lceil \frac{8 \cdot L^{\mathcal{H}}}{\rho \cdot c_{\mathcal{H}}^2} \rceil$. Moreover, its running time is dominated by k gradient queries, and k calls to the head approximation.*

The algorithm can be seen as an instantiation of the Frank-Wolfe or conditional gradient method and the analysis is therefore similar. The non-standard element here is our use of the head approximation rather than an exact projection oracle.

Proof. For the first property, we set $b_0 = \theta^{in}$, and the decomposition into b_i follows from the fact that each $\tilde{\theta} \in C_{\mathcal{H}}$ and $\|\tilde{\theta}^{(i+1)}\|_2 = \frac{r}{c_{\mathcal{H}}}$. What remains is to bound the function value.

To complete the algorithm, we need to define the step size $\gamma^{(i)}$. We will also define an auxiliary sequence of values δ_i for $i \geq 1$. Specifically, we set $\gamma^{(0)} = 1$, $\gamma^{(i)} = \frac{\delta_i}{2}$ for $i > 0$, $\delta_1 = 1$, and $\delta_{i+1} = \delta_i - \frac{\delta_i^2}{4}$ for $\delta \geq 1$.

To track the progress of the algorithm, we define $E_i = \frac{c_{\mathcal{H}}^2}{2L^{\mathcal{H}}r^2}(f(\theta^{in} + \theta^{(i)}) - f(\theta^*))$.

The key guarantee of a single step of the algorithm is that

$$E_{i+1} \leq (1 - \gamma^{(i)})E_i + (\gamma^{(i)})^2. \tag{2.6}$$

To prove this, we first note that the convexity of f gives

$$\langle \nabla f(\theta^{in} + \theta^{(i)}), \theta^* - \theta^{in} - \theta^{(i)} \rangle \leq f(\theta^*) - f(\theta^{in} + \theta^{(i)}).$$

Furthermore, by assumption, $\|\theta^* - \theta^{in}\|_2 \leq r$. Now, by the definition of a head approximation, and using the fact that $\theta^* - \theta^{in}$ is in $C^* - C^{in}$, we have

$$\langle \nabla f(\theta^{in} + \theta^{(i)}), g^{(i+1)} \rangle \leq \frac{c_{\mathcal{H}}}{r} \langle \nabla f(\theta^{in} + \theta^{(i)}), \theta^* - \theta^{in} \rangle$$

and hence

$$\langle \nabla f(\theta^{in} + \theta^{(i)}), \tilde{\theta}^{(i+1)} \rangle \leq \langle \nabla f(\theta^{in} + \theta^{(i)}), \theta^* - \theta^{in} \rangle$$

and

$$\langle \nabla f(\theta^{in} + \theta^{(i)}), \tilde{\theta}^{(i+1)} - \theta^{(i)} \rangle \leq f(\theta^*) - f(\theta^{in} + \theta^{(i)}).$$

Now, $\tilde{\theta}^{(i+1)}$ is a vector in $C^{\mathcal{H}}$ of norm $\frac{r}{c_{\mathcal{H}}}$, and $\theta^{(i)}$ is a convex combination of such vectors. Thus, applying Lemma 2.6

$$\begin{aligned} f(\theta^{in} + \theta^{(i+1)}) &= f(\theta^{in} + \theta^{(i)} + \gamma^{(i)}(\tilde{\theta}^{(i+1)} - \theta^{(i)})) \\ &\leq f(\theta^{in} + \theta^{(i)}) + \gamma^{(i)} \langle \nabla f(\theta^{in} + \theta^{(i)}), \tilde{\theta}^{(i+1)} - \theta^{(i)} \rangle + \frac{(\gamma^{(i)})^2}{2} \cdot \frac{4L^{\mathcal{H}}r^2}{c_{\mathcal{H}}^2} \\ &\leq f(\theta^{in} + \theta^{(i)}) + \gamma^{(i)}(f(\theta^*) - f(\theta^{in} + \theta^{(i)})) + (\gamma^{(i)})^2 \cdot \frac{2L^{\mathcal{H}}r^2}{c_{\mathcal{H}}^2} \\ &= (1 - \gamma^{(i)}) f(\theta^{in} + \theta^{(i)}) + \gamma^{(i)} f(\theta^*) + (\gamma^{(i)})^2 \cdot \frac{2L^{\mathcal{H}}r^2}{c_{\mathcal{H}}^2}. \end{aligned}$$

Plugging in the definition of E_i then gives (2.6).

Next, we get the invariant that $E_i \leq \delta_i$ for all $i \geq 1$. This is immediate from (2.6) for $i = 1$, and follows by induction for $i > 1$, using $(1 - \gamma^{(i)})\delta_i + (\gamma^{(i)})^2 = \delta_{i+1}$. Next, we can see that for all $i > 1$, $\delta_i \leq \frac{4}{i}$; this is immediate for $i = 1$, and follows by induction as $\frac{4}{i} - \frac{\left(\frac{4}{i}\right)^2}{4} = \frac{4}{i} - \frac{4}{i^2} \leq \frac{4}{i} - \frac{4}{i^2-1} = \frac{4}{i+1}$.

This finally gives us $E_k \leq \frac{4}{k}$; setting k as described gives

$$f(\theta^{in} + \theta^{(k)}) \leq f(\theta^*) + \rho \cdot r^2$$

as desired. □

Using the notion of an InnerPhase, we can now establish our guarantee for the OuterLoop. The following theorem also yields Theorem 2.5 above.

Theorem 2.9 (OuterLoop)**.** *There is an algorithm* OuterLoop *with the following properties. Assume that*

- OuterLoop *is given a* $(C^*, C_{\mathcal{T}}, c_{\mathcal{T}})$*-tail approximation such that* $\theta^* \in C^*$.
- OuterLoop *is also given a* $(C, C_{\mathcal{T}}, k, \rho, c)$*-InnerPhase with*

$$\rho \leq \frac{\ell_{all}}{32 \cdot (1 + c_{\mathcal{T}})^2}$$

where f *has restricted strong convexity* ℓ_{all} *over the sum of* C^* *with the negations of* $C_{\mathcal{T}}$ *and* k *copies of* C*, i.e.,* $\ell_{all} = \ell_{C^* - C_{\mathcal{T}} - k \times C}$.

For a given $\varepsilon > 0$ *and* $R > \|\theta^*\|_2$*,* OuterLoop *then returns an estimate* $\hat{\theta}$ *such that*

$$\|\hat{\theta} - \theta^*\|_2^2 \;\leq\; \max\left(64 \cdot \frac{(1 + c_{\mathcal{T}})^2}{\ell_{all}} \max_{\theta' \in (C^* - C_{\mathcal{T}} - c^2 \times C) \cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle,\; \varepsilon R\right) .$$

The time complexity of OuterLoop *is dominated by* $O(\log 1/\varepsilon)$ *calls of* InnerPhase *and the tail approximation.*

First, we analyze a single iteration of OuterLoop under some basic assumptions:

Lemma 2.10. *If*

$$\|\theta^{(t)} - \theta^*\|_2 \leq r^{(t)}$$

and

$$r^{(t)} \geq 16 \cdot \frac{(1 + c_{\mathcal{T}})^2}{\ell_{all}} \left(\max_{\theta' \in (C^* - C_{\mathcal{T}} - c^2 \times C) \cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle\right)$$

then

$$\|\theta^{(t+1)} - \theta^*\|_2 \leq r^{(t+1)}.$$

Proof. We satisfy the precondition for the guarantee of InnerPhase, and hence under this assumption we have that

$$f(\tilde{\theta}^{(t+1)}) \;\leq\; f(\theta^*) + \rho \cdot (r^{(t)})^2$$

and

$$\tilde{\theta}^{(t+1)} \;=\; b_0 + \sum_{i=1}^{k} b_i$$

for $b_0 \in C_{\mathcal{T}}$, $b_i \in C$ and $\sum_{i=1}^{k} \|b_i\|_2 \;\leq\; c \cdot r^{(t)}$.

$\tilde{\theta}^{(t+1)}$ is then contained in $C_{\mathcal{T}} + k \times C$, so $\theta^* - \tilde{\theta}^{(t+1)}$ is contained in $C^* - C_{\mathcal{T}} - k \times C$. We may then apply restricted strong convexity to obtain

$$f(\tilde{\theta}^{(t+1)}) \geq f(\theta^*) + \langle \nabla f(\theta^*), \tilde{\theta}^{(t+1)} - \theta^* \rangle + \frac{\ell_{all}}{2} \|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2$$
$$\|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2 \leq \frac{2}{\ell_{all}} (\rho \cdot (r^{(t)})^2 + \langle \nabla f(\theta^*), \theta^* - \tilde{\theta}^{(t+1)} \rangle)$$

Next, we will bound this by decomposing $\nabla f(\theta^*)$ into a sum of two pieces which we can bound separately. We define g_j to be any minimizer of $\|\nabla f(\theta^*) - g_j\|_2$ over g_j expressible as a difference of a nonnegative multiple of $\theta^* - b_0$ and j vectors in C. Note that g_j must be orthogonal to $\nabla f(\theta^*) - g_j$, and hence $\|\nabla f(\theta^*) - g_j\|_2^2 = \|\nabla f(\theta^*)\|_2^2 - \|g_j\|_2^2$. We define

$$G = \max_{\theta' \in (C^* - C_{\mathcal{T}} - c^2 \times C) \cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle.$$

For every g_j, $j \leq c^2$, we have by construction that $\|g_j\|_2 \leq G$. We define

$$e_j = \max_{\theta' \in C \cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*) - g_j, \theta' \rangle.$$

Then we define g to be any g_i, $0 \leq i < c^2$, that minimizes e_i.

Now, $\|\nabla f(\theta^*) - g_j\|_2^2 - \|\nabla f(\theta^*) - g_{j+1}\|_2^2 \geq e_j^2$. Therefore, the minimum e_i over $0 \leq i < c^2$ satisfies

$$\begin{aligned}
e_i^2 &\leq \frac{1}{c^2}(\|\nabla f(\theta^*) - g_0\|_2^2 - \|\nabla f(\theta^*) - g_{c^2}\|_2^2) \\
&\leq \frac{1}{c^2}(\|\nabla f(\theta^*)\|_2^2 - \|\nabla f(\theta^*) - g_{c^2}\|_2^2) \\
&= \frac{1}{c^2} \cdot \|g_{c^2}\|_2^2 \\
&\leq \frac{1}{c^2} \cdot G^2 \\
e_i &\leq \frac{G}{c}.
\end{aligned}$$

Additionally, $\langle \nabla f(\theta^*) - g, \theta^* - b_0 \rangle \leq 0$ by optimality of g. Putting this together we have

$$\begin{aligned}
\langle \nabla f(\theta^*), \theta^* - \tilde{\theta}^{(t+1)} \rangle &= \langle g + (\nabla f(\theta^*) - g), \theta^* - \tilde{\theta}^{(t+1)} \rangle \\
&= \left\langle g, \theta^* - \tilde{\theta}^{(t+1)} \right\rangle + \left\langle \nabla f(\theta^*) - g, \theta^* - \left(b_0 + \sum_{i=1}^{k} b_i \right) \right\rangle \\
&\leq \|g\|_2 \|\tilde{\theta}^{(t+1)} - \theta^*\|_2 + \langle \nabla f(\theta^*) - g, \theta^* - b_0 \rangle + \sum_{i=1}^{k} \langle \nabla f(\theta^*) - g, b_i \rangle \\
&\leq G \|\tilde{\theta}^{(t+1)} - \theta^*\|_2 + 0 + \frac{G}{c} \sum_{i=1}^{k} \|b_i\|_2 \\
&\leq G \|\tilde{\theta}^{(t+1)} - \theta^*\|_2 + G \cdot r^{(t)}.
\end{aligned}$$

Plugging that into our bound on $\|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2$, we get

$$\begin{aligned}
\|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2 &\leq \frac{2}{\ell_{all}} \left(\rho \cdot (r^{(t)})^2 + G \|\tilde{\theta}^{(t+1)} - \theta^*\|_2 + G \cdot r^{(t)} \right) \\
&\leq \frac{2}{\ell_{all}} \left(\rho \cdot (r^{(t)})^2 + \frac{4}{\ell_{all}} G^2 + \frac{\ell_{all}}{16} \|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2 + G \cdot r^{(t)} \right) \\
\frac{7}{8} \|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2 &\leq \frac{2}{\ell_{all}} \left(\rho \cdot (r^{(t)})^2 + \frac{4}{\ell_{all}} G^2 + G \cdot r^{(t)} \right).
\end{aligned}$$

By the assumption that $G \leq \frac{\ell_{all}}{16\cdot(1+c_{\mathcal{T}})^2} r^{(t)}$, and that $\rho \leq \frac{\ell_{all}}{32\cdot(1+c_{\mathcal{T}})^2}$, this gives

$$\begin{aligned}
\frac{7}{8}\|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2 &\leq \frac{1}{16\cdot(1+c_{\mathcal{T}})^2}\cdot(r^{(t)})^2 + \frac{1}{32(1+c_{\mathcal{T}})^4}\cdot(r^{(t)})^2 + \frac{1}{8\cdot(1+c_{\mathcal{T}})^2}\cdot(r^{(t)})^2 \\
&\leq \frac{7}{32\cdot(1+c_{\mathcal{T}})^2}\cdot(r^{(t)})^2 \\
\|\tilde{\theta}^{(t+1)} - \theta^*\|_2^2 &\leq \frac{1}{4\cdot(1+c_{\mathcal{T}})^2}\cdot(r^{(t)})^2 \\
\|\tilde{\theta}^{(t+1)} - \theta^*\|_2 &\leq \frac{1}{2\cdot(1+c_{\mathcal{T}})}\cdot r^{(t)}.
\end{aligned}$$

We can finally bound $\theta^{(t+1)}$ as

$$\begin{aligned}
\|\theta^{(t+1)} - \theta^*\|_2 &\leq \|\tilde{\theta}^{(t+1)} - \theta^*\|_2 + \|\theta^{(t+1)} - \tilde{\theta}^{(t+1)}\|_2 \\
&\leq \|\tilde{\theta}^{(t+1)} - \theta^*\|_2 + c_{\mathcal{T}}\|\tilde{\theta}^{(t+1)} - \theta^*\|_2 \qquad \text{(definition of tail approx)} \\
&= (1+c_{\mathcal{T}})\|\tilde{\theta}^{(t+1)} - \theta^*\|_2 \\
&\leq \frac{r^{(t)}}{2} = r^{(t+1)}
\end{aligned}$$

as desired. □

We can now prove our main theorem.

Proof of Theorem 2.9. If $r^{T-1} \geq 16 \cdot \frac{(1+c_{\mathcal{T}})^2}{\ell_{all}} \left(\max_{\theta' \in (C^* - C_{\mathcal{T}} - c^2 \times C)\cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle\right)$, then the assumption of Lemma 2.10 is, by induction, satisfied each iteration: for all t, $\|\theta^{(t)} - \theta^*\|_2 \leq r^{(t)}$. By the triangle inequality, $\|\theta^{(t)} - \theta^{(t+1)}\| \leq \frac{3}{2}r^{(t)}$ and so the early return is never triggered; thus the return value is $\hat{\theta} = \theta^{(T)}$ and satisfies $\|\hat{\theta} - \theta^*\|_2 \leq r^{(T)} \leq \epsilon R$.

Otherwise, let T' be the first integer such that

$$r^{(T')} < 16 \cdot \frac{(1+c_{\mathcal{T}})^2}{\ell_{all}} \left(\max_{\theta' \in (C^* - C_{\mathcal{T}} - c^2 \times C)\cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle\right),$$

and let T'' be either the step triggering the early return, or T if it is not triggered. By the same argument, we still have $\|\theta^{(T')} - \theta^*\| \leq r^{(T')}$, and $T'' \geq T'$, while for all $t < T''$ we have $\|\theta^{(t+1)} - \theta^{(t)}\|_2 \leq \frac{3}{2}r^{(t)}$. The return value is $\hat{\theta} = \theta^{(T'')}$. Applying the triangle inequality, we obtain

$$\begin{aligned}
\|\theta^{(T'')} - \theta^*\|_2 &\leq \|\theta^{(T')} - \theta^*\|_2 + \sum_{t=T'}^{T''-1} \|\theta^{(t+1)} - \theta^{(t)}\|_2 \\
&\leq \|\theta^{(T')} - \theta^*\|_2 + \frac{3}{2}\sum_{t=T'}^{\infty} r^{(t)} \\
&= 4 \cdot r^{(T')} \\
&\leq 64 \cdot \frac{(1+c_{\mathcal{T}})^2}{\ell_{all}} \left(\max_{\theta' \in (C^* - C_{\mathcal{T}} - c^2 \times C)\cap \mathbb{S}^{d-1}} \langle \nabla f(\theta^*), \theta' \rangle\right).
\end{aligned}$$

□

2.5 A counterexample

Our algorithm for optimizing over non-convex cones relies on *two* notions of approximate projections. A natural question is whether this is necessary. In particular, the tail approximation (see Definition 2.3) is arguably the natural relaxation of an exact projection. However, we will now show that even a small amount of approximation in the projection can lead to a failure of the overall PGD algorithm.

Consider the standard compressive sensing setting where the constraint is the set of all s-sparse vectors. Of course, finding the optimal model projection in this case is simple: for any vector θ^{in}, the oracle $P_s(\cdot)$ returns the s largest coefficients of θ in terms of absolute value. But to illustrate the failure of an approximate projection, we consider a slightly different oracle that is approximate in the following sense. Let $c_{\mathcal{T}}$ be an arbitrary constant and let $\mathcal{T}_s$ be a projection oracle such that for any $\theta \in \mathbb{R}^d$ we have:

$$\|\theta - \mathcal{T}_s(\theta)\|_2 \leq c_{\mathcal{T}} \|\theta - P_s(\theta)\|_2 \,. \tag{2.7}$$

We now construct an "adversarial" approximation oracle $\mathcal{T}_s$ that always returns $\mathcal{T}_s(\theta) = 0$ but still satisfies (2.7) for all vectors θ encountered during a run of PGD. In particular, we replace the exact projection in PGD with our adversarial oracle and start with the initial estimate $\theta^0 = 0$. We will show that such an oracle satisfies (2.7) for the first iteration of PGD. As a result, PGD with this adversarial oracle remains stuck at the zero estimate and cannot recover the true vector.

In the RIP regime, PGD with projection oracle $\mathcal{T}_s$ iterates

$$\theta^{i+1} \leftarrow \mathcal{T}_s(\theta^i + X^T(y - X\theta^i)) \,, \tag{2.8}$$

which in the first iteration gives

$$\theta^1 \leftarrow \mathcal{T}_s(X^T y) \,.$$

Consider the simplest case where the vector θ^* is 1-sparse with $\theta^*_1 = 1$ and $\theta^*_i = 0$ for $i \neq 1$, i.e., $\theta^* = e_1$ (the first standard basis vector). Given a measurement matrix X with $(\delta, O(1))$-RIP for small δ, PGD needs to perfectly recover θ^* from $X\theta^*$. It is known that random matrices $X \in \mathbb{R}^{n \times d}$ with $X_{i,j} = \pm 1/\sqrt{n}$ chosen i.i.d. uniformly at random satisfy this RIP for $n = O(\log d)$ with high probability [35].[1] We prove that our "adversarial" oracle $\mathcal{T}_s(\theta) = 0$ satisfies the approximation guarantee (2.7) for its input $a = X^T y = X^T X e_1$ with high probability. Hence, $\theta^1 = \theta^0 = 0$ and PGD cannot make progress. Intuitively, the tail $a - \mathcal{T}_s(a)$ contains so much "noise" that the adversarial approximation oracle $\mathcal{T}_s$ does not need to find a good sparse support for a and can simply return an estimate of 0.

Consider the components of the vector $a = X^T X e_1$: a_i is the inner product of the first column of X with the i-th column of X. Clearly, we have $a_1 = 1$ and $-1 \leq a_i \leq 1$ for $i \neq 1$. Therefore, $P_s(a) = e_1$ is an optimal projection and $\|a - P_s(a)\|_2^2 = \|a\|_2^2 - 1$. In order to show that the adversarial oracle $\mathcal{T}_s(a)$ satisfies the guarantee (2.7) with constant $c_{\mathcal{T}}$, we

[1] These are the so-called *Rademacher* matrices.

need to prove that:

$$\|a\|_2^2 \ \le\ c_{\mathcal{T}}^2(\|a\|_2^2 - 1)\ .$$

Therefore, it suffices to show that $\|a\|_2^2 \ge \frac{c_{\mathcal{T}}^2}{c_{\mathcal{T}}^2-1}$. Observe that $\|a\|_2^2 = 1 + \sum_{i=2}^d a_i^2$, where the a_i's are independent. For $i \neq 1$, each a_i is the sum of n independent $\pm\frac{1}{n}$ random variables (with $p = 1/2$) and so $\mathbb{E}[a_i^2] = \frac{1}{n}$. We can use Hoeffding's inequality to show that $\sum_{i=2}^d a_i^2$ does not deviate from its mean $\frac{d-1}{n}$ by more than $O(\sqrt{d \log d})$ with high probability. Since $n = O(\log d)$, this shows that for any constant $c_{\mathcal{T}} > 1$, we will have

$$\begin{aligned}\|a\|_2^2 \ &=\ 1 + \sum_{i=2}^d a_i^2 \\ &\ge\ \frac{c_{\mathcal{T}}^2}{c_{\mathcal{T}}^2 - 1}\ .\end{aligned}$$

with high probability for sufficiently large d.

Therefore, we have shown that (2.8) does *not* give an estimation algorithm with provable convergence to the correct result θ^*. In this part of the thesis, we develop several alternative approaches that do achieve convergence to the correct result while using approximate projections.

2.6 Results for the RIP setting

Several results in the following chapters are for constraint sets also known as *structured sparsity models* [36]. Structured sparsity models refine the widely-used notion of sparsity by adding additional constraints. Common examples include:

- collecting coordinates into groups (e.g., because certain features share a common structure),
- arranging coordinates into a hierarchy (e.g., because a child coordinate can only be active if the parent is as well), and
- graphs defined on the coordinates (e.g., to incorporate an interaction network defined on the features).

Incorporating such constraints leads to more interpretable models and a better sample complexity (both in theory and in practice). Since these advantages already emerge in the simpler RIP setting, we present some of the following chapters in this context. To be concrete, we now instantiate our general algorithm for structured sparsity models in the RIP setting.

Our formal problem of interest here is the following. Given the measurement vector

$$y = Ax + e\ ,$$

where x is a k-sparse vector and e is the "noise" vector, find a signal estimate $\widehat{x}$ such that:

$$\|x - \widehat{x}\|_2 \ \le\ c \cdot \|e\|_2\ . \tag{2.9}$$

Since the RIP setting is usually studied in the signal processing literature, we also adopt their notation here. In particular, we have the following correspondences:

- The design matrix X becomes the measurement matrix A.
- The unknown vector is now x instead of θ. The size of this vector is n (as opposed to d earlier). The sparsity is k (as opposed to s).
- The number of linear observations is m (it was n before).
- The noise e is typically assumed to be worst case and not stochastic.

It is known that there exist matrices A and associated recovery algorithms that produce a signal estimate $\widehat{x}$ satisfying Equation (2.9) with a constant approximation factor c and number of measurements $m = O(k \log(n/k))$. It is also known that this bound on the number of measurements m is asymptotically *optimal* for some constant c; see [88] and [100] (building upon the classical results of [103, 109, 139]). From a theoretial perspective, the goal of incorporating structured sparsity is to go beyond this bound (though we remark that in practice, there are further motivations such as interpretability).

The three algorithms we introduce in this section are the following.

Approximate model-iterative hard threholding (AM-IHT) In Section 2.6.2, we propose a new extension of the iterative hard thresholding (IHT) algorithm [50], which we call *approximate model iterative hard thresholding* (or AM-IHT). Informally, given head- and tail-approximation oracles and measurements $y = Ax + e$ with a matrix A satisfying the model-RIP, AM-IHT returns a signal estimate $\widehat{x}$ satisfying (2.9). We show that AM-IHT exhibits geometric convergence, and that the recovery guarantee for AM-IHT is asymptotically equivalent to the best available guarantees for model-based sparse recovery, despite using only approximate oracles.

Approximate model-CoSaMP (AM-CoSaMP) In Section 2.6.3, we propose a new extension of the compressive sampling matching pursuit algorithm (CoSaMP) [170], which we call *approximate model CoSaMP* (or AM-CoSAMP). As with AM-IHT, our proposed AM-CoSaMP algorithm requires a head-approximation oracle and a tail-approximation oracle. We show that AM-CoSaMP also exhibits geometric convergence, and that the recovery guarantee for AM-CoSaMP, as well as the RIP condition on A required for successful signal recovery, match the corresponding parameters for AM-IHT up to constant factors.

AM-IHT with sparse measurement matrices In Section 2.6.4, we show that an approximation-tolerant approach similar to AM-IHT succeeds even when the measurement matrix A is *itself* sparse. Our approach leverages the notion of the restricted isometry property in the ℓ_1-norm, also called the *RIP-1*, which was first introduced in [42] and developed further in the model-based context by [32, 101, 126]. For sparse A, we propose a modification of AM-IHT, which we call *AM-IHT with RIP-1*. Our proposed algorithm also exhibits geometric convergence under the *model RIP-1* assumption on the measurement matrix A.

2.6.1 Preliminaries

We write $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and $\mathcal{P}(A)$ to denote the power set of a set A. For a vector $x \in \mathbb{R}^n$ and a set $\Omega \subseteq [n]$, we write x_Ω for the restriction of x to Ω, i.e., $(x_\Omega)_i = x_i$ for $i \in \Omega$ and $(x_\Omega)_i = 0$ otherwise. Similarly, we write X_Ω for the submatrix of a matrix $X \in \mathbb{R}^{m \times n}$ containing the columns corresponding to Ω, i.e., a matrix in $\mathbb{R}^{m \times |\Omega|}$. Sometimes, we also restrict a matrix element-wise: for a set $\Omega \subseteq [m] \times [n]$, the matrix X_Ω is identical to X but the entries not contained in Ω are set to zero. The distinction between these two conventions will be clear from context.

A vector $x \in \mathbb{R}^n$ is said to be k-sparse if at most $k \leq n$ coordinates are nonzero. The support of x, $\mathrm{supp}(x) \subseteq [n]$, is the set of indices with nonzero entries in x. Hence $x_{\mathrm{supp}(x)} = x$. Observe that the set of *all* k-sparse signals is geometrically equivalent to the union of the $\binom{n}{k}$ canonical k-dimensional subspaces of $\mathbb{R}^n$. For a matrix $X \in \mathbb{R}^{h \times w}$, the support $\mathrm{supp}(X) \subseteq [h] \times [w]$ is also the set of indices corresponding to nonzero entries. For a matrix support set Ω, we denote the support of a column c in Ω with $\text{col-supp}(\Omega, c) = \{r \,|\, (r, c) \in \Omega\}$.

Often, some prior information about the support of a sparse signal x is available. A flexible way to model such prior information is to consider only the k-sparse signals with a permitted configuration of $\mathrm{supp}(x)$. This restriction motivates the notion of a *structured sparsity model*, which is geometrically equivalent to a subset of the $\binom{n}{k}$ canonical k-dimensional subspaces of $\mathbb{R}^n$.

Definition 2.11 (Structured sparsity model. From Definition 2 in [36])**.** *A structured sparsity model $\mathcal{M} \subseteq \mathbb{R}^n$ is the set of vectors $\mathcal{M} = \{x \in \mathbb{R}^n \,|\, \mathrm{supp}(x) \subseteq S \text{ for some } S \in \mathbb{M}\}$, where $\mathbb{M} = \{\Omega_1, \ldots, \Omega_l\}$ is the set of allowed structured supports with $\Omega_i \subseteq [n]$. We call $l = |\mathbb{M}|$ the size of the model $\mathcal{M}$.*

Note that the Ω_i in the definition above can have different cardinalities, but the largest cardinality will dictate the sample complexity in our bounds. Often it is convenient to work with the closure of $\mathbb{M}$ under taking subsets, which we denote with $\mathbb{M}^+ = \{\Omega \subseteq [n] \,|\, \Omega \subseteq S \text{ for some } S \in \mathbb{M}\}$. Then we can write the set of signals in the model as $\mathcal{M} = \{x \in \mathbb{R}^n \,|\, \mathrm{supp}(x) \in \mathbb{M}^+\}$.

In the analysis of our algorithms, we also use the notion of *model addition*: given two structured sparsity models $\mathcal{A}$ and $\mathcal{B}$, we define the sum $\mathcal{C} = \mathcal{A} + \mathcal{B}$ as $\mathcal{C} = \{a + b \,|\, a \in \mathcal{A} \text{ and } b \in \mathcal{B}\}$ (i.e., the Minkowski sum). Similarly, we define the corresponding set of allowed supports as $\mathbb{C} = \mathbb{A} + \mathbb{B} = \{\Omega \cup \Gamma \,|\, \Omega \in \mathbb{A} \text{ and } \Gamma \in \mathbb{B}\}$. We also use $t \times \mathbb{C}$ as a shorthand for t-times addition, i.e., $\mathbb{C} + \mathbb{C} + \ldots + \mathbb{C}$.

The framework of model-based compressive sensing [36] leverages the above notion of a structured sparsity model to design robust sparse recovery schemes. Specifically, the framework states that it is possible to recover a structured sparse signal $x \in \mathcal{M}$ from linear measurements $y = Ax + e$, provided that two conditions are satisfied: (i) the matrix A satisfies a variant of the restricted isometry property known as the *model-RIP*, and (ii) there exists an oracle that can efficiently *project* an arbitrary signal in $\mathbb{R}^n$ onto the model $\mathcal{M}$. We formalize these conditions as follows.

Definition 2.12 (Model-RIP. From Definition 3 in [36])**.** *The matrix $A \in \mathbb{R}^{m \times n}$ has the*

$(\delta, \mathbb{M})$*-model-RIP if the following inequalities hold for all* x *with* $\operatorname{supp}(x) \in \mathbb{M}^+$*:*

$$(1-\delta)\|x\|_2^2 \;\leq\; \|Ax\|_2^2 \leq (1+\delta)\|x\|_2^2 \,. \tag{2.10}$$

The following properties are direct consequences of the model-RIP and will prove useful in our proofs in Sections 2.6.2 and 2.6.3.

Fact 2.13 (adapted from Section 3 in [170])**.** *Let* $A \in \mathbb{R}^{m \times n}$ *be a matrix satisfying the* $(\delta, \mathbb{M})$*-model-RIP. Moreover, let* Ω *be a support in the model, i.e.,* $\Omega \in \mathbb{M}^+$*. Then the following properties hold for all* $x \in \mathbb{R}^n$ *and* $y \in \mathbb{R}^m$*:*

$$\begin{aligned} \|A_\Omega^T y\|_2 &\leq \sqrt{1+\delta}\|y\|_2 \,, \\ \|A_\Omega^T A_\Omega x\|_2 &\leq (1+\delta)\|x\|_2 \,, \\ \|(I - A_\Omega^T A_\Omega)x\|_2 &\leq \delta\|x\|_2 \,. \end{aligned}$$

Definition 2.14 (Model-projection oracle. From Section 3.2 in [36])**.** *A model-projection oracle is a function* $M : \mathbb{R}^n \to \mathcal{P}([n])$ *such that the following two properties hold for all* $x \in \mathbb{R}^n$*.*

Output model sparsity: $M(x) \in \mathbb{M}^+$*.*

Optimal model projection: Let $\Omega' = M(x)$*. Then,*

$$\|x - x_{\Omega'}\|_2 = \min_{\Omega \in \mathbb{M}} \|x - x_\Omega\|_2 \;.$$

Sometimes, we use a model-projection oracle M as a function from $\mathbb{R}^n$ to $\mathbb{R}^n$. This can be seen as a simple extension of Definition 2.14 where $M(x) = x_\Omega$, $\Omega = M'(x)$, and M' satisfies Definition 2.14.

Under these conditions, the authors of [36] show that compressive sampling matching pursuit (CoSaMP [170]) and iterative hard thresholding (IHT [50]) — two popular algorithms for sparse recovery — can be modified to achieve robust sparse recovery for the model $\mathcal{M}$. In particular, the modified version of IHT (called *Model-IHT* [36]) executes the following iterations until convergence:

$$x^{i+1} \leftarrow M(x^i + A^T(y - Ax^i)) \,, \tag{2.11}$$

where $x^1 = 0$ is the initial signal estimate. From a sampling complexity perspective, the benefit of this approach stems from the model-RIP assumption. Indeed, the following result indicates that with high probability, a large class of measurement matrices A satisfies the model-RIP with a *nearly optimal* number of rows:

Fact 2.15 ([36, 48])**.** *Let* $\mathbb{M}$ *be a structured sparsity model and let* k *be the size of the largest support in the model, i.e.,* $k = \max_{\Omega \in \mathbb{M}} |\Omega|$*. Let* $A \in \mathbb{R}^{m \times n}$ *be a matrix with i.i.d. sub-Gaussian entries. Then there is a constant* c *such that for* $0 < \delta < 1$*, any* $t > 0$*, and*

$$m \;\geq\; \frac{c}{\delta^2}\left(k \log \frac{1}{\delta} + \log|\mathbb{M}| + t\right) \,,$$

A *has the* $(\delta, \mathbb{M})$*-model-RIP with probability at least* $1 - e^{-t}$*.*

Since δ and t are typically constants, this bound can often be summarized as

$$m \;=\; O(k + \log|\mathbb{M}|) \;.$$

If the number of permissible supports (or equivalently, subspaces) $|\mathbb{M}|$ is asymptotically smaller than $\binom{n}{k}$, then m can be significantly smaller than the $O(k \log \frac{n}{k})$ measurement bound from "standard" compressive sensing. In particular, the situation where $|\mathbb{M}| = \text{poly}(n) \cdot 2^{O(k)}$ implies a measurement bound of $m = O(k)$ under the very mild assumption that $k = \Omega(\log n)$. Since $m = k$ measurements are necessary to reconstruct any k-sparse signal, this asymptotic behavior of m is information-theoretically optimal up to constant factors.

While model-based recovery approaches improve upon "standard" sparsity-based approaches in terms of sample-complexity, the computational cost of signal recovery crucially depends on the model-projection oracle M. Observe that Model-IHT (Equation 2.11) involves one invocation of the model-projection oracle M per iteration, and hence its overall running time scales with that of M. Therefore, model-based recovery approaches are relevant *only* in situations where efficient algorithms for finding the optimal model-projection are available.

2.6.2 Approximate Model-IHT

We now introduce our *approximation-tolerant model-based compressive sensing* framework. Essentially, we extend the model-based compressive sensing framework to work with approximate projection oracles, which we formalize in the definitions below. This extension enables model-based compressive sensing in cases where optimal model projections are beyond our reach, but approximate projections are still efficiently computable.

The core idea of our framework is to utilize *two* different notions of approximate projection oracles, defined as follows.

Definition 2.16 (Head approximation oracle)**.** *Let $\mathbb{M}, \mathbb{M}_H \subseteq \mathcal{P}([n])$, $p \geq 1$, and $c_H \in \mathbb{R}$ such that $c_H \leq 1$. Then $H : \mathbb{R}^n \to \mathcal{P}([n])$ is a $(c_H, \mathbb{M}, \mathbb{M}_H, p)$-head-approximation oracle if the following two properties hold for all $x \in \mathbb{R}^n$:*

Output model sparsity: $H(x) \in \mathbb{M}_H^+$.

Head approximation: Let $\Omega' = H(x)$. Then

$$\|x_{\Omega'}\|_p \geq c_H \|x_\Omega\|_p \textit{ for all } \Omega \in \mathbb{M} \;.$$

Definition 2.17 (Tail approximation oracle)**.** *Let $\mathbb{M}, \mathbb{M}_T \subseteq \mathcal{P}([n])$, $p \geq 1$ and $c_T \in \mathbb{R}$ such that $c_T \geq 1$. Then $T : \mathbb{R}^n \to \mathcal{P}([n])$ is a $(c_T, \mathbb{M}, \mathbb{M}_T, p)$-tail-approximation oracle if the following two properties hold for all $x \in \mathbb{R}^n$:*

Output model sparsity: $T(x) \in \mathbb{M}_T^+$.
Tail approximation: Let $\Omega' = T(x)$. Then

$$\|x - x_{\Omega'}\|_p \leq c_T \|x - x_\Omega\|_p \textit{ for all } \Omega \in \mathbb{M} \;.$$

We trivially observe that a head approximation oracle with approximation factor $c_H = 1$ is equivalent to a tail approximation oracle with factor $c_T = 1$, and vice versa. Further, we

Algorithm 3 Approximate Model-IHT

```
function AM-IHT($y, A, t$)
    $x^0 \leftarrow 0$
    for $i \leftarrow 0, \ldots, t$ do
        $b^i \leftarrow A^T(y - Ax^i)$
        $x^{i+1} \leftarrow T(x^i + H(b^i))$
    end for
    return $x^{t+1}$
end function
```

observe that for any model $\mathcal{M}$, if $x \in \mathcal{M}$ then $\|x - x_\Omega\|_2 = 0$ for some $\Omega \in \mathbb{M}$. Hence, *any* tail approximation oracle must be exact in the sense that the returned support Ω' has to satisfy $\|x - x_{\Omega'}\|_2 = 0$, or equivalently, $\operatorname{supp}(x) \subseteq T(x)$. On the other hand, we note that $H(x)$ does not need to return an optimal support if the input signal x is in the model $\mathcal{M}$.

An important feature of the above definitions of approximation oracles is that they permit projections into *larger* models. In other words, the oracle can potentially return a signal that belongs to a larger model $\mathbb{M}' \supseteq \mathbb{M}$. Our algorithms in the following chapters will use this feature repeatedly.

Equipped with these notions of approximate projection oracles, we introduce a new algorithm for model-based compressive sensing. We call our algorithm *Approximate Model-IHT* (AM-IHT); see Algorithm 3 for a full description. Notice that every iteration of AM-IHT uses *both* a head-approximation oracle H and a tail-approximation oracle T. This is in contrast to the Model-IHT algorithm discussed above in Section 2.5, which solely made use of a tail approximation oracle T'.

Our main result of this section (Theorem 2.20) states the following: if the measurement matrix A satisfies the model-RIP for $\mathbb{M} + \mathbb{M}_T + \mathbb{M}_H$ and approximate projection oracles H and T are available, then AM-IHT exhibits provably robust recovery. We make the following assumptions in the analysis of AM-IHT: (i) $x \in \mathbb{R}^n$ and $x \in \mathcal{M}$. (ii) $y = Ax + e$ for an arbitrary $e \in \mathbb{R}^m$ (the measurement noise). (iii) T is a $(c_T, \mathbb{M}, \mathbb{M}_T, 2)$-tail-approximation oracle. (iv) H is a $(c_H, \mathbb{M}_T + \mathbb{M}, \mathbb{M}_H, 2)$-head-approximation-oracle. (v) A has the $(\delta, \mathbb{M} + \mathbb{M}_T + \mathbb{M}_H)$-model-RIP.

As in IHT, we use the *residual proxy* $b^i = A^T(y - Ax^i)$ as the update in each iteration (see Algorithm 3). The key idea of our proof is the following: when applied to the residual proxy b^i, the head-approximation oracle H returns a support Γ that contains "most" of the relevant mass contained in r^i. Before we formalize this statement in Lemma 2.19, we first establish the RIP of A on all relevant vectors.

Lemma 2.18. *Let $r^i = x - x^i$, $\Omega = \operatorname{supp}(r^i)$, and $\Gamma = \operatorname{supp}(H(b^i))$. For all $x' \in \mathbb{R}^n$ with $\operatorname{supp}(x') \subseteq \Omega \cup \Gamma$ we have*

$$(1 - \delta)\|x'\|_2^2 \leq \|Ax'\|_2^2 \leq (1 + \delta)\|x'\|_2^2 \,.$$

Proof. By the definition of T, we have $\operatorname{supp}(x^i) \in \mathbb{M}_T$. Since $\operatorname{supp}(x) \in \mathbb{M}$, we have $\operatorname{supp}(x - x^i) \in \mathbb{M}_T + \mathbb{M}$ and hence $\Omega \in \mathbb{M}_T + \mathbb{M}$. Moreover, $\operatorname{supp}(H(b^i)) \in \mathbb{M}_H$ by the definition of H. Therefore $\Omega \cup \Gamma \in \mathbb{M} + \mathbb{M}_T + \mathbb{M}_H$, which allows us to use the model-RIP of A on x' with $\mathrm{supp}(x') \subseteq \Omega \cup \Gamma$. □

We now establish our main lemma, which will also prove useful in Section 2.6.3. A similar result (with a different derivation approach and different constants) appears in Section 4 of the conference version of this manuscript [119].

Lemma 2.19. *Let $r^i = x - x^i$ and $\Gamma = \mathrm{supp}(H(b^i))$. Then,*

$$\|r^i_{\Gamma^c}\|_2 \;\le\; \sqrt{1-\alpha_0^2}\|r^i\|_2 + \left[\frac{\beta_0}{\alpha_0} + \frac{\alpha_0\beta_0}{\sqrt{1-\alpha_0^2}}\right]\|e\|_2 \,. \tag{2.12}$$

where

$$\begin{aligned} \alpha_0 &= c_H(1-\delta) - \delta \qquad \textit{and} \\ \beta_0 &= (1+c_H)\sqrt{1+\delta}\,. \end{aligned}$$

We assume that c_H and δ are such that α_0.

Proof. We provide lower and upper bounds on $\|H(b^i)\|_2 = \|b^i_\Gamma\|_2$, where $b^i = A^T(y - Ax_i) = A^TAr^i + A^Te$. Let $\Omega = \mathrm{supp}(r^i)$. From the head-approximation property, we can bound $\|b^i_\Gamma\|_2$ as:

$$\begin{aligned} \|b^i_\Gamma\|_2 &= \|A^T_\Gamma A r^i + A^T_\Gamma e\|_2 \\ &\ge c_H\|A^T_\Omega A r^i + A^T_\Omega e\|_2 \\ &\ge c_H\|A^T_\Omega A_\Omega r^i\|_2 - c_H\|A^T_\Omega e\|_2 \\ &\ge c_H(1-\delta)\|r^i\|_2 - c_H\sqrt{1+\delta}\|e\|_2\,, \end{aligned}$$

where the inequalities follow from Fact 2.13 and the triangle inequality. This provides the lower bound on $\|b^i_\Gamma\|_2$.

Now, consider r_Γ. By repeated use of the triangle inequality, we get

$$\begin{aligned} \|b^i_\Gamma\|_2 &= \|A^T_\Gamma A r^i + A^T_\Gamma e\|_2 \\ &= \|A^T_\Gamma A r^i - r^i_\Gamma + r^i_\Gamma + A^T_\Gamma e\|_2 \\ &\le \|A^T_\Gamma A r^i - r^i_\Gamma\|_2 + \|r^i_\Gamma\|_2 + \|A^T_\Gamma e\|_2 \\ &\le \|A^T_{\Gamma\cup\Omega} A r^i - r^i_{\Gamma\cup\Omega}\|_2 + \|r^i_\Gamma\|_2 + \sqrt{1+\delta}\|e\|_2 \\ &\le \delta\|r^i\|_2 + \|r^i_\Gamma\|_2 + \sqrt{1+\delta}\|e\|_2\,, \end{aligned}$$

where the last inequality again follows from Fact 2.13. This provides the upper bound on $\|b^i_\Gamma\|_2$.

Combining the two bounds and grouping terms, we obtain the following inequality. In order to simplify notation, we write $\alpha_0 = c_H(1-\delta) - \delta$ and $\beta_0 = (1+c_H)\sqrt{1+\delta}$.

$$\|r^i_\Gamma\|_2 \ge \alpha_0\|r^i\|_2 - \beta_0\|e\|_2\,. \tag{2.13}$$

Next, we examine the right hand side of (2.13) more carefully. Let us assume that the RIP constant δ is set to be small enough such that it satisfies $c_H > \delta/(1-\delta)$. There are two mutually exclusive cases:

Case 1: The value of $\left\|r^i\right\|_2$ satisfies $\alpha_0\left\|r^i\right\|_2 \leq \beta_0\|e\|_2$. Then, consider the vector $r^i_{\Gamma^c}$, i.e., the vector r^i restricted to the set of coordinates in the complement of Γ. Clearly, its norm is smaller than $\|r^i\|_2$. Therefore, we have

$$\|r^i_{\Gamma^c}\|_2 \leq \frac{\beta_0}{\alpha_0}\|e\|_2 \,. \tag{2.14}$$

Case 2: The value of $\left\|r^i\right\|_2$ satisfies $\alpha_0\left\|r^i\right\|_2 \geq \beta_0\|e\|_2$. Rewriting (2.13), we get

$$\|r^i_{\Gamma}\|_2 \geq \|r^i\|_2\left(\alpha_0 - \frac{\beta_0\|e\|_2}{\|r_i\|_2}\right) \,.$$

Moreover, we also have $\|r^i\|_2^2 = \|r^i_{\Gamma}\|_2^2 + \|r^i_{\Gamma^c}\|_2^2$. Therefore, we obtain

$$\|r^i_{\Gamma^c}\|_2 \leq \|r^i\|_2\sqrt{1-\left(\alpha_0 - \beta_0\frac{\|e\|_2}{\|r^i\|_2}\right)^2} \,. \tag{2.15}$$

We can simplify the right hand side using the following geometric argument, adapted from [149]. Denote $\omega_0 = \alpha_0 - \beta_0\|e\|_2/\|r^i\|_2$. Then, $0 \leq \omega_0 < 1$ because $\alpha_0\left\|r^i\right\|_2 \geq \beta_0\|e\|_2$, $\alpha_0 < 1$, and $\beta_0 \geq 1$. The function $g(\omega_0) = \sqrt{1-\omega_0^2}$ traces an arc of the unit circle as a function of ω_0 and therefore is upper-bounded by the y-coordinate of *any* tangent line to the circle evaluated at ω_0. For a free parameter $0 < \omega < 1$ (the tangent point of the tangent line), a straightforward calculation yields that

$$\sqrt{1-\omega_0^2} \leq \frac{1}{\sqrt{1-\omega^2}} - \frac{\omega}{\sqrt{1-\omega^2}}\omega_0 \,.$$

Therefore, substituting into the bound for $\|r^i_{\Gamma^c}\|_2$, we get:

$$\begin{aligned}\|r^i_{\Gamma^c}\|_2 &\leq \|r^i\|_2\left(\frac{1}{\sqrt{1-\omega^2}} - \frac{\omega}{\sqrt{1-\omega^2}}\left(\alpha_0 - \beta_0\frac{\|e\|_2}{\|r^i\|_2}\right)\right)\\ &= \frac{1-\omega\alpha_0}{\sqrt{1-\omega^2}}\|r^i\|_2 + \frac{\omega\beta_0}{\sqrt{1-\omega^2}}\|e\|_2 \,.\end{aligned}$$

The coefficient preceding $\|r^i\|_2$ determines the overall convergence rate, and the minimum value of the coefficient is attained by setting $\omega = \alpha_0$. Substituting, we obtain

$$\|r^i_{\Gamma^c}\|_2 \leq \sqrt{1-\alpha_0^2}\|r^i\|_2 + \frac{\alpha_0\beta_0}{\sqrt{1-\alpha_0^2}}\|e\|_2 \,. \tag{2.16}$$

Combining the mutually exclusive cases (2.14) and (2.16), we obtain

$$\|r^i_{\Gamma^c}\|_2 \leq \sqrt{1-\alpha_0^2}\|r^i\|_2 + \left[\frac{\beta_0}{\alpha_0} + \frac{\alpha_0\beta_0}{\sqrt{1-\alpha_0^2}}\right]\|e\|_2 \,,$$

which proves the lemma. □

Theorem 2.20 (Geometric convergence of AM-IHT)**.** *Let $r^i = x - x^i$, where x^i is the signal estimate computed by AM-IHT in iteration i. Then,*

$$\|r^{i+1}\|_2 \leq \alpha \|r^i\|_2 + \beta \|e\|_2 \,,$$

where

$$\begin{aligned}
\alpha &= (1 + c_T)\left[\delta + \sqrt{1 - \alpha_0^2}\right] \,, \\
\beta &= (1 + c_T)\left[\frac{\beta_0}{\alpha_0} + \frac{\alpha_0 \beta_0}{\sqrt{1 - \alpha_0^2}} + \sqrt{1 + \delta}\right] \,, \\
\alpha_0 &= c_H(1 - \delta) - \delta \,, \\
\beta_0 &= (1 + c_H)\sqrt{1 + \delta} \,.
\end{aligned}$$

We assume that c_H and δ are such that $\alpha_0 > 0$.

Proof. Let $a = x^i + H(b^i)$. From the triangle inequality, we have:

$$\begin{aligned}
\|x - x^{i+1}\|_2 &= \|x - T(a)\|_2 \\
&\leq \|x - a\|_2 + \|a - T(a)\|_2 \\
&\leq (1 + c_T)\|x - a\|_2 \\
&= (1 + c_T)\|x - x^i - H(b^i)\|_2 \\
&= (1 + c_T)\|r^i - H(A^T A r^i + A^T e)\|_2 \,. \qquad (2.17)
\end{aligned}$$

We can further bound $\|r^i - H(A^T A r^i + A^T e)\|_2$ in terms of $\|r^i\|_2$. Let $\Omega = \operatorname{supp}(r^i)$ and $\Gamma = \operatorname{supp}(H(A^T A r^i + A^T e))$. We have the inequalities

$$\begin{aligned}
&\|r^i - H(A^T A r^i + A^T e)\|_2 \\
&\quad = \|r^i_\Gamma + r^i_{\Gamma^c} - A^T_\Gamma A r^i + A^T_\Gamma e\|_2 \\
&\quad \leq \|A^T_\Gamma A r^i - r^i_\Gamma\|_2 + \|r^i_{\Gamma^c}\|_2 + \|A^T_\Gamma e\|_2 \\
&\quad \leq \|A^T_{\Gamma \cup \Omega} A r^i - r^i_{\Gamma \cup \Omega}\|_2 + \|r^i_{\Gamma^c}\|_2 + \|A^T_\Gamma e\|_2 \\
&\quad \leq \delta \|r^i\|_2 + \sqrt{1 - \alpha_0^2}\|r^i\|_2 \\
&\quad + \left[\frac{\beta_0}{\alpha_0} + \frac{\alpha_0 \beta_0}{\sqrt{1 - \alpha_0^2}} + \sqrt{1 + \delta}\right] \|e\|_2 \,,
\end{aligned}$$

where the last inequality follows from the RIP and (2.12). Putting this together with (2.17) and grouping terms, we get

$$\|x - x^{i+1}\|_2 \leq \alpha \|x - x^i\|_2 + \beta \|e\|_2 \,, \qquad (2.18)$$

thus proving the Theorem. □

In the noiseless case, we can ignore the second term and only focus on the leading recurrence factor:

$$\alpha = (1 + c_T)\left(\delta + \sqrt{1 - (c_H(1-\delta) - \delta)^2}\right) .$$

For convergence, we need α to be strictly smaller than 1. Note that we can make δ as small as we desire since this assumption only affects the measurement bound by a constant factor. Therefore, the following condition must hold for guaranteed convergence:

$$\begin{aligned} &(1 + c_T)\sqrt{1 - c_H^2} < 1 \, , \qquad \text{or equivalently,} \\ &c_H^2 > 1 - \frac{1}{(1 + c_T)^2} \, . \end{aligned} \tag{2.19}$$

Under this condition, AM-IHT exhibits geometric convergence comparable to the existing model-based compressive sensing results of [36]. AM-IHT achieves this *despite using only approximate projection oracles.* In Section 2.6.5, we relax condition (2.19) so that geometric convergence is possible for *any* constants c_T and c_H.

The geometric convergence of AM-IHT implies that the algorithm quickly recovers a good signal estimate. Formally, we obtain:

Corollary 2.21. *Let T and H be approximate projection oracles with c_T and c_H such that $0 < \alpha < 1$. Then after*

$$t = \left\lceil \frac{\log \|x\|_2/\|e\|_2}{\log \frac{1}{\alpha}} \right\rceil$$

iterations, AM-IHT returns a signal estimate $\widehat{x}$ satisfying

$$\|x - \widehat{x}\|_2 \le \left(1 + \frac{\beta}{1-\alpha}\right)\|e\|_2 \, .$$

Proof. As before, let $r^i = x - x^i$. Using $\left\|r^0\right\|_2 = \|x\|_2$, Theorem 2.20, and a simple inductive argument shows that

$$\left\|r^{i+1}\right\|_2 \le \alpha^i \|x\|_2 + \beta\|e\|_2 \sum_{j=0}^{i} \alpha^j \, .$$

For

$$i \;=\; \left\lceil \frac{\log \|x\|_2/\|e\|_2}{\log \frac{1}{\alpha}} \right\rceil ,$$

we get $\alpha^i\|x\|_2 \le \|e\|_2$. Moreover, we can bound the geometric series $\sum_{j=0}^{t} \alpha^j$ by $\frac{1}{1-\alpha}$. Combining these bounds gives the guarantee stated in the theorem. □

2.6.3 Approximate Model-CoSaMP

In this section, we propose a second algorithm for model-based compressive sensing with approximate projection oracles. Our algorithm is a generalization of model-based CoSaMP, which was initially developed in [36]. We call our variant *Approximate Model-CoSaMP* (or AM-CoSaMP); see Algorithm 4 for a complete description.

Algorithm 4 closely resembles the *Signal-Space CoSaMP* (or SSCoSaMP) algorithm proposed and analyzed in [82, 108]. Like our approach, SSCoSaMP also makes assumptions about

Algorithm 4 Approximate Model-CoSaMP

```
function AM-CoSaMP($y, A, t$)
    $x^0 \leftarrow 0$
    for $i \leftarrow 0, \ldots, t$ do
        $b^i \leftarrow A^T(y - Ax^i)$
        $\Gamma \leftarrow \mathrm{supp}(H(b^i))$
        $S \leftarrow \Gamma \cup \mathrm{supp}(x^i)$
        $z|_S \leftarrow A_S^{\dagger} y, \quad z|_{S^C} \leftarrow 0$
        $x^{i+1} \leftarrow T(z)$
    end for
    return $x^{t+1}$
end function
```

the existence of head- and tail-approximation oracles. However, there are some important technical differences in our development. SSCoSaMP was introduced in the context of recovering signals that are sparse in overcomplete and incoherent dictionaries. In contrast, we focus on recovering signals from structured sparsity models.

Moreover, the authors of [82, 108] assume that a *single* oracle simultaneously achieves the conditions specified in Definitions 2.16 and 2.17. In contrast, our approach assumes the existence of two separate head- and tail-approximation oracles and consequently is somewhat more general. Finally, our analysis is simpler and more concise than that provided in [82, 108] and follows directly from the results in Section 2.6.2.

We prove that AM-CoSaMP (Alg. 4) exhibits robust signal recovery. We make the same assumptions as in Section 2.6.2: (i) $x \in \mathbb{R}^n$ and $x \in \mathcal{M}$. (ii) $y = Ax + e$ for an arbitrary $e \in \mathbb{R}^m$ (the measurement noise). (iii) T is a $(c_T, \mathbb{M}, \mathbb{M}_T, 2)$-tail-approximation oracle. (iv) H is a $(c_H, \mathbb{M}_T + \mathbb{M}, \mathbb{M}_H, 2)$-head-approximation-oracle. (v) A has the $(\delta, \mathbb{M} + \mathbb{M}_T + \mathbb{M}_H)$-model-RIP. Our main result in this section is the following:

Theorem 2.22 (Geometric convergence of AM-CoSaMP)**.** *Let $r^i = x - x^i$, where x^i is the signal estimate computed by AM-CoSaMP in iteration i. Then,*

$$\|r^{i+1}\|_2 \leq \alpha \|r^i\|_2 + \beta \|e\|_2 \,,$$

where

$$\begin{aligned}
\alpha &= (1 + c_T)\sqrt{\frac{1+\delta}{1-\delta}}\sqrt{1 - \alpha_0^2}\,, \\
\beta &= (1 + c_T)\left[\sqrt{\frac{1+\delta}{1-\delta}}\left(\frac{\beta_0}{\alpha_0} + \frac{\alpha_0 \beta_0}{\sqrt{1-\alpha_0^2}}\right) + \frac{2}{\sqrt{1-\delta}}\right]\,, \\
\alpha_0 &= c_H(1-\delta) - \delta\,, \\
\beta_0 &= (1 + c_H)\sqrt{1+\delta}\,.
\end{aligned}$$

Proof. We can bound the error $\|r^{i+1}\|_2$ as follows:

$$\begin{aligned}\|r^{i+1}\|_2 &= \|x - x^{i+1}\|_2 \\ &\leq \|x^{i+1} - z\|_2 + \|x - z\|_2 \\ &\leq c_T\|x - z\|_2 + \|x - z\|_2 \\ &= (1 + c_T)\|x - z\|_2 \\ &\leq (1 + c_T)\frac{\|A(x - z)\|_2}{\sqrt{1 - \delta}} \\ &= (1 + c_T)\frac{\|Ax - Az\|_2}{\sqrt{1 - \delta}} \,.\end{aligned}$$

Most of these inequalities follow the same steps as the proof provided in [170]. The second relation above follows from the triangle inequality, the third relation follows from the tail approximation property and the fifth relation follows from the $(\delta, \mathbb{M}+\mathbb{M}_T+\mathbb{M}_H)$-model-RIP of A.

We also have $Ax = y - e$ and $Az = A_S z_S$. Substituting, we get:

$$\begin{aligned}\|r^{i+1}\|_2 &\leq (1 + c_T)\left(\frac{\|y - A_S z_S\|_2}{\sqrt{1 - \delta}} + \frac{\|e\|_2}{\sqrt{1 - \delta}}\right) \\ &\leq (1 + c_T)\left(\frac{\|y - A_S x_S\|_2}{\sqrt{1 - \delta}} + \frac{\|e\|_2}{\sqrt{1 - \delta}}\right) \,. \qquad (2.20)\end{aligned}$$

The first inequality follows from the triangle inequality and the second from the fact that z_S is the least squares estimate $A_S^\dagger y$ (in particular, it is at least as good as x_S).

Now, observe that $y = Ax + e = A_S x_S + A_{S^c} x_{S^c} + e$. Therefore, we can further simplify inequality (2.20) as

$$\begin{aligned}&\|r^{i+1}\|_2 \\ &\leq (1 + c_T)\frac{\|A_{S^c} x_{S^c}\|_2}{\sqrt{1 - \delta}} + (1 + c_T)\frac{2\|e\|_2}{\sqrt{1 - \delta}} \\ &\leq (1 + c_T)\frac{\sqrt{1 + \delta}}{\sqrt{1 - \delta}}\|x_{S^c}\|_2 + (1 + c_T)\frac{2\|e\|_2}{\sqrt{1 - \delta}} \\ &= (1 + c_T)\sqrt{\frac{1 + \delta}{1 - \delta}}\|(x - x^i)_{S^c}\|_2 + (1 + c_T)\frac{2\|e\|_2}{\sqrt{1 - \delta}} \\ &\leq (1 + c_T)\sqrt{\frac{1 + \delta}{1 - \delta}}\|r^i_{\Gamma^c}\|_2 + (1 + c_T)\frac{2\|e\|_2}{\sqrt{1 - \delta}} \,. \qquad (2.21)\end{aligned}$$

The first relation once again follows from the triangle inequality. The second relation follows from the fact that $\text{supp}(x_{S^c}) \in \mathbb{M}^+$ (since $\text{supp}(x) \in \mathbb{M}^+$), and therefore, $A_{S^c} x_{S^c}$ can be upper-bounded using the model-RIP. The third follows from the fact that x_i supported on S^c is zero because S fully subsumes the support of x^i. The final relation follows from the fact that $S^c \subseteq \Gamma^c$ (see line 6 in the algorithm).

Note that the support Γ is defined as in Lemma 2.18. Therefore, we can use (2.12) and bound $\|r^i_{\Gamma^c}\|_2$ in terms of $\|r^i\|_2$, c_H, and δ. Substituting into (2.21) and rearranging terms, we obtain the stated theorem. □

As in the analysis of AM-IHT, suppose that $e = 0$ and δ is very small. Then, we achieve geometric convergence, i.e., $\alpha < 1$, if the approximation factors c_T and c_H satisfy

$$\begin{aligned} &(1 + c_T)\sqrt{1 - c_H^2} < 1\,, \qquad \text{or equivalently,} \\ &c_H^2 > 1 - \frac{1}{(1 + c_T)^2}\,. \end{aligned}$$

Therefore, the conditions for convergence of AM-IHT and AM-CoSaMP are identical in this regime. As for AM-IHT, we relax this condition for AM-CoSaMP in Section 2.6.5 and show that geometric convergence is possible for *any* constants c_T and c_H.

2.6.4 Approximate Model-IHT with RIP-1 matrices

AM-IHT and AM-CoSaMP (Algorithms 3 and 4) rely on measurement matrices satisfying the model-RIP (Definition 2.12). It is known that $m \times n$ matrices whose elements are drawn i.i.d. from a sub-Gaussian distribution satisfy this property with high probability while requiring only a small number of rows m [35, 36]. However, such matrices are *dense* and consequently incur significant costs of $\Theta(m \cdot n)$ for both storage and matrix-vector multiplications.

One way to circumvent this issue is to consider *sparse* measurement matrices [104]. Sparse matrices can be stored very efficiently and enable fast matrix-vector multiplication (with both costs scaling proportionally to the number of nonzeros). However, the usual RIP does not apply for such matrices. Instead, such matrices are known to satisfy the RIP in the ℓ_1-norm (or *RIP-1*). Interestingly, it can be shown that this property is sufficient to enable robust sparse recovery for arbitrary signals [42]. Moreover, several existing algorithms for sparse recovery can be modified to work with sparse measurement matrices; see [42, 101].

In the model-based compressive sensing context, one can analogously define the RIP-1 over structured sparsity models as follows:

Definition 2.23 (Model RIP-1)**.** *A matrix $A \in \mathbb{R}^{m\times n}$ has the $(\delta, \mathbb{M})$-model RIP-1 if the following holds for all x with* $\mathrm{supp}(x) \in \mathbb{M}^+$*:*

$$(1 - \delta)\|x\|_1 \leq \|Ax\|_1 \leq (1 + \delta)\|x\|_1\,. \tag{2.22}$$

Indyk and Razenshteyn [126] establish both lower and upper bounds on the number of measurements required to satisfy the model RIP-1 for certain structured sparsity models. Similar to Fact 2.15, they also provides a general sampling bound based on the cardinality of the model:

Fact 2.24 (Theorem 9 in [126])**.** *Let $\mathcal{M}$ be a structured sparsity model and let k be the size of the largest support in the model, i.e., $k = \max_{\Omega\in\mathbb{M}}|\Omega|$. Then there is a $m \times n$ matrix satisfying the $(\delta, \mathbb{M})$-model RIP-1 with*

$$m = O(\frac{k}{\delta^2} \cdot \frac{\log(n/l)}{\log(k/l)})\,,$$

where $l = \frac{\log|\mathbb{M}|}{\log(n/k)}$.

Algorithm 5 AM-IHT with RIP-1

```
function AM-IHT-RIP-1($y, A, t$)
    $x^0 \leftarrow 0$
    for $i \leftarrow 0, \ldots, t$ do
        $x^{i+1} \leftarrow T(x^i + H(\mathrm{MED}(y - Ax^i)))$
    end for
    return $x^{t+1}$
end function
```

Subsequently, Bah, Baldaserre, and Cevher [32] propose a modification of *expander iterative hard thresholding* (EIHT) [101] that achieves stable recovery for arbitrary structured sparsity models. As before, this modified algorithm only works when provided access to *exact* model-projection oracles. Below, we propose a more general algorithm suitable for model-based recovery using only approximate projection oracles.

Before proceeding further, it is worthwhile to understand a particular class of matrices that satisfy the RIP-1. It is known that adjacency matrices of certain carefully chosen random bipartite graphs, known as *bipartite expanders*, satisfy the model RIP-1 [42, 126]. Indeed, suppose that such a matrix A represents the bipartite graph $G = ([n], [m], E)$, where E is the set of edges. For any $S \subseteq [n]$, define $\Gamma(S)$ to be the set of nodes in $[m]$ connected to S by an edge in E. Therefore, we can define the *median operator* $\mathrm{MED}(u) : \mathbb{R}^m \to \mathbb{R}^n$ for any $u \in \mathbb{R}^m$ component-wise as follows:

$$[\mathrm{MED}(u)]_i = \mathrm{median}[u_j : j \in \Gamma(\{i\})] \, .$$

This operator is crucial in our algorithm and proofs below.

We now propose a variant of AM-IHT (Algorithm 3) that is suitable when the measurement matrix A satisfies the RIP-1. The description of this new version is provided as Algorithm 5. Compared to AM-IHT, the important modification in the RIP-1 algorithm is the use of the median operator $\mathrm{MED}(\cdot)$ instead of the transpose of the measurement matrix A.

We analytically characterize the convergence behavior of Algorithm 5. First, we present the following Lemma, which is proved in [32] based on [101].

Lemma 2.25 (Lemma 7.2 in [32])**.** *Suppose that A satisfies the $(\delta, \mathbb{M})$-model-RIP-1. Then, for any vectors $x \in \mathbb{R}^n$, $e \in \mathbb{R}^m$, and any support $S \in \mathbb{M}^+$,*

$$\|[x - \mathrm{MED}(Ax_S + e)]_S\|_1 \leq \rho_0 \|x_S\|_1 + \tau_0 \|e\|_1 \, .$$

Here, $\rho_0 = 4\delta/(1 - 4\delta)$ and τ_0 is a positive scalar that depends on δ.

Armed with this Lemma, we now prove the main result of this section. We make similar assumptions as in Section 2.6.2, this time using the model-RIP-1 and approximate projection oracles for the ℓ_1-norm: (i) $x \in \mathbb{R}^n$ and $x \in \mathcal{M}$. (ii) $y = Ax + e$ for an arbitrary $e \in \mathbb{R}^m$ (the measurement noise). (iii) T is a $(c_T, \mathbb{M}, \mathbb{M}_T, 1)$-tail-approximation oracle. (iv) H is a $(c_H, \mathbb{M}_T + \mathbb{M}, \mathbb{M}_H, 1)$-head-approximation-oracle. (v) A has the $(\delta, \mathbb{M} + \mathbb{M}_T + \mathbb{M}_H)$-model-RIP-1. Then, we obtain:

Theorem 2.26 (Geometric convergence of AM-IHT with RIP-1)**.** *Let $r^i = x - x^i$, where*

x^i is the signal estimate computed by AM-IHT-RIP-1 in iteration i. Let ρ_0, τ_0 be as defined in Lemma 2.25. Then, AM-IHT-RIP-1 exhibits the following convergence property:

$$\|r^{i+1}\|_1 \leq \rho \|r^i\|_1 + \tau \|e\|_1 \,,$$

where

$$\begin{aligned} \rho &= (1 + c_T)(2\rho_0 + 1 - c_H(1 - \rho_0)) \,, \\ \tau &= (1 + c_T)(2 + c_H)\tau_0 \,. \end{aligned}$$

Proof. Let $a_i = x_i + H(\mathrm{MED}(y - Ax_i))$. The triangle inequality gives:

$$\begin{aligned} \|r^{i+1}\|_1 &= \|x - x^{i+1}\|_1 \\ &\leq \|x - a^i\|_1 + \|x^{i+1} - a^i\|_1 \\ &\leq (1 + c_T)\|x - a^i\|_1 \\ &\leq (1 + c_T)\|x - x^i - H(\mathrm{MED}(y - Ax^i))\|_1 \\ &= (1 + c_T)\|r^i - H(\mathrm{MED}(Ar^i + e))\|_1 \,. \end{aligned}$$

Let $v = \mathrm{MED}(Ar^i + e)$, $\Omega = \mathrm{supp}(r^i)$, and Γ be the support returned by the head oracle H. We have:

$$\|H(v)\|_1 = \|v_\Gamma\|_1 \geq c_H \|v_\Omega\|_1 \,, \tag{2.23}$$

due to the head-approximation property of H.

On the other hand, we also have

$$\begin{aligned} \|v_\Omega - r^i\|_1 &= \|(\mathrm{MED}(Ar^i + e) - r^i)_\Omega\|_1 \\ &\leq \|(\mathrm{MED}(Ar^i + e) - r^i)_{\Omega \cup \Gamma}\|_1 \\ &\leq \rho_0 \|r^i\|_1 + \tau_0 \|e\|_1 \,. \end{aligned}$$

where the last inequality follows from Lemma 2.25 (note that we use the lemma for the model $\mathbb{M} + \mathbb{M}_T + \mathbb{M}_H$). Further, by applying the triangle inequality again and combining with (2.23), we get

$$\|H(v)\|_1 \geq c_H(1 - \rho_0)\|r^i\|_1 - c_H \tau_0 \|e\|_1 \,. \tag{2.24}$$

We also have the following series of inequalities:

$$\begin{aligned} \|H(v)\|_1 &= \|H(v) - r^i_\Gamma + r^i_\Gamma\|_1 \\ &\leq \|v_\Gamma - r^i_\Gamma\|_1 + \|r^i_\Gamma\|_1 \\ &\leq \|v_{\Gamma \cup \Omega} - r^i_{\Gamma \cup \Omega}\|_1 + \|r^i_\Gamma\|_1 \\ &= \|(\mathrm{MED}(Ar^i + e) - r^i)_{\Omega \cup \Gamma}\|_1 + \|r^i_\Gamma\|_1 \\ &\leq \rho_0 \|r^i\|_1 + \tau_0 \|e\|_1 + \|r^i_\Gamma\|_1 \,. \end{aligned}$$

Here, we have once again invoked Lemma 2.25. Moreover, $\|r^i_\Gamma\|_1 = \|r^i\|_1 - \|r^i_{\Gamma^c}\|_1$. Combining with (2.24) and rearranging terms, we get:

$$\|r^i_{\Gamma^c}\|_1 \leq (\rho_0 + 1 - c_H(1 - \rho_0))\|r^i\|_1 + (1 + c_H)\tau_0 \|e\|_1 \,. \tag{2.25}$$

Recall that

$$\begin{aligned}\|r^{i+1}\|_1 &\leq (1+c_T)\|r^i - H(v)\|_1\\ &= (1+c_T)\left(\|r^i_\Gamma - v_\Gamma\|_1 + \|r^i_{\Gamma^c}\|_1\right),\end{aligned}$$

since $v_\Gamma = H(v) = H(\mathrm{MED}(Ar^i + e))$. Invoking Lemma 2.25 one last time and combining with (2.25), we obtain

$$\begin{aligned}\|r^{i+1}\|_1 \leq\ & (1+c_T)\Big[\rho_0\|r^i\|_1 + \tau_0\|e\|_1\\ &+ (\rho_0 + 1 - c_H(1-\rho_0))\|r^i\|_1 + (1+c_H)\tau_0\|e\|_1\Big]\\ \leq\ & (1+c_T)(2\rho_0 + 1 - c_H(1-\rho_0))\|r^i\|_1\\ &+ (1+c_T)(2+c_H)\tau_0\|e\|_1\,,\end{aligned}$$

as claimed. □

Once again, if $e = 0$ and ρ_0 is made sufficiently small, AM-IHT with RIP-1 achieves geometric convergence to the true signal x provided that $c_H > 1 - 1/(1+c_T)$. Thus, we have developed an analogue of AM-IHT that works purely with the RIP-1 assumption on the measurement matrix and hence is suitable for recovery using sparse matrices. It is likely that a similar analogue can be developed for AM-CoSaMP, but we will not pursue this direction here.

2.6.5 Improved Recovery via Boosting

As stated in Sections 2.6.2 and 2.6.3, AM-IHT and AM-CoSaMP require stringent assumptions on the head- and tail-approximation factors c_H and c_T. The condition (2.19) indicates that for AM-IHT to converge, the head- and tail-approximation factors must be tightly coupled. Observe that by definition, c_T is no smaller than 1. Therefore, c_H must be at least $\sqrt{3}/2$. If c_T is large (i.e., if the tail-approximation oracle gives only a crude approximation), then the head-approximation oracle needs to be even more precise. For example, if $c_T = 10$, then $c_H > 0.995$, i.e., the head approximation oracle needs to be very accurate. Such a stringent condition can severely constrain the choice of approximation algorithms.

In this section, we overcome this barrier by demonstrating how to "boost" the approximation factor of any given head-approximation algorithm. Given a head-approximation algorithm with arbitrary approximation factor c_H, we can boost its approximation factor to any arbitrary constant $c'_H < 1$. Our approach requires only a constant number of invocations of the original head-approximation algorithm and inflates the sample complexity of the resulting output model only by a constant factor. Combining this boosted head-approximation algorithm with AM-IHT or AM-CoSaMP, we can provide an overall recovery scheme for approximation algorithms with *arbitrary* approximation constants c_T and c_H. This is a much weaker condition than (2.19) and therefore significantly extends the scope of our framework for model-based compressive sensing with approximate projection oracles.

We achieve this improvement by iteratively applying the head-approximation algorithm to the residual of the currently selected support. Each iteration guarantees that we add

Algorithm 6 Boosting for head-approximation algorithms

```
1: function BoostHead($x, H, t$)
2:     $\Omega_0 \leftarrow \{\}$
3:     for $i \leftarrow 1, \ldots, t$ do
4:         $\Lambda_i \leftarrow H(x_{[n]\setminus\Omega_{i-1}})$
5:         $\Omega_i \leftarrow \Omega_{i-1} \cup \Lambda_i$
6:     end for
7:     return $\Omega_t$
8: end function
```

another c_H-fraction of the best remaining support to our result. Algorithm 6 contains the corresponding pseudo code and Theorem 2.27 the main guarantees.

Theorem 2.27. *Let H be a $(c_H, \mathbb{M}, \mathbb{M}_H, p)$-head-approximation algorithm with $0 < c_H \leq 1$ and $p \geq 1$. Then* BoostHead(x, H, t) *is a $((1-(1-c_H^p)^t)^{1/p}, \mathbb{M}, t \times \mathbb{M}_H, p)$-head-approximation algorithm. Moreover,* BoostHead *runs in time $O(t \cdot T_H)$, where T_H is the time complexity of H.*

Proof. Let $\Gamma \in \mathbb{M}$ be an optimal support, i.e., $\|x_\Gamma\|_p = \max_{\Omega\in\mathbb{M}} \|x_\Omega\|_p$. We now prove that the following invariant holds at the beginning of iteration i:

$$\|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}}\|_p^p \leq (1-c_H^p)^{i-1}\|x_\Gamma\|_p^p \; . \tag{2.26}$$

Note that the invariant (Equation 2.26) is equivalent to $\|x_{\Omega_{i-1}}\|_p^p \geq (1-(1-c_H^p)^{i-1})\|x_\Gamma\|_p^p$. For $i = t+1$, this gives the head-approximation guarantee stated in the theorem.

For $i = 1$, the invariant directly follows from the initialization.

Now assume that the invariant holds for an arbitrary $i \geq 1$. From line 4 we have

$$\begin{aligned}
\|(x_{[n]\setminus\Omega_{i-1}})_{\Lambda_i}\|_p^p &\geq c_H^p \max_{\Omega\in\mathbb{M}} \|(x_{[n]\setminus\Omega_{i-1}})_\Omega\|_p^p \\
\|x_{\Lambda_i\setminus\Omega_{i-1}}\|_p^p &\geq c_H^p \max_{\Omega\in\mathbb{M}} \|(x - x_{\Omega_{i-1}})_\Omega\|_p^p \\
&\geq c_H^p \|(x - x_{\Omega_{i-1}})_\Gamma\|_p^p \\
&= c_H^p \|x_\Gamma - x_{\Omega_{i-1}\cap\Gamma}\|_p^p \\
&= c_H^p (\|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}\cap\Gamma}\|_p^p) \\
&\geq c_H^p (\|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}}\|_p^p) \; .
\end{aligned} \tag{2.27}$$

We now prove the invariant for $i+1$:

$$\begin{aligned}
\|x_\Gamma\|_p^p - \|x_{\Omega_i}\|_p^p &= \|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}}\|_p^p - \|x_{\Lambda_i\setminus\Omega_{i-1}}\|_p^p \\
&\leq \|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}}\|_p^p - c_H^p(\|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}}\|_p^p) \\
&= (1-c_H^p)(\|x_\Gamma\|_p^p - \|x_{\Omega_{i-1}}\|_p^p) \\
&\leq (1-c_H^p)^{i+1}\|x_\Gamma\|_p^p \; .
\end{aligned}$$

The second line follows from (2.27) and the fourth line from the invariant.

Since $\Lambda_i \in \mathbb{M}_H$, we have $\Omega_t \in t \times \mathbb{M}_H$. The time complexity of BoostHead follows directly from the definition of the algorithm. □

We now use Theorem 2.27 to relax the conditions on c_T and c_H in Corollary 2.21. As before, we assume that we have compressive measurements of the form $y = Ax + e$, where $x \in \mathcal{M}$ and e is arbitrary measurement noise.

Corollary 2.28. *Let T and H be approximate projection oracles with $c_T \geq 1$ and $0 < c_H < 1$. Moreover, let δ be the model-RIP constant of the measurement matrix A and let*

$$\gamma = \frac{\sqrt{1 - \left(\frac{1}{1+c_T} - \delta\right)^2} + \delta}{1 - \delta}\,,$$
$$t = \left\lceil \frac{\log(1 - \gamma^2)}{\log(1 - c_H^2)} \right\rceil + 1\,.$$

We assume that δ is small enough so that $\gamma < 1$ and that A satisfies the model-RIP for $\mathbb{M} + \mathbb{M}_T + t \times \mathbb{M}_H$. Then AM-IHT with T and BoostHead*(x, H, t) as projection oracles returns a signal estimate $\widehat{x}$ satisfying*

$$\|x - \widehat{x}\|_2 \leq C\|e\|_2$$

after $O(\log \frac{\|x\|_2}{\|e\|_2})$ iterations. The constants in the error and runtime bounds depend only on c_T, c_H, and δ.

Proof. In order to use Corollary 2.21, we need to show that $\alpha < 1$. Recall that

$$\alpha = (1 + c_T)(\delta + \sqrt{1 - (c_H(1 - \delta) - \delta)^2})\,.$$

A simple calculation shows that a head-approximation oracle with $c'_H > \gamma$ achieves $\alpha < 1$.

Theorem 2.27 shows that boosting the head-approximation oracle H with t' iterations gives a head-approximation factor of

$$c'_H = \sqrt{1 - (1 - c_H^2)^{t'}}\,.$$

Setting $t' = t$ as defined in the theorem yields $c'_H > \gamma$. We can now invoke Corollary 2.21 for the recovery guarantee of AM-IHT. □

Analogous corollaries can be proven for AM-CoSaMP (Section 2.6.3) and AM-IHT with RIP-1 (Section 2.6.4). We omit detailed statements of these results here.

2.6.6 A fast model-RIP matrix

Besides sample complexity, a second crucial property of a good measurement matrix in compressive sensing is its *time complexity.* To be precise, we are interested in the time complexity of a *matrix-vector* multiplication with the measurement matrix $A \in \mathbb{R}^{m \times n}$ and an arbitrary vector $x \in \mathbb{R}^n$. Since many compressive sensing recovery algorithms are gradient based methods (such as ours in the previous sections), each iteration requires at least one matrix-vector multiplication. Hence the running time of this operation is an important component of the overall running time.

Note that the time complexity of the measurement matrix is particularly important when we use approximate projections. With exact projections, the projection step in each iteration usually dominates the overall running time and hence the matrix-vector multiplication is a lower-order term. But when we achieve a *nearly-linear* running time with our approximate projections, the time complexity of the measurement matrix can become a bottleneck. For instance, using a dense i.i.d. Gaussian matrix gives a running time of $O(n \cdot m)$ for a single matrix-vector multiplication, which can be significantly more than a nearly-linear time projection. Hence we now investigate the time complexity of measurement matrices in more detail.

For standard compressive sensing, researchers have identified several classes of measurement matrices that satisfy the RIP and enable fast matrix-vector multiplication in time $O(n \log n)$. The most common are based on sub-sampled Fourier measurements since such ensembles are also relevant for applications in medical imaging (MRI). For concrete results on fast measurement matrices, see for instance [53, 62, 75, 115, 172, 194].

In contrast, matrices known to satisfy the structured sparsity-equivalent of the RIP (also called *model-RIP*) only admit slow multiplication in time $O(nm)$ [36]. Since known recovery algorithms utilizing structured sparsity perform several matrix-vector multiplications, this can become a bottleneck in the overall time complexity. One approach to overcome this barrier is to use *sparse* matrices that satisfy the ℓ_1-variant of the RIP. However, recent work shows that this implies a lower bound of $m = \Omega(k \log \frac{n}{k} / \log\log \frac{n}{k})$ for the tree-sparsity model [126]. In contrast, dense Gaussian matrices achieve a sample complexity of $m = O(k)$.

Building on [172], we construct a measurement matrix which satisfies the model-RIP and enables multiplication in time

$$O(n \log n + k^2 \log n \log^2(k \log n))$$

for general k. For $k \leq n^{1/2-\mu}$, $\mu > 0$, the multiplication time is $O(n \log n)$. Moreover, our matrix has the same bound on the number of measurements as existing, slower model-RIP matrices: $m = O(k + \log|\mathbb{M}_k|)$. For instance, this simplifies to $m = O(k)$ for the tree sparsity model.

Ideally, a model-RIP matrix with $m = O(k + \log|\mathbb{M}_k|)$ rows would offer a multiplication time of $O(n \log n)$ for all values of k. However, we conjecture that such a result is connected to progress on the measurement bound for subsampled Fourier matrices in k-sparse compressive sensing. Note that improving this bound is a notoriously hard problem that has seen only slow progress over the past decade [53, 62, 75, 115, 172, 194].

2.6.6.1 Related work

There is a large body of work on matrices satisfying the RIP for general k-sparse vectors. For instance, see [35, 75, 104, 194] and references therein. For matrices with fast matrix-vector multiplication in $O(n \log n)$ time, the best known measurement bound is $m = O(k \log n \log^2 k)$ [115]. For $k \leq n^{1/2-\mu}$ and $\mu > 0$, there exist fast matrices with $m = O(k \log n)$ [11]. Note that in this regime, $O(k \log n) = O(k \log \frac{n}{k})$.

For the model-RIP, the only known matrices with $m = O(k + \log|\mathbb{M}_k|)$ are *dense* matrices with i.i.d. subgaussian entries [36]. Vector-matrix multiplication with such matrices requires $O(mn)$ time. While ℓ_1-model-RIP matrices support faster multiplication, they also entail a measurement lower bound of $m = \Omega(k \log \frac{n}{k} / \log\log \frac{n}{k})$ for the tree-sparsity model [126]. This is considered a challenging open problem in the field.

2.6.6.2 Our construction

Following the techniques of [172], we construct a matrix that supports fast matrix-vector multiplication and satisfies the model-RIP for the tree-sparsity model. In particular, we prove the following theorem.

Theorem 2.29. *There exists a randomized construction of $A \in \mathbb{R}^{m\times n}$, with optimal parameters $m = O(k)$, that satisfies the model-RIP for the tree-sparsity model $\mathcal{M}_k$. Moreover, the matrix A supports matrix-vector multiplication with complexity*

$$O(n \log n + k^2 \log n \log^2(k \log n))$$

for any $k \leq n$. For the regime $k \leq n^{1/2-\mu}$, this complexity can be refined to $O(n \log n)$.

Proof. We follow a two step-approach to construct A. First, from the results of [62, 75, 115, 172, 194], it is known that with high probability, one can construct matrices $F \in \mathbb{R}^{q\times n}$ with $q = O(k\mathrm{polylog} n)$ that satisfy the RIP over *all* sparse vectors in $\mathbb{R}^n$. We use the constructions described in [172], which satisfy the RIP with $q = O(k \log n \log^2(k \log n))$. Their proposed F is of the form SH, where $H \in \mathbb{R}^{n\times n}$ is a Fourier matrix and S is a *sparse* matrix with random ±1 elements as nonzeros.

For smaller values of k (in particular, for $k \leq n^{1/2-\mu}$ for any $\mu > 0$), an elegant (randomized) approach to construct such an F is described in [11]. Specifically, a suitable F can be obtained by concatenating independently chosen linear transformations of the form DH (where $H \in \mathbb{R}^{n\times n}$ is a Fourier or Hadamard matrix and $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix with random ±1 elements along the diagonal), followed by left multiplication with any row-orthonormal matrix (such as a row-selection matrix) of size $q \times n$, where $q = O(k \log n)$.

In either case, F provides a stable embedding of the set of all k-sparse signals into $\mathbb{R}^q$ with high probability. In other words, given any subset of indices $\Lambda \subset [n]$ with cardinality k, the following relation holds for all vectors x supported on Λ:

$$(1 - \delta_F)\|x\|_2^2 \leq \|Fx\|_2^2 \leq (1 + \delta_F)\|x\|_2^2$$

for some small constant δ_F.

Next, consider a random matrix $G \in \mathbb{R}^{m\times q}$ that satisfies the following *concentration-of-measure* property: for any $v \in \mathbb{R}^q$, the following holds:

$$\mathbb{P}(|\|Gv\|_2^2 - \|v\|_2^2| \geq \epsilon\|v\|_2^2) \leq 2e^{-c\frac{n}{2}\varepsilon^2}, \quad \forall\, \varepsilon \in (0, 1/3)\ . \tag{2.28}$$

Again, it is known that a matrix $G = \frac{1}{\sqrt{m}}\bar{G}$, with the elements of $\bar{G} \in \mathbb{R}^{m\times q}$ drawn from a standard normal distribution, satisfy (2.28). Now, choose any index set $\Lambda \in \mathbb{M}_k$ belonging to the tree-sparsity model, and a small constant $\delta_G > 0$. From Lemma 2.1 of [182], the following property holds for all x supported on Λ: if $\delta := \delta_F + \delta_G + \delta_F \delta_G$, then

$$(1-\delta)\|x\|_2^2 \leq \|GFx\|_2^2 \leq (1+\delta)\|x\|_2^2$$

with probability exceeding

$$1 - 2\left(1 + \frac{12}{\delta_G}\right)^k e^{-\frac{c}{9}\delta_G^2 m} \ .$$

In other words, for signals with a given support set $\Lambda \in \mathbb{M}_k$, the probability that GF fails to have a isometry constant δ is no greater than $2\left(1 + \frac{12}{\delta_G}\right)^k e^{-\frac{c}{9}\delta_G^2 m}$. The total number of supports Λ in the tree-sparsity model can be upper bounded by $(2e)^k/(k+1)$ [36]. Therefore, performing a union bound over all possible Λ, the probablity that GF fails to have an isometry constant δ over the model $\mathcal{M}_k$ is upper bounded by

$$2\frac{(2e)^k}{k+1}\left(1 + \frac{12}{\delta_G}\right)^k e^{-\frac{c}{9}\delta_G^2 m} \ . \tag{2.29}$$

Choosing $m = O(k)$ and δ_G sufficiently small, (2.29) can be made exponentially small. Therefore, with high probability, $A = G \cdot F$ satisfies the RIP over all signals belonging to the model $\mathcal{M}_k$, with $m = O(k)$ and a sufficiently small constant δ.

Multiplication of F with any vector $x \in \mathbb{R}^n$ incurs $O(n \log n)$ complexity, while multiplication of G with Fx incurs a complexity of $O(k \times q)$. Therefore, the overall complexity scales as $O(n \log n + kq)$. Substituting for the best available choices of F for different ranges of k, we obtain the stated result. □ □

Chapter 3

Graph Sparsity

3.1 Introduction

Over the past decade, sparsity has emerged as an important tool in several fields including signal processing, statistics, and machine learning. In compressive sensing, sparsity reduces the sample complexity of measuring a signal, and statistics utilizes sparsity for high-dimensional inference tasks. In many settings, sparsity is a useful ingredient because it enables us to model structure in high-dimensional data while still remaining a mathematically tractable concept. For instance, natural images are often sparse when represented in a wavelet basis, and objects in a classification task usually belong to only a small number of classes.

Due to the success of sparsity, a natural question is how we can refine the notion of sparsity in order to capture more complex structures. There are many examples where such an approach is applicable: (i) large wavelet coefficients of natural images tend to form connected *trees*, (ii) active genes can be arranged in functional *groups*, and (iii) approximate point sources in astronomical data often form *clusters*. In such cases, exploiting this additional structure can lead to improved compression ratio for images, better multi-label classification, or smaller sample complexity in compressive sensing and statistics. Hence an important question is the following: how can we model such sparsity structures, and how can we make effective use of this additional information in a computationally efficient manner?

There has been a wide range of work addressing these questions, e.g., [27, 46, 92, 96, 116, 122, 130, 140, 171, 181, 212, 242]. Usually, the proposed solutions offer a trade-off between the following conflicting goals:

Generality What range of sparsity structures does the approach apply to?

Statistical efficiency What statistical performance improvements does the use of structure enable?

Computational efficiency How fast are the resulting algorithms?

In this chapter, we introduce a framework for sparsity models defined through *graphs*, and we show that it achieves a compelling trade-off between the goals outlined above. At a high level, our approach applies to data with an underlying graph structure in which the large coefficients form a small number of connected components (optionally with additional constraints on the edges). Our approach offers three main features: (i) *Generality*: the framework encompasses several previously studied sparsity models, e.g., tree sparsity and cluster sparsity. (ii) *Statistical efficiency*: our sparsity model leads to reduced sample complexity in sparse recovery and achieves the information-theoretic optimum for a wide range of parameters. (iii) *Computational efficiency*: we give a nearly-linear time algorithm for our sparsity model, significantly improving on prior work both in theory and in practice. Due to the growing size of data sets encountered in science and engineering, algorithms with (nearly-)linear running time are becoming increasingly important.

We achieve these goals by connecting our sparsity model to the *prize collecting Steiner tree* (PCST) problem, which has been studied in combinatorial optimization and approximation algorithms. To establish this connection, we introduce a generalized version of the PCST problem and give a nearly-linear time algorithm for our variant. We believe that our sparsity model and the underlying algorithms are useful beyond sparse recovery, and we have already obtained results in this direction. To keep the presentation in this chapter coherent, we focus on our results for sparse recovery and briefly mention further applications in Section 3.7.

Before we present our theoretical results in Sections 3.3 to 3.5, we give an overview in Section 3.2. Section 3.6 complements our theoretical results with an empirical evaluation on both synthetic and real data (a background-subtracted image, an angiogram, and an image of text). We defer proofs and additional details to Section 3.8. Before we proceed, we briefly review some notation.

Basic notation Let $[d]$ be the set $\{1, 2, \ldots, d\}$. We say that a vector $\theta \in \mathbb{R}^d$ is s-sparse if at most s of its coefficients are nonzero. The support of θ contains the indices corresponding to nonzero entries in θ, i.e., $\operatorname{supp}(\theta) = \{i \in [d] \,|\, \theta_i \neq 0\}$. Given a subset $S \subseteq [d]$, we write θ_S for the restriction of θ to indices in S: we have $(\theta_S)_i = \theta_i$ for $i \in S$ and $(\theta_S)_i = 0$ otherwise. The ℓ_2-norm of θ is $\|\theta\| = \sqrt{\sum_{i\in[d]} \theta_i^2}$.

Sparsity models In some cases, we have more information about a vector than only "standard" s-sparsity. A natural way of encoding such additional structure is via *sparsity models* [36]: let $\mathbb{M}$ be a family of supports, i.e., $\mathbb{M} = \{S_1, S_2, \ldots, S_L\}$ where $S_i \subseteq [d]$. Then the corresponding sparsity model $\mathcal{M}$ is the set of vectors supported on one of the S_i:

$$\mathcal{M} \; = \; \{\theta \in \mathbb{R}^d \,|\, \operatorname{supp}(\theta) \subseteq S \text{ for some } S \in \mathbb{M}\} \; . \tag{3.1}$$

3.2 Our contributions

We state our main contributions in the context of sparse recovery (see Section 3.7 for further applications). Our goal is to estimate an unknown s-sparse vector $\theta \in \mathbb{R}^d$ from observations of the form

$$y \; = \; X\theta + e \; , \tag{3.2}$$

where $X \in \mathbb{R}^{n\times d}$ is the design matrix, $y \in \mathbb{R}^n$ are the observations, and $e \in \mathbb{R}^n$ is an observation noise vector. By imposing various assumptions on X and e, sparse recovery

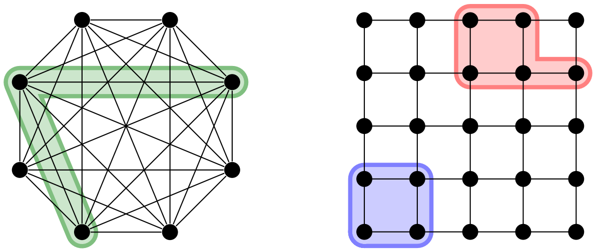

(a) s-sparsity (b) Cluster sparsity

Figure (3-1): Two examples of the weighted graph model. (a) In a complete graph, any s-sparse support can be mapped to a single tree ($g = 1$). (b) Using a grid graph, we can model a small number of clusters in an image by setting g accordingly. For simplicity, we use unit edge weights and set $B = s - g$ in both examples.

encompasses problems such as sparse linear regression and compressive sensing.

3.2.1 Weighted graph model (WGM)

The core of our framework for structured sparsity is a novel, **general** sparsity model which we call the *weighted graph model*. In the WGM, we use an underlying graph $G = (V, E)$ defined on the coefficients of the unknown vector θ, i.e., $V = [d]$. Moreover, the graph is weighted and we denote the edge weights with $w : E \to \mathbb{N}$. We identify supports $S \subseteq [d]$ with subgraphs in G, in particular *forests* (unions of individual trees). Intuitively, the WGM captures sparsity structures with a small number of connected components in G. In order to control the sparsity patterns, the WGM offers three parameters:

- s, the total sparsity of S.
- g, the maximum number of connected components formed by the forest F corresponding to S.
- B, the bound on the total weight $w(F)$ of edges in the forest F corresponding to S.

More formally, let $\gamma(H)$ be the number of connected components in a graph H. Then we can define the WGM:

Definition 3.1. *The* (G, s, g, B)*-WGM is the set of supports*

$$\begin{aligned}\mathbb{M} = \{S \subseteq [d] \mid |S| = s \textit{ and there is a } F \subseteq G \\ \textit{with } V_F = S,\ \gamma(F) = g,\ \textit{and } w(F) \leq B\}\ .\end{aligned} \tag{3.3}$$

Fig. 3-1 shows how two sparsity structures can be encoded with the WGM. Since our sparsity model applies to *arbitrary* graphs G, it can describe a wide range of structures. In particular, the model generalizes several previously studied sparsity models, including 1D-clusters, (wavelet) tree hierarchies, the Earth Mover Distance (EMD) model, and the unweighted graph model (see Table 3.1).

3.2.2 Recovery of vectors in the WGM

We analyze the **statistical efficiency** of our framework in the context of sparse recovery. In particular, we prove that the sample complexity of recovering vectors in the WGM is provably smaller than the sample complexity for "standard" s-sparse vectors. To formally state this result, we first introduce a key property of graphs.

Definition 3.2. *Let $G = (V, E)$ be a weighted graph with edge weights $w : E \to \mathbb{N}$. Then the weight-degree $\rho(v)$ of a node v is the largest number of adjacent nodes connected by edges with the same weight, i.e.,*

$$\rho(v) \;=\; \max_{b \in \mathbb{N}} \left|\{(v', v) \in E \,|\, w(v', v) = b\}\right| \,. \tag{3.4}$$

We define the weight-degree of G to be the maximum weight-degree of any $v \in V$.

Note that for graphs with uniform edge weights, the weight-degree of G is the same as the maximum node degree. Intuitively, the (weight) degree of a graph is an important property for quantifying the sample complexity of the WGM because the degree determines how restrictive the bound on the number of components g is. In the extreme case of a complete graph, any support can be formed with only a single connected component (see Figure 3-1). Using Definitions 3.1 and 3.2, we now state our sparse recovery result (see Theorem 3.12 in Section 3.5 for a more general version):

Theorem 3.3. *Let $\theta \in \mathbb{R}^d$ be in the (G, s, g, B)-WGM. Then*

$$n = O(s(\log \rho(G) + \log \frac{B}{s}) + g \log \frac{d}{g}) \tag{3.5}$$

i.i.d. Gaussian observations suffice to estimate θ. More precisely, let $e \in \mathbb{R}^n$ be an arbitrary noise vector and let $y \in \mathbb{R}^n$ be defined as in Eq. 3.2 where X is an i.i.d. Gaussian matrix. Then we can efficiently find an estimate $\hat{\theta}$ such that

$$\|\theta - \hat{\theta}\| \;\leq\; C\|e\| \,, \tag{3.6}$$

where C is a constant independent of all variables above.

Note that in the noiseless case ($e = 0$), we are guaranteed to recover θ exactly. Moreover, our estimate $\hat{\theta}$ is in a slightly enlarged WGM for any amount of noise, see Section 3.5. Our bound (3.5) can be instantiated to recover previous sample complexity results, e.g., the $n = O(s \log \frac{d}{s})$ bound for "standard" sparse recovery, which is tight [88].[1] For the image grid graph example in Figure 3-1, Equation (3.5) becomes $n = O(s + g \log \frac{d}{g})$, which matches the information-theoretic optimum $n = O(s)$ as long as the number of clusters is not too large, i.e., $g = O(s / \log d)$.[2]

[1] To be precise, encoding s-sparsity with a complete graph as in Figure 3-1 gives a bound of $n = O(s \log d)$. To match the $\log \frac{d}{s}$ term, we can encode s-sparsity as $g = s$ clusters of size one in a fully disconnected graph with no edges.

[2] Optimality directly follows from a simple dimensionality argument: even if the s-sparse support of the vector θ is known, recovering the unknown coefficients requires solving a linear system with s unknowns uniquely. For this, we need at least s linear equations, i.e., s observations.

3.2.3 Efficient projection into the WGM

The algorithmic core of our sparsity framework is a **computationally efficient** procedure for projecting arbitrary vectors into the WGM. More precisely, the model-projection problem is the following: given a vector $b \in \mathbb{R}^d$ and a WGM $\mathcal{M}$, find the best approximation to b in $\mathcal{M}$, i.e.,

$$P_{\mathcal{M}}(b) \;=\; \operatorname*{arg\,min}_{b' \in \mathcal{M}} \|b - b'\| \, . \tag{3.7}$$

If such a model-projection algorithm is available, one can instantiate the framework of [36] in order to get an algorithm for sparse recovery with the respective sparsity model.[1] However, solving Problem (3.7) exactly is NP-hard for the WGM due to a reduction from the classical Steiner tree problem [137]. To circumvent this hardness result, we use our *approximation-tolerant* framework from Chapter 2. Instead of solving (3.7) exactly, our framework requires *two* algorithms with the following complementary approximation guarantees.

Tail approximation: Find an $S \in \mathbb{M}$ such that

$$\|b - b_S\| \le c_T \cdot \min_{S' \in \mathbb{M}} \|b - b_{S'}\| \, . \tag{3.8}$$

Head approximation: Find an $S \in \mathbb{M}$ such that

$$\|b_S\| \ge c_H \cdot \max_{S' \in \mathbb{M}} \|b_{S'}\| \, . \tag{3.9}$$

Here, $c_T > 1$ and $c_H < 1$ are arbitrary, fixed constants. Note that a head approximation guarantee does not imply a tail guarantee (and vice versa). In fact, stable recovery is not possible with only one type of approximate projection guarantee (recall Section 2.5). We provide two algorithms for solving (3.8) and (3.9) (one per guarantee) which both run in *nearly-linear time*.

Our model-projection algorithms are based on a connection to the prize-collecting Steiner tree problem (PCST), which is a generalization of the classical Steiner tree problem. Instead of finding the cheapest way to connect *all* terminal nodes in a given weighted graph, we can instead omit some terminals from the solution and pay a specific price for each omitted node. The goal is to find a subtree with the optimal trade-off between the cost paid for edges used to connect a subset of the nodes and the price of the remaining, unconnected nodes (see Section 3.3 for a formal definition).

We make the following three main algorithmic contributions. Due to the wide applicability of the PCST problem, we believe that these algorithms can be of independent interest (see Section 3.7).

- We introduce a variant of the PCST problem in which the goal is to find a set of g trees instead of a single tree. We call this variant the prize-collecting Steiner forest (PCSF) problem and adapt the algorithm of [110] for this variant.
- We reduce the projection problems (3.8) and (3.9) to a small set of adaptively constructed PCSF instances.

[1] Note that the framework does not supply general projection algorithms. Instead, the model-projection algorithms have to be designed from scratch for each model.

Model	Best previous sample complexity	Our sample complexity	Best previous running time	Our running time
1D-cluster [67]	$O(s + g \log \frac{d}{g})$	$O(s + g \log \frac{d}{g})$	$O(d \log^2 d)$	$O(d \log^4 d)$
Trees [121]	$O(s)$	$O(s)$	$O(d \log^2 d)$	$O(d \log^4 d)$
EMD [119]	$O(s \log \frac{B \log \frac{s}{w}}{s})$	$\boldsymbol{O(s \log \frac{B}{s})}$	$O(sh^2 B \log d)$	$\boldsymbol{O(wh^2 \log^4 d)}$
Clusters [122]	$O(s + g \log d)$	$\boldsymbol{O(s + g \log \frac{d}{g})}$	$O(d^\tau)$	$\boldsymbol{O(d \log^4 d)}$

Table (3.1): Results of our sparsity framework applied to several sparsity models.

In order to simplify the running time bounds, we assume that all coefficients are polynomially bounded in d, and that $s \leq d^{1/2-\mu}$ for some $\mu > 0$. For the graph cluster model, we consider the case of graphs with constant degree. The exponent τ depends on the degree of the graph and is always greater than 1. The parameters w and h are specific to the EMD model, see Chapter 6 for details. We always have $w \cdot h = d$ and $s \geq w$. Our sparsity framework improves on the sample complexity and running time of both the EMD and graph cluster models (bold entries).

- We give a nearly-linear time algorithm for the PCSF problem and hence also the model projection problem.

3.2.4 Improvements for existing sparsity models

Our results are directly applicable to several previously studied sparsity models that can be encoded with the WGM. Table 3.1 summarizes these results. In spite of its generality, our approach at least matches the sample complexity of prior work in all cases and actually offers an improvement for the EMD model. Moreover, our running time is always within a polylogarithmic factor of the best algorithm, even in the case of models with specialized solvers such as tree sparsity. For the EMD and cluster models, our algorithm is significantly faster than prior work and improves the time complexity by a polynomial factor. To complement these theoretical results, our experiments in Section 3.6 show that our algorithm is more than one order of magnitude faster than previous algorithms with provable guarantees and offers a better sample complexity in many cases.

3.2.5 Comparison to related work

In addition to the "point-solutions" for individual sparsity models outlined above, there has been a wide range of work on general frameworks for utilizing structure in sparse recovery. The approach most similar to ours is [36], which gives a framework underlying many of the algorithms in Table 3.1. However, the framework has one important drawback: it does not come with a full recovery algorithm. Instead, the authors only give a recovery scheme that assumes the existence of a model-projection algorithm satisfying (3.7). Such an algorithm must be constructed from scratch for each model, and the techniques that have been used for various models so far are quite different. Our contribution can be seen as complementing the framework of [36] with a nearly-linear time projection algorithm that is applicable to a wide range of sparsity structures. This answers a question raised by the authors of [122], who also give a framework for structured sparsity with a universal and complete recovery algorithm. Their framework is applicable to a wide range of sparsity models, but the corresponding algorithm is significantly slower than ours, both in theory ("Graph clusters" in Table 3.1) and in practice (see Section 3.6). Moreover, our recovery algorithm shows more robust performance across different shapes of graph clusters.

Both of the approaches mentioned above use iterative greedy algorithms for sparse recovery. There is also a large body of work on combining M-estimators with convex regularizers that induce structured sparsity, e.g., see the surveys [26] and [232]. The work closest to ours is [130], which uses an overlapping group Lasso to enforce graph-structured sparsity (graph Lasso). In contrast to their approach, our algorithm gives more fine-grained control over the number of clusters in the graph. Moreover, our algorithm has better computational complexity, and to the best of our knowledge there are no formal results relating the graph structure to the sample complexity of the graph Lasso. Emprirically, our algorithm recovers an unknown vector with graph structure faster and from fewer observations than the graph Lasso (see Section 3.8.1).

3.3 The prize-collecting Steiner forest problem

We now establish our connection between prize-collecting Steiner tree (PCST) problems and the weighted graph model. First, we formally define the PCST problem: Let $G = (V, E)$ be an undirected, weighted graph with edge costs $c : E \to \mathbb{R}_0^+$ and node prizes $\pi : V \to \mathbb{R}_0^+$. For a subset of edges $E' \subseteq E$, we write $c(E') = \sum_{e \in E'} c(e)$ and adopt the same convention for node subsets. Moreover, for a node subset $V' \subseteq V$, let $\overline{V'}$ be the complement $\overline{V'} = V \setminus V'$. Then the goal of the PCST problem is to find a subtree $T = (V', E')$ such that $c(E') + \pi(\overline{V'})$ is minimized. We sometimes write $c(T)$ and $\pi(\overline{T})$ if the node and edge sets are clear from context.

Similar to the classical Steiner tree problem, PCST is NP-hard. Most algorithms with provable approximation guarantees build on the seminal work of [110] (GW), who gave an efficient primal-dual algorithm with the following guarantee:

$$c(T) + 2\pi(\overline{T}) \;\leq\; 2 \min_{T' \text{ is a tree}} c(T') + \pi(\overline{T'}) \; . \tag{3.10}$$

Note that the PCST problem already captures three important aspects of the WGM: (i) there is an underlying graph G, (ii) edges are weighted, and (iii) nodes have prizes. If we set the prizes to correspond to vector coefficients, i.e., $\pi(i) = b_i^2$, the term $\pi(\overline{T})$ in the PCST objective function becomes $\pi(\overline{T}) = \|b - b_T\|^2$, which matches the objective in the model-projection problems (3.8) and (3.9). However, there are two important differences. First, the objective in the PCST problem is to find a *single* tree T, while the WGM can contain supports defined by *multiple* connected components (if $g > 1$). Moreover, the PCST problem optimizes the trade-off $c(T) + \pi(\overline{T})$, but we are interested in minimizing $\|b - b_T\|$ subject to hard constraints on the support cardinality $|T|$ and the support cost $c(T)$ (the parameters s and B, respectively). In this section, we address the first of these two issues; Section 3.4 then completes the connection between PCST and the WGM. We begin by defining the following variant of the PCST problem.

Definition 3.4 (The prize-collecting Steiner forest problem). *Let $g \in \mathbb{N}$ be the target number of connected components. Then the goal of the prize-collecting Steiner forest (PCSF) problem is to find a subgraph $F = (V', E')$ with $\gamma(F) = g$ that minimizes $c(E') + \pi(\overline{V'})$.*

As defined in Section 3.2.1, $\gamma(F)$ is the number of connected components in the (sub-)graph F. To simplify notation in the rest of the chapter, we say that a forest F is a g-forest if $\gamma(F) = g$. There is always an optimal solution for the PCSF problem which consists of g trees because removing edges cannot increase the objective value. This allows us to employ the PCSF problem for finding supports in the WGM that consist of several connected components. In order to give a computationally efficient algorithm for the PCSF variant, we utilize prior work for PCST: (i) To show correctness of our algorithm, we prove that the GW scheme for PCST can be adapted to our PCSF variant. (ii) To achieve a good time complexity, we show how to simulate the GW scheme in nearly-linear time.

3.3.1 The Goemans-Williamson (GW) scheme for PCSF

A useful view of the GW scheme is the "moat-growing" interpretation of [136], which describes the algorithm as an iterative clustering method that constructs "moats" around every cluster. These moats are essentially the dual variables in the linear program of the GW scheme. Initially, every node forms its own active cluster with a moat of size 0. The moats around each active cluster then grow at a uniform rate until one of the following two events occurs:

Cluster deactivation When the sum of moats in a cluster reaches the sum of node prizes in that cluster, the cluster is deactivated.

Cluster merge When the sum of moats that are intersected by an edge e reaches the cost of e, the clusters at the two endpoints of e are merged and e is added to the current solution.

The moat-growing stage of the algorithm terminates when only a single active cluster remains. After that, the resulting set of edges is pruned in order to achieve a provable approximation ratio. We generalize the proof of [99] and show that it is possible to extract more than one tree from the moat-growing phase as long as the trees come from different clusters. Our modification of GW terminates the moat-growing phase when exactly g active clusters remain, and we then apply the GW pruning algorithm to each resulting tree separately. This gives the following result.

Theorem 3.5. *There is an algorithm for the PCSF problem that returns a g-forest F such that*

$$c(F) + 2\pi(\overline{F}) \;\leq \min_{F' \subseteq G,\, \gamma(F') \leq g} 2c(F') + 2\pi(\overline{F'}) \;. \tag{3.11}$$

For $g = 1$, the theorem recovers the guarantee in (3.10). We defer the proof to Section 3.8.4.1.

3.3.2 A fast algorithm for Goemans-Williamson

While the modified GW scheme produces good approximate solutions, it is not yet sufficient for a nearly-linear time algorithm: we still need an efficient way of simulating the moat-growing phase. There are two main difficulties: (i) The remaining "slack" amounts on edges can shrink at different rates depending on how many of the edge endpoints are in active clusters. (ii) A single cluster event (merge or deactivation) can change this rate for up to $\Theta(|V|)$ edges. In order to maintain edge events efficiently, we use the dynamic edge splitting approach introduced by [76]. This technique essentially ensures that every edge always has at most one active endpoint, and hence its slack either shrinks at rate 1 or not at all. However, edge splitting introduces additional edge events that do not directly lead to a cluster merge. While it is relatively straightforward to show that every such intermediate edge event halves the remaining amount of slack on an edge, we still need an overall bound on the number of intermediate edge events necessary to achieve a given precision. For this, we prove the following new result about the GW moat growing scheme.

Theorem 3.6. *Let all edge costs $c(e)$ and node prizes $\pi(v)$ be even integers. Then all finished moats produced by the GW scheme have integer sizes.*

In a nutshell, this theorem shows that one additional bit of precision is enough to track all events in the moat-growing phase accurately. We prove the theorem via induction over the events in the GW scheme, see Section 3.8.5.2 for details. On the theoretical side, this result allows us to bound the overall running time of our algorithm for PCSF. Combined with suitable data structures, we can show the following:

Theorem 3.7. *Let α be the number of bits used to specify a single edge cost or node prize. Then there is an algorithm achieving the PCSF guarantee of Theorem 3.5 in time $O(\alpha \cdot |E| \log|V|)$.*

On the practical side, we complement Theorem 3.6 with a new adaptive edge splitting scheme that leads to a small number of intermediate edge events. Our experiments show that our scheme results in less than 3 events per edge on average (see Section 3.8.5.3).

3.4 Sparse approximation with the WGM

In order to utilize the WGM in sparse recovery, we employ the framework of [119]. As outlined in Section 3.2.3, the framework requires us to construct *two* approximation algorithms satisfying the head- and tail-approximation guarantees (3.8) and (3.9). We now give two such model-projection algorithms, building on our tools for PCSF developed in the previous section.

3.4.1 Tail-approximation algorithm

We can connect the PCSF objective to the WGM quantities by setting $\pi(i) = b_i^2$ and $c(e) = w(e) + 1$, which gives:

$$c(F) = w(F) + (|F| - g) \quad \text{and} \quad \pi(\overline{F}) = \|b - b_F\|^2 \ .$$

After multiplying the edge costs with a trade-off parameter λ, the PCSF objective $\lambda \cdot c(F) + \pi(\overline{F})$ essentially becomes a *Lagrangian relaxation* of the model-constrained optimization problem (3.8). We build our tail-approximation algorithm on top of this connection, starting with an algorithm for the "tail"-variant of the PCSF problem. By performing a binary search over the parameter λ (see Algorithm 7), we get a bicriterion guarantee for the final forest.

Algorithm 7 PCSF-Tail

```
Input: $G$, $c$, $\pi$, $g$, cost-budget $C$, parameters $\nu$ and $\delta$.
We write $c_\lambda(e) = \lambda \cdot c(e)$ .
$\pi_{\min} \leftarrow \min_{\pi(i)>0} \pi(i)$,    $\lambda_0 \leftarrow \frac{\pi_{\min}}{2C}$
$F \leftarrow$ PCSF-GW$(G, c_{\lambda_0}, \pi, g)$
if $c(F) \leq 2C$ and $\pi(\overline{F}) = 0$ then return $F$
$\lambda_r \leftarrow 0$,    $\lambda_l \leftarrow 3\pi(G)$,    $\varepsilon \leftarrow \frac{\pi_{\min}\delta}{C}$
while $\lambda_l - \lambda_r > \varepsilon$ do
    $\lambda_m \leftarrow (\lambda_l + \lambda_r)/2$
    $F \leftarrow$ PCSF-GW$(G, c_{\lambda_m}, \pi, g)$
    if $c(F) \geq 2C$ and $c(F) \leq \nu C$ then return $F$
    if $c(F) > \nu C$ then $\lambda_r \leftarrow \lambda_m$ else $\lambda_l \leftarrow \lambda_m$
end while
return $F \leftarrow$ PCSF-GW$(G, c_{\lambda_l}, \pi, g)$
```

Theorem 3.8. *Let $\nu > 2$ and $\delta > 0$. Then* PCSF-Tail *returns a g-forest $F \subseteq G$ such that $c(F) \leq \nu \cdot C$ and*

$$\pi(\overline{F}) \leq (1 + \frac{2}{\nu - 2} + \delta) \min_{\gamma(F')=g, c(F') \leq C} \pi(\overline{F'}) \ . \tag{3.12}$$

Theorem 3.8 does not give $c(F) \leq C$ exactly, but the cost of the resulting forest is still guaranteed to be within a constant factor of C. The framework of [119] also applies to projections into such slightly larger models. As we will see in Section 3.5, this increase by a constant factor does not affect the sample complexity.

For the trade-off between support size and support weight, we also make use of approximation. By scalarizing the two constraints carefully, i.e., setting $c(e) = w(e) + \frac{B}{s}$, we get the following result. The proofs of Theorems 3.8 and 3.9 can be found in Section 3.8.3.1.

Theorem 3.9. *Let $\mathcal{M}$ be a (G, s, g, B)-WGM, let $b \in \mathbb{R}^d$, and let $\nu > 2$. Then there is an algorithm that returns a support $S \subseteq [d]$ in the $(G,\, 2\nu \cdot s + g,\, g,\, 2\nu \cdot B)$-WGM satisfying* (3.8) *with $c_T = \sqrt{1 + 3/(\nu - 2)}$. Moreover, the algorithm runs in time $O(|E| \log^3 d)$.*

3.4.2 Head-approximation algorithm

For our head-approximation algorithm, we also use the PCSF objective as a Lagrangian relaxation of the model-constraint problem (3.9). This time, we multiply the node prizes instead of the edge costs with a parameter λ. We perform a binary search similar to Alg. 7, but the final step of the algorithm requires an extra subroutine. At the end of the binary search, we are guaranteed to have a forest with good "density" $\frac{\pi(F)}{c(F)}$, but the good forest could correspond to either the lower bound λ_l or the upper bound λ_r. In the latter case, we

Algorithm 8 Graph-CoSaMP

```
Input: $y$, $X$, $G$, $s$, $g$, $B$, number of iterations $t$.
$\hat{\theta}_0 \leftarrow 0$
for $i \leftarrow 1, \ldots, t$ do
    $b \leftarrow X^T(y - X\hat{\theta}_{i-1})$
    $S' \leftarrow \mathrm{supp}(\hat{\theta}_{i-1}) \cup \textsc{HeadApprox}'(b, G, s, g, B)$
    $z_{S'} \leftarrow X^{\dagger}_{S'} y, \quad z_{S'^C} \leftarrow 0$
    $S \leftarrow \textsc{TailApprox}(z, G, s, g, B)$
    $\hat{\theta}_i \leftarrow z_S$
end for
return $\hat{\theta} \leftarrow \hat{\theta}_i$
```

have no bound on the cost of the corresponding forest F_r. However, it is always possible to extract a high-density sub-forest with bounded cost from F_r:

Lemma 3.10. *Let T be a tree and $C' \leq c(T)$. Then there is a subtree $T' \subseteq T$ such that $c(T') \leq C'$ and $\pi(T') \geq \frac{C'}{6} \cdot \frac{\pi(T)}{c(T)}$. Moreover, we can find such a subtree T' in linear time.*

The algorithm first converts the tree into a list of nodes corresponding to a tour through T. Then we can extract a good sub-tree by either returning a single, high-prize node or sliding a variable-size window across the list of nodes. See Section 3.8.3.2 for details. Combining these components, we get a head-approximation algorithm with the following properties.

Theorem 3.11. *Let $\mathcal{M}$ be a (G, s, g, B)-WGM and let $b \in \mathbb{R}^d$. Then there is an algorithm that returns a support $S \subseteq [d]$ in the $(G, 2s + g, g, 2B)$-WGM satisfying* (3.9) *with $c_H = \sqrt{1/14}$. The algorithm runs in time $O(|E| \log^3 d)$.*

3.5 Application in sparse recovery

We now instantiate the framework of [119] to give a sparse recovery algorithm using the WGM. The resulting algorithm (see Alg. 8) is a variant of CoSaMP [170] and uses the head- and tail-approximation algorithms instead of the hard thresholding operators.[1] In order to state the corresponding recovery guarantee in full generality, we briefly review the definition of the (model-) restricted isometry property (RIP) [36, 61]. We say that a matrix X satisfies the $(\mathcal{M}, \delta)$-model-RIP if for all $\theta \in \mathcal{M}$:

$$(1 - \delta) \cdot \|\theta\|^2 \leq \|X\theta\|^2 \leq (1 + \delta) \cdot \|\theta\|^2 \ . \tag{3.13}$$

Theorem 3.12. *Let $\theta \in \mathbb{R}^d$ be in the (G, s, g, B)-WGM $\mathcal{M}$ and let $X \in \mathbb{R}^{n \times d}$ be a matrix satisfying the model-RIP for a $(G, c_1 s, g, c_2 B)$-WGM and a fixed constant δ, where c_1 and c_2 are also fixed constants. Moreover, let $e \in \mathbb{R}^n$ be an arbitrary noise vector and let $y \in \mathbb{R}^n$ be defined as in* (3.2)*. Then* Graph-CoSaMP *returns a $\hat{\theta}$ in the $(G, 5s, g, 5B)$-WGM such that $\|\theta - \hat{\theta}\| \leq c_3 \|e\|$, where c_3 is a fixed constant. Moreover,* Graph-CoSaMP *runs in*

[1] Strictly speaking, HeadApprox' is a "boosted" version of the head-approximation algorithm developed here. See [119] for details.

time

$$O((T_X + |E| \log^3 d) \log \frac{\|\theta\|}{\|e\|}) \, ,$$

where T_X is the time complexity of a matrix-vector multiplication with X.

In order to establish sample complexity bounds for concrete matrix ensembles (e.g., random Gaussian matrices as in Theorem 3.3), we use a result of [36] that relates the sample complexity of sub-Gaussian matrix ensembles to the size of the model, i.e., the quantity $|\mathbb{M}|$. More precisely, $n = O(s + \log|\mathbb{M}|)$ rows / observations suffice for such matrices to satisfy the model-RIP for $\mathcal{M}$ and a fixed constant δ. For the WGM, we use a counting argument to bound $|\mathbb{M}|$ (see Section 3.8.2). Together with Theorem 3.12, the following theorem establishes Theorem 3.3 from Section 3.2.2.

Theorem 3.13. *Let $\mathbb{M}$ be the set of supports in the (G, s, g, B)-WGM. Then*

$$\log|\mathbb{M}| \; = \; O(s(\log \rho(G) + \log \frac{B}{s}) + g \log \frac{d}{g}) \, .$$

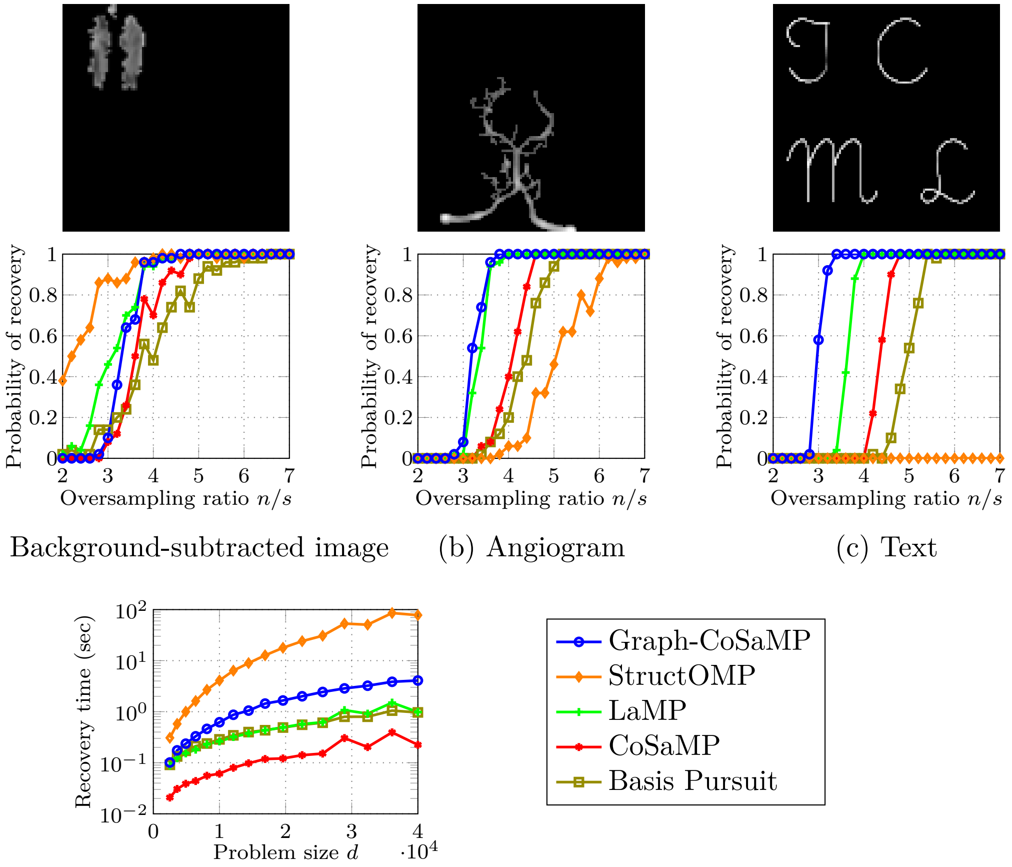
Next, we turn our attention to the running time of Graph-CoSaMP. Since our model-projection algorithms run in nearly-linear time, the matrix-vector products involving X can become the bottleneck in the overall time complexity:[1] for a dense Gaussian matrix, we have $T_X = \Omega(sd)$, which would dominate the overall running time. If we can control the design matrix (as is often the case in compressive sensing), we can use the construction of [121] to get a sample-optimal matrix with nearly-linear T_X in the regime of $s \leq d^{1/2-\mu}, \mu > 0$. Such a matrix then gives an algorithm with nearly-linear running time. Note that the bound on s is only a theoretical restriction in this construction: as our experiments show, a partial Fourier matrix empirically performs well for significantly larger values of s.

3.6 Experiments

We focus on the performance of our algorithm Graph-CoSaMP for the task of recovering 2D data with clustered sparsity. Multiple methods have been proposed for this problem, and our theoretical analysis shows that our algorithm should improve upon the state of the art (see Table 3.1). We compare our results to StructOMP [122] and the heuristic Lattice Matching Pursuit (LaMP) [66]. The implementations were supplied by the authors and we used the default parameter settings. Moreover, we ran two common recovery algorithms for "standard" s-sparsity: Basis Pursuit [60] and CoSaMP [170].

We follow a standard evaluation procedure for sparse recovery / compressive sensing: we record n observations $y = X\theta$ of the (vectorized) image $\theta \in \mathbb{R}^d$ using a subsampled Fourier matrix X. We assume that all algorithms possess prior knowledge of the sparsity s and the number of connected-components g in the true support of the image θ. We declare a trial successful if the squared ℓ_2-norm of the recovery error is at most 5% of the squared ℓ_2-norm of the original vector θ. The probability of successful recovery is then estimated by averaging over 50 trials. We perform several experiments with varying oversampling

[1] It is not necessary to compute a full pseudo-inverse $X^\dagger$. See [170] for details.



(a) Background-subtracted image

(b) Angiogram

(c) Text

(d) Running times

Figure (3-2): Sparse recovery experiments. The images in the top row are the original images θ. In the regime where the algorithms recover with high probability, the estimates $\hat{\theta}$ are essentially identical to the original images. Our algorithm Graph-CoSaMP achieves consistently good recovery performance and offers the best sample complexity for images (b) and (c). Moreover, our algorithm is about 20 times faster than StructOMP, the other method with provable guarantees for the image cluster model.

ratios n/s and three different images. See Section 3.8.1 for a description of the dataset, experiments with noise, and a comparison with the graph Lasso.

Figure 3-2 demonstrates that Graph-CoSaMP yields consistently competitive phase transitions and exhibits the best sample complexity for images with "long" connected clusters, such as the angiogram image (b) and the text image (c). While StructOMP performs well on "blob"-like images such as the background-subtracted image (a), its performance is poor in our other test cases. For example, it can successfully recover the text image only for oversampling ratios $n/s > 15$. Note that the performance of Graph-CoSaMP is very consistent: in all three examples, the phase transition occurs between oversampling ratios 3 and 4. Other methods show significantly more variability.

We also investigate the computational efficiency of Graph-CoSaMP. We consider resized versions of the angiogram image and record $n = 6s$ observations for each image size d. Figure 3-2(d) displays the recovery times (averaged over 50 trials) as a function of d. We observe that the runtime of Graph-CoSaMP scales nearly linearly with d, comparable to the conventional sparse recovery methods. Moreover, Graph-CoSaMP is about 20× faster than StructOMP.

3.7 Further applications

We expect our algorithms to be useful beyond sparse recovery and now briefly describe two promising applications.

Seismic feature extraction In [201], the authors use Steiner tree methods for a seismic feature extraction task. Our new algorithms for PCSF give a principled way of choosing tuning parameters for their proposed optimization problem. Moreover, our fast algorithms for PCSF can speed-up their method.

Event detection in social networks [191] introduce a method for event detection in social networks based on the PCST problem. Their method performs well but produces spurious results in the presence of multiple disconnected events because their PCST algorithm produces only a *single* tree instead of a forest. Our new algorithm for PCSF gives exact control over the number of trees in the solution and hence directly addresses this issue. Furthermore, the authors quote a running time of $O(|V|^2 \log|V|)$ for their GW scheme, so our nearly-linear time algorithm allows their method to scale to larger data.

3.8 Proofs and further results

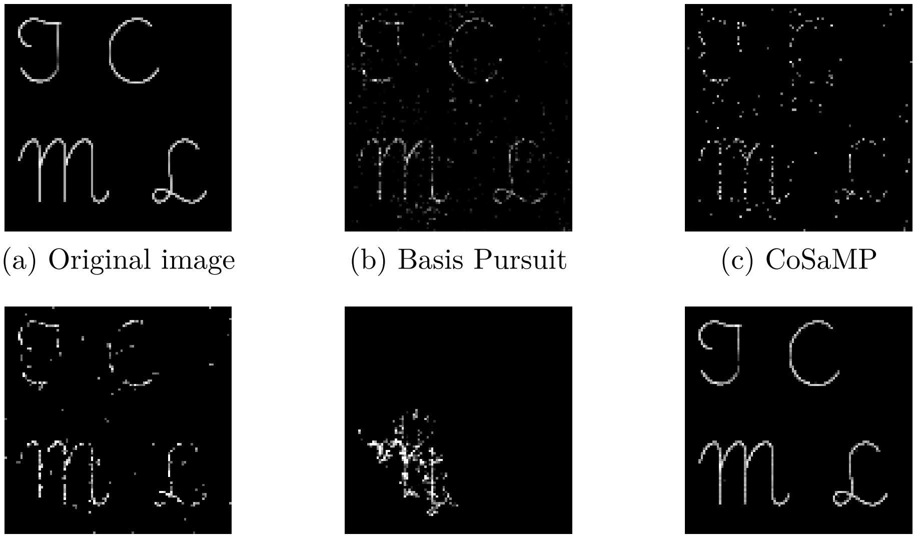
3.8.1 Further experimental results

We start with a more detailed description of our experimental setup. All three images used in Section 3.6 (Figure 3-2) are grayscale images of dimension 100 × 100 pixels with sparsity around 4% to 6%. The background-subtracted image was also used for the experimental evaluation in [122]. The angiogram image is a slightly sparsified version of the image on the Wikipedia page about angiograms;[1] it shows cerebral blood vessels. The text image was created by us.

We used SPGL1[2] as implementation of Basis Pursuit. The implementation of CoSaMP was written by us, closely following [170]. Graph-CoSaMP and CoSaMP share the same code, only the projection methods differ (hard s-thresholding for CoSaMP and our model projections for Graph-CoSaMP). Empirically it is not necessary to "boost" the head-approximation algorithm as strongly as suggested by the analysis in [119], we use only a single approximate model projection in place of HeadApprox' (see Alg. 8). The timing experiments in Figure 3-2(d) were conducted on a Windows machine with a 2.30 GHz Intel Core i7 CPU, 8 MB of cache, and 32 GB of RAM.

[1] `http://commons.wikimedia.org/wiki/File:Cerebral_angiography,_arteria_vertebralis_sinister_injection.JPG`

[2] `https://www.math.ucdavis.edu/~mpf/spgl1/`



(a) Original image (b) Basis Pursuit (c) CoSaMP

(d) LaMP (e) StructOMP (f) Graph-CoSaMP

Figure (3-3): Recovery examples for the text image (see Figure 3-2) and $n = 3.3s$ noisy linear observations using different recovery algorithms. Only Graph-CoSaMP is able to recover the image correctly.

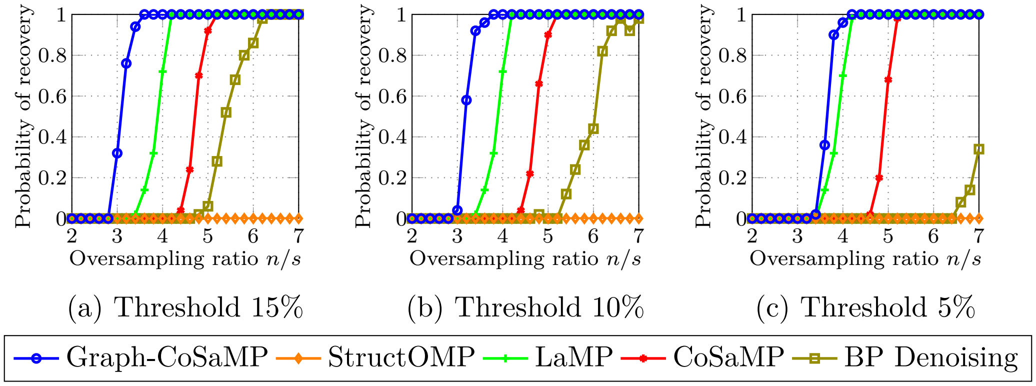
Recovered images In order to illustrate the outcomes of unsuccessful recovery trials, we show examples in the regime where Graph-CoSaMP recovers correctly but the other algorithms fail. This is the most relevant regime because it demonstrates that Graph-CoSaMP accurately recovers the image while other methods still show significant errors. See Figure 3-3 for the corresponding results.

Noise tolerance We also investigate the performance of the recovery algorithms in the noisy setting (the error term e in (3.2)). For this, we add Gaussian noise at a measurement-SNR level of roughly 15dB. Since we cannot hope for exact recovery in the noisy setting, we consider different tolerance levels for declaring a trial as successful (the ratio $\left\|\theta - \hat{\theta}\right\|^2/\|\theta\|^2$). Figure 3-4 contains the phase transition plots for the text image from Figure 3-2(c). The results show that our algorithm also gives the best performance for noisy observations.

Graph Lasso Next, we compare our approach to the graph Lasso introduced in [130]. Since the implementation in the SPArse Modeling toolbox (SPAMS)[1] focuses on dense design matrices, we limit our experiments to a smaller image than those in Figure 3-2. In particular, we use a 30×30 pixel synthetic image similar to the experiment in Section 9.3 of [130]. The nonzeros form a 5×5 square and hence correspond to a single component in the underlying grid graph. As suggested in [130], we encode the graph structure by using all 4-cycles as groups and use the variable replication approach to implement the overlapping group penalty.

We record n observations $y = X\theta$ with an i.i.d. Gaussian design matrix and follow the experimental procedure outlined in Section 3.6 (recovery threshold 5%, 50 trials per data

[1] `http://spams-devel.gforge.inria.fr/index.html`



(a) Threshold 15%

(b) Threshold 10%

(c) Threshold 5%

Figure (3-4): Phase transitions for successful recovery under noisy observations. The three plots are for the same image (the text image from Fig. 3-2 (c)) but use different thresholds for declaring a trial as successful (the ratio $\|\theta - \hat{\theta}\|^2 / \|\theta\|^2$). Our algorithm offers the best performance for all thresholds.

point). See Figure 3-5 for our results. While the graph Lasso improves over Basis Pursuit, our algorithm Graph-CoSaMP recovers the unknown vector θ from significantly fewer observations. Moreover, our algorithm is significantly faster than this implementation of the graph Lasso via variable replication.[1] While there are faster algorithms for the overlapping group Lasso such as [161], the recovery performance of the graph Lasso only matches Graph-CoSaMP for $n/s \geq 5$. In this regime, Graph-CoSaMP is already almost as fast as an efficient implementation of Basis Pursuit (SPGL1).

3.8.2 Sparse recovery with the WGM

We now give proofs for theorems in Section 3.5. First, we establish our general sample complexity bound.

Theorem 3.13. *Let $\mathbb{M}$ be the set of supports in the (G, s, g, B)-WGM. Then*

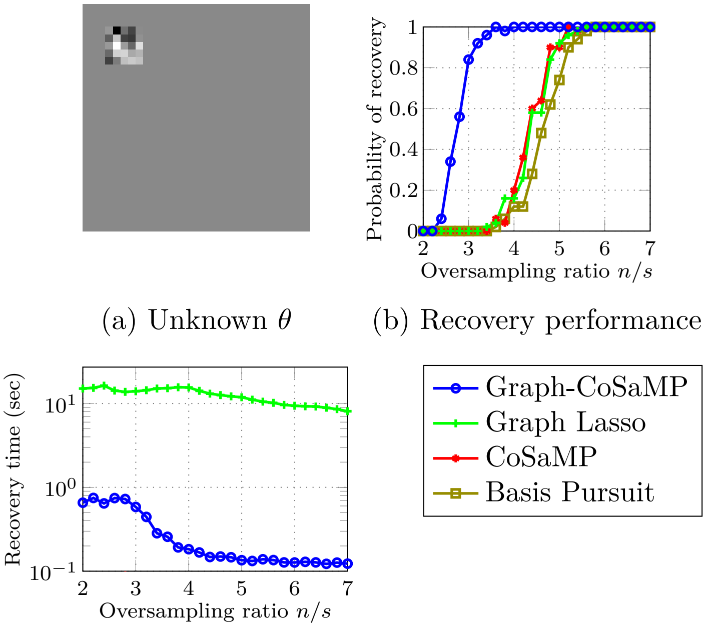
$$\log|\mathbb{M}| \;=\; O(s(\log \rho(G) + \log \frac{B}{s}) + g \log \frac{d}{g}) \; .$$

Proof. Note that every support in the WGM corresponds to a g-forests, which contains exactly $s - g$ edges. We prove the theorem by counting the possible locations of g tree roots in the graph G, and then the local arrangements of the $s - g$ edges in the g trees.

Consider the following process:

1. Choose g root nodes out of the entire graph. There are $\binom{d}{g}$ possible choices.
2. Consider the $s - g$ edges as an ordered list and distribute the total weight budget B to the edges. There are $\binom{B+s-g-1}{s-g}$ possible allocations.

[1] As suggested by the documentation of the SPAMS toolbox, we ran this set of experiments under Linux. The corresponding machine has an Intel Core 2 Duo CPU with 2.93 GHz, 3 MB of cache, and 8 GB of RAM.



(a) Unknown θ

(b) Recovery performance

(c) Running time

Figure (3-5): Comparison of our algorithm Graph-CoSaMP with the graph Lasso. Subfigure (a) shows the synthetic test image (30×30 pixels). Graph-CoSaMP recovers the vector θ from significantly fewer measurements than the other approaches (phase transition plot (b)). Moreover, Graph-CoSaMP is significantly faster than the variable replication implementation of the graph Lasso and essentially matches the performance of Basis Pursuit in the regime where both algorithms succeed ($n/s \geq 5$ in Subfigure (c)).

3. Assign a "target index" $t_e \in [\rho(G)]$ to each edge. There are $\rho(G)^{s-g}$ possible assignments. Note that the combination of edge weight and target index uniquely determines a neighbor of a fixed node v because there are at most $\rho(G)$ neighbors of v connected with edges of the same weight.

4. We now split the list of edges (together with their weight and target index assignments) into s sets. There are $\binom{2s-g}{s-1}$ possible partitions of the edge list.

We now have a list L consisting of s edge sets together with weight assignments and target indices. Moreover, we have a list of root nodes. We convert this structure to a g-forest (and hence a support in the WGM) according to the following rules, which essentially form a breadth-first search:

While there is a remaining root node, repeat the following:

1. Add the root node to a queue Q.
2. Initialize a new empty tree T_i.
3. While Q is non-empty, repeat the following
   (a) Let v be the first node in Q and remove v from Q.
   (b) Add v to T_i.
   (c) Let A be the first edge set in L and remove A from L.
   (d) For each pair of target index and weight in A, add the corresponding neighbor to Q.

Note that this process does not always succeed: for some weight allocations, there might be no neighbor connected by an edge with the corresponding weight. Nevertheless, it is easy to see that every possible support in the (G, s, g, B)-WGM can be constructed from at least one allocation via the process described above. Hence we have a surjection from the set of allocations to supports in the (G, s, g, B)-WGM $\mathbb{M}$, which gives the following bound:

$$|\mathbb{M}| \leq \binom{B+s-g-1}{s-g} \cdot \rho^{s-g} \cdot \binom{2s+g}{s-1} \cdot \binom{d}{g} .$$

Taking a logarithm on both sides and simplifying yields the bound in the theorem. □

The proof of the recovery result in Theorem 3.12 directly follows by combining the guarantees established for our tail- and head-approximation algortihms (Theorems 3.9 and 3.11) with the framework of [119].

Theorem 3.12. *Let $\theta \in \mathbb{R}^d$ be in the (G, s, g, B)-WGM $\mathcal{M}$ and let $X \in \mathbb{R}^{n\times d}$ be a matrix satisfying the model-RIP for a $(G, c_1 s, g, c_2 B)$-WGM and a fixed constant δ, where c_1 and c_2 are also fixed constants. Moreover, let $e \in \mathbb{R}^n$ be an arbitrary noise vector and let $y \in \mathbb{R}^n$ be defined as in (3.2). Then* Graph-CoSaMP *returns a $\hat{\theta}$ in the $(G, 5s, g, 5B)$-WGM such that $\|\theta - \hat{\theta}\| \leq c_3\|e\|$, where c_3 is a fixed constant. Moreover,* Graph-CoSaMP *runs in time*

$$O((T_X + |E|\log^3 d)\log\frac{\|\theta\|}{\|e\|}) ,$$

where T_X is the time complexity of a matrix-vector multiplication with X.

Proof. Note that both our head- and tail-approximation algorithms project into an output model with parameters bounded by constant multiples of s and B (we always maintain that the support corresponds to a g-forest), see Theorems 3.9 and 3.11. This allows us to use the CoSaMP version of Corollary 19 in [119] to establish the recovery result in our theorem. The claim about the running time follows from the near-linear running time of our model-projection algorithms and the running time analysis of CoSaMP in [170]. The $\log \frac{\|\theta\|}{\|e\|}$ term in the running time comes from the geometric convergence of Graph-CoSaMP. □

3.8.3 Approximate model-projection algorithms for the WGM

We now formally prove the head- and tail-approximation guarantees for our model-projection algorithms. We assume that we have access to an algorithm PCSF-GW for the PCSF problem with the approximation guarantee from Theorem 3.5, which we restate for completeness:

Theorem 3.5. *There is an algorithm for the PCSF problem that returns a g-forest F such that*

$$c(F) + 2\pi(\overline{F}) \;\leq \min_{F' \subseteq G,\, \gamma(F') \leq g} 2c(F') + 2\pi(\overline{F'}) \;. \tag{3.11}$$

We denote the running time of PCSF-GW with T_{PCSF}. See Section 3.8.4 for an algorithm that achieves guarantee (3.11) in nearly-linear time.

3.8.3.1 Tail-approximation

We first address the special case that there is a g-forest F^* with $c(F^*) \leq C$ and $\pi(\overline{F^*}) = 0$. In this case, we have to find a g-forest F with $\pi(\overline{F}) = 0$ in order to satisfy (3.12).

Lemma 3.14. *Let $\pi_{\min} = \min_{\pi(v)>0} \pi(v)$ and $\lambda_0 = \frac{\pi_{\min}}{2C}$. If there is a g-forest F^* with $c(F^*) \leq C$ and $\pi(\overline{F^*}) = 0$, then* PCSF-GW$(G, c_{\lambda_0}, \pi, g)$ *returns a g-forest F with $c(F) \leq 2C$ and $\pi(\overline{F}) = 0$.*

Proof. Applying the GW guarantee (3.11) gives

$$\begin{aligned} \lambda_0 \cdot c(F) + 2\pi(\overline{F}) \;&\leq\; 2\lambda_0 \cdot c_(F^*) + 2\pi(\overline{F^*}) \\ \pi(\overline{F}) \;&\leq\; \lambda_0 C \;=\; \frac{\pi_{\min}}{2} \;. \end{aligned}$$

Since $\pi_{\min} > 0$, we must have $\pi(\overline{F}) < \pi_{\min}$ and hence $\pi(\overline{F}) = 0$.

Applying (3.11) again then gives $c_{\lambda_0}(F) \leq 2c_{\lambda_0}(F^*)$, which shows that $c(F) \leq 2c(F^*) \leq 2C$ as desired. □

We can now proceed to prove an approximation guarantee for PCSF-Tail.

Theorem 3.8. *Let $\nu > 2$ and $\delta > 0$. Then* PCSF-Tail *returns a g-forest $F \subseteq G$ such that $c(F) \leq \nu \cdot C$ and*

$$\pi(\overline{F}) \;\leq (1 + \frac{2}{\nu - 2} + \delta) \min_{\gamma(F')=g, c(F') \leq C} \pi(\overline{F'}) \;. \tag{3.12}$$

Proof. We consider the three different cases in which PCSF-Tail returns a forest. Note that the resulting forest is always the output of PCSF-GW with parameter g, so the resulting forest is always a g-forest. To simplify notation, in the following we use

$$OPT = \min_{\gamma(F')=g, c(F')\leq C} \pi(\overline{F'}) \ .$$

First, if PCSF-Tail returns in Line 5, the forest F directly satisfies (3.12). Otherwise, there is no g forest F^* with $c(F^*) \leq C$ and $\pi(\overline{F^*}) = 0$ (contrapositive of Lemma 3.14). Hence in the following we can assume that $OPT \geq \pi_{\min}$.

If the algorithm returns in Line 10, we clearly have $c(F) \leq \nu \cdot C$. Moreover, the GW guarantee gives

$$\lambda_m \cdot c(F) + 2\pi(\overline{F}) \ \leq \ 2\lambda_m C + 2 \cdot OPT \ .$$

Since $c(F) \geq 2C$, we have $\pi(\overline{F}) \leq OPT$, satisfying (3.12).

Finally, consider the case that PCSF-Tail returns in Line 13. Let F_l and F_r be the forests corresponding to λ_l and λ_r, respectively. We show that the final output F_l satisfies the desired approximation guarantee if $\lambda_r - \lambda_l$ is small. Note that during the binary search, we always maintain the invariant $c(F_l) \leq 2C$ and $c(F_r) \geq \nu \cdot C$.

Using the GW guarantee and $\pi(\overline{F_r}) \geq 0$ gives $\lambda_r c(F_r) \leq 2\lambda_r C + 2 \cdot OPT$. Therefore,

$$\lambda_r \ \leq \ \frac{2 \cdot OPT}{c(F_r) - 2C} \ \leq \ \frac{2 \cdot OPT}{C(\nu - 2)} \ . \tag{3.14}$$

At the end of the binary search, we have $\lambda_l \leq \lambda_r + \varepsilon$. Combining this with (3.14) above and the GW guarantee (3.11) gives

$$\pi(\overline{F}) \ \leq \ \lambda_l C + OPT \ \leq \ OPT + (\lambda_r + \varepsilon)C \ \leq \ OPT + \frac{2 \cdot OPT}{\nu - 2} + \varepsilon C \ \leq \ (1 + \frac{2}{\nu - 2} + \delta) OPT \ .$$

In the last inequality, we used $OPT \geq \pi_{\min}$ and $\varepsilon = \frac{\pi_{\min}\delta}{C}$. This concludes the proof. □

Finally, we consider the running time of PCSF-Tail.

Theorem 3.15. PCSF-Tail *runs in time* $O(T_{\text{PCSF}} \cdot \log \frac{C \cdot \pi(G)}{\delta \cdot \pi_{\min}})$.

Proof. The time complexity is dominated by the number of calls to PCSF-GW. Hence we bound the number of binary search iterations in order to establish the overall time complexity. Let $\lambda^{(0)}$ be the initial value of λ_l in PCSF-Tail. Then the maximum number of iterations of the binary search is

$$\lceil \log \frac{\lambda_l^{(0)}}{\varepsilon} \rceil \ = \ \lceil \log \frac{3\pi(G) \cdot C}{\delta \cdot \pi_{\min}} \rceil \ = \ O(\log \frac{C \cdot \pi(G)}{\delta \cdot \pi_{\min}}) \ .$$

Since each iteration of the binary search takes $O(T_{\text{PCSF}})$ time, the time complexity stated in the theorem follows. □

If the node prizes π and edge costs c are polynomially bounded in $|V|$, the running time of PCSF-Tail simplifies to $O(T_{\text{PCSF}} \cdot \log|V|)$ for constant δ.

We now have all results to complete our tail-approximation algorithm for the WGM.

Theorem 3.9. *Let $\mathcal{M}$ be a (G, s, g, B)-WGM, let $b \in \mathbb{R}^d$, and let $\nu > 2$. Then there is an algorithm that returns a support $S \subseteq [d]$ in the $(G,\, 2\nu \cdot s + g,\, g,\, 2\nu \cdot B)$-WGM satisfying* (3.8) *with $c_T = \sqrt{1 + 3/(\nu - 2)}$. Moreover, the algorithm runs in time $O(|E| \log^3 d)$.*

Proof. We run the algorithm PCSF-Tail on the graph G with node prizes $\pi(i) = b_i^2$, edge costs $c(e) = w(e) + \frac{B}{s}$, a cost budget $C = 2B$, and the parameter $\delta = \min(\frac{1}{2}, \frac{1}{\nu})$. Let F be the resulting forest and S the corresponding support. The running time bound follows from combining Theorems 3.15 and 3.28.

First, we show that S is in the $(G,\, 2\nu \cdot s + g,\, g,\, 2\nu \cdot B)$-WGM. From Theorem 3.8 we know that F is a g-forest and that $c(F) \leq 2\nu \cdot B$. This directly implies that $w(F) \leq 2\nu \cdot B$. Moreover, the g-forest F has $|V_F| - g$ edges, all with cost at least $\frac{B}{s}$ because $w(e) \geq 0$ for all $e \in E$. Since $|V_F| = |S|$, this allows us to bound the sparsity of S as

$$(|S| - g) \cdot \frac{B}{s} \leq 2\nu \cdot B \; ,$$

which gives $|S| \leq 2\nu \cdot s + g$ as desired.

Now, let S^* be an optimal support in the (G, s, g, B)-WGM $\mathbb{M}$ and let F^* be a corresponding g-forest, i.e.,

$$\pi(\overline{F^*}) \;=\; \|b - b_{S^*}\|^2 \;=\; \min_{S' \in \mathbb{M}} \|b - b_{S'}\|^2 \; .$$

Then we have

$$\pi(\overline{F^*}) \;\geq\; \min_{\gamma(F') = g, c(F') \leq 2B} \pi(\overline{F'})$$

because by construction, every support in $\mathbb{M}$ corresponds to a g-forest with cost at most $2B$. Since $\pi(\overline{F}) = \|b - b_S\|^2$, applying guarantee (3.12) gives

$$\|b - b_S\|^2 \;\leq\; (1 + \frac{2}{\nu - 2} + \delta) \min_{S' \in \mathbb{M}} \|b - b_{S'}\|^2 \; .$$

Simplifying this inequality with our choice of δ then completes the proof. □

3.8.3.2 Head-approximation

We first state our head-approximation algorithm (see Alg. 9 and Alg. 10). In addition to a binary search over the Lagrangian parameter λ, the algorithm also uses the subroutines PruneTree and PruneForest in order to extract sub-forests with good "density" $\frac{\pi(F)}{c(F)}$.

We start our analysis by showing that PruneTree extracts sub-trees of good density $\frac{\pi(T')}{c(T')}$.

Lemma 3.10. *Let T be a tree and $C' \leq c(T)$. Then there is a subtree $T' \subseteq T$ such that $c(T') \leq C'$ and $\pi(T') \geq \frac{C'}{6} \cdot \frac{\pi(T)}{c(T)}$. Moreover, we can find such a subtree T' in linear time.*

Proof. We show that PruneTree satisfies the guarantees in the theorem. We use the definitions of L, π', c', and ϕ given in PruneTree (see Alg. 10). Moreover, let T' be the tree returned by PruneTree. First, note that PruneTree clearly runs in linear time by

Algorithm 9 Head approximation for the WGM: main algorithm PCSF-Head

```
function PCSF-Head($G, c, \pi, g, C, \delta$)
    We write $\pi_\lambda(i) = \lambda \cdot \pi(i)$ .
    $\pi_{\min} \leftarrow \min_{\pi(i)>0} \pi(i)$
    $\lambda_r \leftarrow \frac{2C}{\pi_{\min}}$
    $F \leftarrow$ PCSF-GW($G, c, \pi_{\lambda_r}, g$)
    if $c(F) \leq 2C$ then    ▷ Ensure that we have the invariant $c(F_r) > 2C$ (see Theorem 3.17)
        return $F$
    end if
    $\varepsilon \leftarrow \frac{\delta \cdot C}{2\pi(G)}$
    $\lambda_l \leftarrow \frac{1}{4\pi(G)}$
    while $\lambda_r - \lambda_l > \varepsilon$ do    ▷ Binary search over the Lagrange parameter $\lambda$
        $\lambda_m \leftarrow (\lambda_l + \lambda_r)/2$
        $F \leftarrow$ PCSF-GW($G, c, \pi_{\lambda_m}, g$)
        if $c(F) > 2C$ then
            $\lambda_r \leftarrow \lambda_m$
        else
            $\lambda_l \leftarrow \lambda_m$
        end if
    end while
    $F_l \leftarrow$ PCSF-GW($G, c, \pi_{\lambda_l}, g$)
    $F_r \leftarrow$ PCSF-GW($G, c, \pi_{\lambda_r}, g$)
    $F_r' \leftarrow$ PruneForest($F_r, c, \pi, C$)    ▷ Prune the potentially large solution $F_r$ (See Alg. 10)
    if $\pi(F_l) \geq \pi(F_r')$ then
        return $F_l$
    else
        return $F_r'$
    end if
end function
```

Algorithm 10 Head approximation for the WGM: subroutine PruneForest

```
function PruneForest($F, c, \pi, C$)
    Let $\{T_1, \ldots, T_{|F|}\}$ be the trees in $F$ sorted by $\frac{\pi(T_i)}{c(T_i)}$ descendingly.
    $C_r \leftarrow C$
    for $i \leftarrow 1, \ldots, |F|$ do
        if $C_r \geq c(T_i)$ then
            $T'_i \leftarrow T_i$
            $C_r \leftarrow C_r - c(T_i)$                          ▷ Cost budget $C^{(i)} = c(T_i)$
        else if $C_r > 0$ then
            $T'_i \leftarrow$ PruneTree($T_i, c, \pi, C_r$)
            $C_r \leftarrow 0$                                     ▷ Cost budget $C^{(i)} = C_r$
        else
            $T'_i \leftarrow \{\arg\max_{j \in T_i} \pi(j)\}$              ▷ Cost budget $C^{(i)} = 0$
        end if
    end for
    return $\{T'_1, \ldots, T'_{|F|}\}$
end function


function PruneTree($T, c, \pi, C'$)
    Let $L = (v_1, \ldots, v_{2|V_T|-1})$ be a tour through the nodes of $T$.        ▷ $T = (V_T, E_T)$
    Let $\pi'(j) = \begin{cases} \pi(v_j) & \text{if position } j \text{ is the first appearance of } v_j \text{ in } L \\ 0 & \text{otherwise} \end{cases}$
    Let $c'(P) = \sum_{i=1}^{|P|-1} c(P_i, P_{i+1})$
    Let $\phi = \frac{\pi(T)}{c(T)}$
    if there is a $v \in V_T$ with $\pi(v) \geq \frac{C' \cdot \phi}{6}$ then     ▷ Check if there is a single good node (cost is automatically 0)
        return the tree $\{v\}$
    end if
    $l \leftarrow 1$
    $P^1 = ()$                                                  ▷ Empty list
    for $i \leftarrow 1, \ldots, 2|V_T| - 1$ do                        ▷ Search for good sublists of $L$
        Append $i$ to $P^l$
        if $c'(P^l) > C'$ then                    ▷ Start a new sublist if the cost reaches $C$
            $l \leftarrow l + 1$
            $P^l \leftarrow ()$
        else if $\pi'(P^l) \geq \frac{C' \cdot \phi}{6}$ then                ▷ Return if we have found a good sublist
            return the subtree of $T$ on the nodes in $P^l$
        end if
    end for
    Merge $P^l$ and $P^{l-1}$ ▷ The algorithm will never reach this point (see Lemma 3.10).
end function
```

definition. Hence it remains to establish the approximation guarantee

$$\pi(T') \;\geq\; \frac{C'}{6} \cdot \frac{\pi(T)}{c(T)} \;=\; \frac{C' \cdot \phi}{6} \; .$$

Consider the first case in which PruneTree returns in line 23. Then T' is a tree consisting of a single node, so $c(T') = 0 \leq C'$. Moreover, we have $\pi(T') = \pi(v) \geq \frac{C' \cdot \phi}{6}$, which satisfies the guarantee in the theorem.

Next, we consider the case in which PruneTree returns in line 33. By definition of the algorithm, we have $c'(P^l) \leq C'$ and hence $c(T') \leq C'$ because the spanning tree T' of the nodes in P^l contains only edges that are also included at least once in $c'(P^l)$. Moreover, we have $\pi(T') \geq \pi'(P^l) \geq \frac{C' \cdot \phi}{6}$, so T' satisfies the guarantee in the theorem.

It remains to show that PruneTree always returns in one of the two cases above, i.e., never reaches line 36. We prove this statement by contradiction: assume that PruneTree reaches line 36. We first consider the partition of V_T induced by the lists P^i just before line 36. Note that there are no nodes $v \in V_T$ with $\pi(v) \geq \frac{C' \cdot \phi}{6}$ because otherwise PruneTree would have returned in line 23. Hence for every list P^i we have $\pi'(P^i) \leq \frac{C' \cdot \phi}{3}$ because the last element that was added to P^i can have increased $\pi'(P^i)$ by at most $\frac{C' \cdot \phi}{6}$, and we had $\pi'(P^i) < \frac{C' \cdot \phi}{6}$ before the last element was added to P^i because otherwise PruneTree would have returned in line 33. Moreover, every list P^i except P^l satisfies $c'(P^i) > C'$ by construction. Hence after merging the last two lists P^l and P^{l-1}, we have $c'(P^i) > C'$ for all P^i and also $\pi'(P^i) < \frac{C' \cdot \phi}{2}$.

We now derive the contradiction: note that all lists P^i have a low density $\frac{\pi'(P^i)}{c'(P^i)}$ but form a partition of the nodes in V_T. We can use this fact to show that the original tree had a density lower than $\frac{\pi(T)}{c(T)}$, which is a contradiction. More formally, we have

$$\pi(T) \;=\; \sum_{i=1}^{l-1} \pi'(P^i) \;<\; (l-1)\frac{C' \cdot \phi}{2}$$

and

$$2c(T) \;\geq\; \sum_{i=1}^{l-1} c'(P^i) \;>\; (l-1)C \; .$$

Combining these two inequalities gives

$$\frac{\phi}{2} \;=\; \frac{\pi(T)}{2c(T)} \;<\; \frac{(l-1)\frac{C'\phi}{2}}{(l-1)C'} \;=\; \frac{\phi}{2} \; ,$$

which is a contradiction. Hence PruneTree always returns in line 23 or 33 and satisfies the guarantee of the theorem. □

Extending the guarantee of PruneTree to forests is now straightforward: we can prune each tree in a forest F individually by assigning the correct cost budget to each tree. More formally, we get the following lemma.

Lemma 3.16. *Let F be a g-forest. Then* PRUNEFOREST(F, c, π, C) *returns a g-forest F' with $c(F') \leq C$ and*

$$\pi(F') \geq \frac{C}{6 \cdot c(F)} \pi(F) \; .$$

Proof. By construction, F' is a g-forest with $c(F) \leq C$. Let $C^{(i)}$ be the cost budget assigned to tree T_i' (see the comments in PRUNEFOREST). Using Lemma 3.10, we get

$$\pi(F') \; = \; \sum_{i=1}^{g} \pi(T_i') \; \geq \; \sum_{i=1}^{g} \frac{C^{(i)}}{6} \frac{\pi(T_i)}{c(T_i)} \; .$$

Note that the $C^{(i)}$ are the optimal allocation of budgets to the ratios $\frac{\pi(T_i)}{c(T_i)}$ with $0 \leq C^{(i)} \leq c(T_i)$ and $\sum_{i=1}^{g} C^{(i)} = C$. In particular, we have

$$\sum_{i=1}^{g} \frac{C^{(i)}}{6} \frac{\pi(T_i)}{c(T_i)} \; \geq \; \sum_{i=1}^{g} \frac{C \cdot \frac{c(T_i)}{c(F)}}{6} \frac{\pi(T_i)}{c(T_i)} \; = \; \frac{C}{6 \cdot c(F)} \pi(F) \; ,$$

which completes the proof. □

We can now prove our main theorem about PCSF-HEAD.

Theorem 3.17. *Let $0 < \delta < \frac{1}{13}$. Then* PCSF-HEAD *returns a g-forest F such that $c(F) \leq 2C$ and*

$$\pi(F) \; \geq \; (1 - \frac{12}{13(1-\delta)}) \max_{\gamma(F')=g,\, c(F') \leq C} \pi(F') \; . \tag{3.15}$$

Proof. Let F^* be an optimal g-forest with $c(F) \leq C$ and $\pi(F^*) = OPT$, where

$$OPT = \max_{\gamma(F')=g,\, c(F') \leq C} \pi(F') \; .$$

In this proof, the following rearranged version of the GW guarantee 3.11 will be useful:

$$\begin{aligned} c(F) + 2(\pi(G) - \pi(F)) \; &\leq \; 2C + 2(\pi(G) - \pi(F^*)) \\ \pi(F) \; &\geq \; \pi(F^*) + \frac{c(F) - 2C}{2} \; . \end{aligned} \tag{3.16}$$

As in the definition of PCSF-HEAD, we write π_λ for the node prize function $\pi_\lambda(i) = \lambda \cdot \pi(i)$. Using such modified node prizes, (3.16) becomes

$$\pi(F) \; \geq \; OPT + \frac{c(F) - 2C}{2\lambda} \; . \tag{3.17}$$

We now analyze two cases: either PCSF-HEAD returns in line 7 or in one of the lines 24 and 26. Note that in all cases, the returned forest F is a g-forest because it is produced by PCSF-GW (and PRUNEFOREST maintains this property).

First, we consider the case that the algorithm returns in line 7. Then by definition we have

$c(F) \leq 2C$. Moreover, the modified GW guarantee (3.17) gives

$$\pi(F) \;\geq\; OPT + \frac{c(F) - 2C}{2\lambda_r} \;\geq\; OPT - \frac{C}{\lambda_r} \;\geq\; OPT - \frac{\pi_{\min}}{2} \;\geq\; \frac{1}{2} OPT \;,$$

because clearly $OPT \geq \pi_{\min}$. Hence the guarantee in the theorem is satisfied.

Now, consider the case that the algorithm enters the binary search. Let F_l and F_r be the g-forests corresponding to λ_l and λ_r, respectively. During the binary search, we maintain the invariant that $c(F_l) \leq 2C$ and $c(F_r) > 2C$. Note that our initial choices for λ_l and λ_r satisfy this condition (provided the algorithm reaches the binary search).

When the algorithm terminates in line 24 or 26, we have $\lambda_r > \lambda_l > \lambda_r - \varepsilon$. Rearranging (3.17) gives

$$\begin{aligned} 2\lambda_r(\pi(F_r) - OPT) \;&\geq\; c(F_r) - 2C \\ \lambda_r \;&\geq\; \frac{c(F_r) - 2C}{2(\pi(F_r) - OPT)} \;. \end{aligned}$$

We now introduce a variable $\alpha \geq 2$ and distinguish two cases:

Case 1: Assume Assume $\frac{c(F_r)}{\pi(F_r)} > \alpha \frac{C}{OPT}$. Then Equation (3.17) gives

$$\begin{aligned} \lambda_r \;&\geq\; \frac{\frac{1}{2}\frac{c(F_r)}{OPT} - \frac{C}{OPT}}{\frac{\pi(F_r)}{OPT} - 1} \\ &= \frac{\frac{1}{2}\frac{\pi(F_r)}{OPT}\frac{c(F_r)}{\pi(F_r)} - \frac{C}{OPT}}{\frac{\pi(F_r)}{OPT} - 1} \\ &\geq \frac{\frac{1}{2}\frac{\pi(F_r)}{OPT}\alpha - 1}{\frac{\pi(F_r)}{OPT} - 1}\frac{C}{OPT} \\ &\geq \frac{\frac{\alpha}{2}\frac{\pi(F_r)}{OPT} - \frac{\alpha}{2}}{\frac{\pi(F_r)}{OPT} - 1}\frac{C}{OPT} \\ &= \frac{\alpha}{2}\frac{C}{OPT} \;. \end{aligned}$$

So we get $\lambda_l \;\geq\; \lambda_r - \varepsilon \;\geq\; \frac{\alpha}{2}\frac{C}{OPT} - \frac{\delta \cdot C}{2\pi(G)} \;\geq\; (1-\delta)\frac{\alpha C}{2OPT}$. We can now use this together with (3.17) to get:

$$\begin{aligned} \pi(F_l) \;&\geq\; OPT - \frac{C}{\lambda_l} \\ &\geq\; OPT - \frac{2OPT}{(1-\delta)\alpha} \\ &= (1 - \frac{2}{(1-\delta)\alpha})OPT \;. \end{aligned} \qquad (3.18)$$

Case 2: Assume $\frac{c(F_r)}{\pi(F_r)} \leq \alpha \frac{C}{OPT}$, which is equivalent to $\frac{\pi(F_r)}{c(F_r)} \geq \frac{1}{\alpha}\frac{OPT}{C}$. Since $c(F_r) > 2C$, we can invoke PruneForest on F_r with cost budget C. Let F_r' be the resulting g-forest.

From Lemma 3.16 we have $c(F'_r) \leq C$. Moreover,

$$\pi(F'_r) \;\geq\; \frac{C}{6 \cdot c(F_r)} \pi(F_r) \;=\; \frac{C}{6} \frac{\pi(F_r)}{c(F_r)} \;\geq\; \frac{1}{6\alpha} OPT \; . \tag{3.19}$$

Either case 1 or case 2 must hold. Since HEADAPPROX chooses the better forest among F_l and F'_r, we can combine Equations (3.18) and (3.19) to get the following guarantee on the final result F:

$$\pi(F) \;\geq\; \min(1 - \frac{2}{(1-\delta)\alpha}, \; \frac{1}{6\alpha}) OPT \; .$$

Choosing $\alpha = \frac{13}{6}$ to balance the two expressions (assuming δ is close to 0) then gives the approximation guarantee stated in the theorem. □

Next, we consider the running time of PCSF-HEAD.

Theorem 3.18. PCSF-HEAD *runs in time* $O(T_{\text{PCSF}} \cdot \log \frac{\pi(G)}{\delta \cdot \pi_{\min}})$.

Proof. As in Theorem 3.15 it suffices to bound the number of iterations of the binary search. Let $\lambda_r^{(0)}$ be the initial value of λ_r in PCSF-HEAD. Then the maximum number of iterations is

$$\lceil \log \frac{\lambda_r^{(0)}}{\varepsilon} \rceil \;=\; \lceil \log \frac{4 \cdot C \cdot \pi(G)}{\delta \cdot C \cdot \pi_{\min}} \rceil \;=\; O(\frac{\pi(G)}{\delta \cdot \pi_{\min}}) \; .$$

□

As before, the running time simplifies to $O(T_{\text{PCSF}} \cdot \log|V|)$ for constant δ if the node prizes and edge costs are polynomially bounded in the size of the graph.

We can now conclude with our head-approximation algorithm for the WGM.

Theorem 3.11. *Let $\mathcal{M}$ be a (G, s, g, B)-WGM and let $b \in \mathbb{R}^d$. Then there is an algorithm that returns a support $S \subseteq [d]$ in the $(G, 2s+g, g, 2B)$-WGM satisfying* (3.9) *with $c_H = \sqrt{1/14}$. The algorithm runs in time $O(|E| \log^3 d)$.*

Proof. We embed the WGM into a PCSF instance similar to Theorem 3.9: we run PCSF-HEAD on the graph G with node prizes $\pi(i) = b_i^2$, edge costs $c(e) = w(e) + \frac{B}{s}$, a cost budget $C = 2B$, and the parameter $\delta = \frac{1}{169}$. Let F be the resulting forest and S be the corresponding support. The running time bound follows from combining Theorems 3.18 and 3.28.

From Theorem 3.17 we directly have that F is a g-forest with $w(F) \leq 2B$. Following a similar argument as in Theorem 3.9, we also get $|S| \leq 2s+g$. So S is in the $(G, 2s+g, g, 2B)$-WGM.

Now, let S^* be an optimal support in the (G, s, g, B)-WGM $\mathbb{M}$ and let F^* be a corresponding g-forest, i.e.,

$$\pi(F^*) \;=\; \|b_{S^*}\|^2 \;=\; \max_{S' \in \mathbb{M}} \|b_{S'}\|^2 \; .$$

By construction, every support in $\mathbb{M}$ corresponds to a g-forest with cost at most $2B$. Hence we have

$$\pi(F^*) \;\leq\; \max_{\gamma(F')=g, c(F') \leq 2B} \pi(F')$$

Since $\pi(F) = \|b_S\|^2$, applying Theorem 3.17 gives

$$\|b_S\|^2 \;\geq\; (1 - \frac{12}{13(1-\delta)}) \max_{S' \in \mathbb{M}} \|b_{S'}\|^2 \; .$$

Substituting $\delta = \frac{1}{169}$ completes the proof. □

3.8.4 The prize-collecting Steiner forest problem (PCSF)

For completeness, we first review the relevant notation and the definition of the PCSF problem. Let $G = (V, E)$ be an undirected, weighted graph with edge costs $c : E \to \mathbb{R}_0^+$ and node prizes $\pi : V \to \mathbb{R}_0^+$. For a subset of edges $E' \subseteq E$, we write $c(E') = \sum_{e \in E'} c(e)$ and adopt the same convention for node subsets. Moreover, for a node subset $V' \subseteq V$, let $\overline{V'}$ be the complement $\overline{V'} = V \setminus V'$. We denote the number of connected components in the (sub-)graph F with $\gamma(F)$.

Definition 3.4 (The prize-collecting Steiner forest problem)**.** *Let $g \in \mathbb{N}$ be the target number of connected components. Then the goal of the prize-collecting Steiner forest (PCSF) problem is to find a subgraph $F = (V', E')$ with $\gamma(F) = g$ that minimizes $c(E') + \pi(\overline{V'})$.*

We divide our analysis in two parts: we first modify the Goemans-Williamson (GW) scheme to get an efficient algorithm with provable approximation guarantee for the PCSF problem (Subsection 3.8.4.1). Then we show how to simulate the GW scheme in nearly-linear time (Subsection 3.8.5).

3.8.4.1 The Goemans-Williamson (GW) scheme for PCSF

Before we introduce our variant of the GW scheme and prove the desired approximation guarantee, we introduce additional notation. For a set of nodes $U \subseteq V$ and a set of edges $D \subseteq E$, we write $\delta_D U$ to denote the set of edges contained in D with exactly one endpoint in U. If $D = E$, we write δU. The degree of a node v in an edge set D is $\deg_D(v) = |\delta_D\{v\}|$. We say that a (sub-)graph F is a g-forest if F is a forest with $\gamma(F) = g$.

At its core, the GW algorithm produces three results: a *laminar family* of clusters, a *dual value* for each cluster, and a forest connecting the nodes within each cluster.

Definition 3.19 (Laminar family)**.** *A family $\mathscr{L}$ of non-empty subsets of V is a* laminar *family if one of the following three cases holds for all $L_1, L_2 \in \mathscr{L}$: either $L_1 \cap L_2 = \{\}$, or $L_1 \subseteq L_2$, or $L_2 \subseteq L_1$.*

Let U be a subset of V. Then we define the following two subsets of $\mathscr{L}$:

- *$\mathscr{L}_{\downarrow U} := \{L \in \mathscr{L} \,|\, L \subseteq U\}$ (going "down" in the laminar hierarchy).*
- *$\mathscr{L}_{\uparrow U} := \{L \in \mathscr{L} \,|\, U \subseteq L\}$ (going "up" in the laminar hierarchy).*

Let $\mathscr{L}^*$ be the family of maximal sets in $\mathscr{L}$, i.e., $L \in \mathscr{L}^*$ iff there is no $L' \in \mathscr{L}$ with $L \subsetneq L'$. If $\bigcup_{L \in \mathscr{L}} = V$, then $\mathscr{L}^*$ is a partition of V.

Let $e \in E$, then we write $\mathscr{L}(e) := \{L \in \mathscr{L} \,|\, e \in \delta L\}$ for the sub-family of sets that contain exactly one endpoint of e.

Definition 3.20 (Dual values)**.** *Let $\mathscr{L}$ be a laminar family. Then the dual values are a function $y : \mathscr{L} \to \mathbb{R}_0^+$ with the following two properties (as before, we write $y(\mathscr{L}') := \sum_{L \in \mathscr{L}'} y(L)$ for a sub-family $\mathscr{L}' \subseteq \mathscr{L}$).*

- *$y(\mathscr{L}(e)) \leq c(e)$ for each $e \in E$.*
- *$y(\mathscr{L}_{\downarrow L}) \leq \pi(L)$ for each $L \in \mathscr{L}$.*

We also define several properties of g-forests related to the new concepts introduced above.

Let $\mathscr{L}$ be a laminar family. We say a tree T is $\mathscr{L}$-connected iff for every $L \in \mathscr{L}$, the subgraph on $V_T \cap L$ is connected (we consider an empty graph to be connected). A g-forest F is $\mathscr{L}$-connected iff every $T \in F$ is $\mathscr{L}$-connected.

Let $L \in \mathscr{L}^*$ and let $L(F)$ be the trees in F with at least one node in L, i.e., $L(F) = \{T \in F \,|\, V_T \cap L \neq \{\}\}$. A g-tree F is $\mathscr{L}^*$-disjoint iff $|L(F)| \leq 1$ for every $L \in \mathscr{L}^*$.

Let $\mathscr{D}$ be a family of subsets of V. A tree T has a leaf component in $\mathscr{D}$ iff there is a $D \in \mathscr{D}$ with $|\delta_T D| = 1$. A g-forest F has a leaf component in $\mathscr{D}$ iff there is a tree $T \in F$ that has a leaf component in $\mathscr{D}$.

A tree T is contained in $\mathscr{D}$ iff there is a $D \in \mathscr{D}$ such that $V_T \subseteq D$. A g-forest F is contained in $\mathscr{D}$ iff there is a tree $T \in F$ that is contained in $\mathscr{D}$.

3.8.4.2 Algorithm

Our algorithm is a modification of the unrooted GW PCST algorithm in [135]. In contrast to their unrooted prize-collecting Steiner tree algorithm, our algorithm stops the growth phase when exactly g active clusters are left. We use these active clusters as starting point in the pruning phase to identify a g-forest as the final result.

Since the focus of this section is the approximation guarantee rather than the time complexity, the pseudo code in Algorithm 11 is intentionally stated at a high level.

3.8.4.3 Analysis

We now show that the forest returned by PCSF-GW has the desired properties: it is a g-forest and satisfies the guarantee in Equation (3.11). Our analysis follows the overall approach of [99].

Lemma 3.21. *Let $H = (V_H, E_H)$ be a graph and let $A, B \subseteq V_H$ be a partition of V_H. Moreover, let $F = \{T_1, \ldots, T_g\}$ be a g-forest such that each T_i has no leaves in B and is not contained in B. Then*

$$\sum_{v \in A} \deg_F(v) + 2|A \setminus V_F| \;\leq\; 2|A| - 2g \; .$$

Proof. Since each T_i has no leaf in B and is not contained in B, every $v \in V_F \cap B$ satisfies $\deg_F(v) \geq 2$. Therefore,

$$\sum_{v \in V_F \cap B} \deg_F(v) \;\geq\; 2|V_F \cap B| \; .$$

Algorithm 11 Prize-collecting Steiner forest

```
function PCSF-GW($G, c, \pi, g$)
    $\mathscr{L} \leftarrow \{\{v\} \mid v \in V\}$                    ▷ Laminar family of clusters.
    $y(C) \leftarrow 0$ for all $C \in \mathscr{L}$.                        ▷ Initial dual values.
    $V_F \leftarrow V, \quad E_F \leftarrow \{\}$                              ▷ Initial forest.
    $\mathscr{D} \leftarrow \{\}$                                ▷ Family of inactive clusters.
    $\mathscr{A} \leftarrow \mathscr{L}^* \setminus \mathscr{D}$                             ▷ Family of active clusters.
    while $|\mathscr{A}| > g$ do                                        ▷ Growth phase.
        $\varepsilon_d \leftarrow \min_{C\in\mathscr{A}} \pi(C) - y(\mathscr{L}_{\downarrow C})$          ▷ Next cluster deactivation time
        $\varepsilon_m \leftarrow \min_{\substack{e\in\delta C\\ C\in\mathscr{A}}} c(e) - y(\mathscr{L}(e))$                ▷ Next cluster merge time
        $\varepsilon \leftarrow \min(\varepsilon_d, \varepsilon_m)$
        for $C \in \mathscr{A}$ do
            $y(C) \leftarrow y(C) + \varepsilon$             ▷ Increase dual variables for active clusters.
        end for
        if $\varepsilon_c \leq \varepsilon_m$ then                         ▷ Cluster deactivation next.
            Let $C \in \mathscr{A}$ be such that $\pi(C) - y(\mathscr{L}_{\downarrow C}) = 0$.
            $\mathscr{D} \leftarrow \mathscr{D} \cup \{C\}$                          ▷ Mark cluster as inactive.
        else                                                    ▷ Cluster merge next
            Let $e$ be such that $c(e) - y(\mathscr{L}(e)) = 0$ and $e \in \delta C$ for some $C \in \mathscr{A}$.
            Let $C_1$ and $C_2$ be the endpoints of $e$ in $\mathscr{L}^*$.
            $\mathscr{L} \leftarrow \mathscr{L} \cup \{C_1 \cup C_2\}$                      ▷ Merge the two clusters.
            $y(C_1 \cup C_2) \leftarrow 0$
            $E_F \leftarrow E_F \cup \{e\}$                              ▷ Add $e$ to the forest.
        end if
        $\mathscr{A} \leftarrow \mathscr{L}^* \setminus \mathscr{D}$                            ▷ Update active clusters.
    end while
    Restrict $F$ to the $g$ trees contained in $\mathscr{A}$. ▷ Discard trees spanning inactive clusters.
    while there is a $D \in \mathscr{D}$ such that $|\delta_F D| = 1$ do              ▷ Pruning phase.
        $V_F \leftarrow V_F \setminus D$                          ▷ Remove leaf component in $\mathscr{D}$.
        Remove all edges from $E_F$ with at least one endpoint in $D$.
    end while
    return $F$
end function
```

Note that $\sum_{v \in V_F} \deg_F(v) = 2(|V_F| - g)$ because F divides V_F into g connected components. Hence

$$\begin{aligned} \sum_{v \in V_F \cap A} \deg_F(v) &= \sum_{v \in V_f} \deg_F(v) - \sum_{v \in V_F \cap B} \deg_F(v) \\ &\le 2(|V_F| - g) - 2|V_F \cap B| \\ &= 2|V_F \cap A| - 2g \ . \end{aligned}$$

Moreover, $|A| = |A \cup V_F| + |A \setminus V_F|$. Combining this with the inequality above gives

$$\begin{aligned} \sum_{v \in V_F \cap A} \deg_F(v) + 2|A \setminus V_F| &\le 2|V_F \cap A| - 2g + 2|A \setminus V_F| \\ &\le 2|A| - 2g \ . \end{aligned}$$

Since $\sum_{v \in A} \deg_F(v) = \sum_{v \in V_F \cap A} \deg_F(v)$, the statement of the lemma follows. □

Lemma 3.22. *Let $\mathscr{L}$ be a laminar family, let $\mathscr{D} \subseteq \mathscr{L}$ be a sub-family, and let $\mathscr{A} = \mathscr{L}^* \setminus \mathscr{D}$. Let F be a g-forest which is $\mathscr{L}$-connected, $\mathscr{L}^*$-disjoint, has no leaf component in $\mathscr{D}$, and is not contained in $\mathscr{D}$. Then*

$$\sum_{C \in \mathscr{A}} |\delta_F C| + 2|\{C \in \mathscr{A} \mid C \in \mathscr{L}_{\downarrow \overline{F}}\}| \ \le \ 2|\mathscr{A}| - 2g \ .$$

Proof. Contract each set $C \in \mathscr{L}^*$ into a single node, keeping only edges with endpoints in distinct sets in $\mathscr{L}^*$. Call the resulting graph H and let A and B be the sets of vertices corresponding to $\mathscr{A}$ and $\mathscr{L}^* \cap \mathscr{D}$, respectively.

Note that F is still a g-forest in H. Since F is $\mathscr{L}^*$-disjoint, no trees in T are connected by the contraction process. Moreover, no cycles are created because F is $\mathscr{L}$-connected. Let F' be the resulting g-forest in H. Since F has no leaf component in $\mathscr{D}$, F' has no leaves in B. Furthermore, no tree in F is contained in $\mathscr{D}$ and thus no tree in F' is contained in B. Therefore, F' satisfies the conditions of Lemma 3.21.

Since F is $\mathscr{L}$-connected, there is a bijection between edges in F' and edges in F with endpoints in distinct elements of $\mathscr{L}^*$. Thus we have

$$\sum_{C \in \mathscr{A}} |\delta_F C| \ = \ \sum_{v \in A} \deg_{F'}(v) \ .$$

Furthermore, the contraction process gives

$$|\{C \in \mathscr{A} \mid C \in \mathscr{L}_{\downarrow \overline{F}}\}| \ = \ |A \setminus V_{F'}|$$

and $|\mathscr{A}| = |A|$. Now the statement of the lemma follows directly from applying Lemma 3.21. □

Lemma 3.23. *At the beginning of every iteration of the growth phase in* PCSF-GW *(lines 7 to 24), the following invariant (I) holds:*

Let F be a g-forest which is $\mathscr{L}$-connected, $\mathscr{L}^*$-disjoint, has no leaf component in $\mathscr{D}$, and is not contained in $\mathscr{D}$. Moreover, let $A = \{v_1, \ldots, v_g\}$ be an arbitrary set of g nodes in G and *let* $\mathscr{B} = \bigcup_{v \in A} \mathscr{L}_{\uparrow\{v\}}$. *Then*

$$\sum_{e \in E_F} y(\mathscr{L}(e)) + 2y(\mathscr{L}_{\downarrow \overline{F}}) \;\leq\; y(\mathscr{L} \setminus \mathscr{B})\;. \tag{3.20}$$

Proof. Clearly, (I) holds at the beginning of the first iteration because the dual values y are 0 for every element in $\mathscr{L}$. We now assume that (I) holds at the beginning of an arbitrary iteration and show that (I) then also holds at the beginning of the next iteration. By induction, this then completes the proof.

Let $\mathscr{L}'$, $\mathscr{D}'$, $\mathscr{A}'$, and y' be the values of $\mathscr{L}$, $\mathscr{D}$, $\mathscr{A}$, and y at the beginning of the iteration. We analyze two separate cases based on the current event in this iteration of the loop: either a cluster is deactivated (lines 15 to 16) or two clusters are merged (lines 18 to 22).

First, we consider the cluster deactivation case. Let F be a g-forest satisfying the conditions of invariant (I). Since $\mathscr{L}' = \mathscr{L}$ and $\mathscr{D}' \subseteq \mathscr{D}$, F is also $\mathscr{L}'$-connected, $\mathscr{L}'^*$-disjoint, has no leaf component in $\mathscr{D}'$, and is not contained in $\mathscr{D}'$. Hence we can invoke Equation (3.20):

$$\sum_{e \in E_F} y'(\mathscr{L}'(e)) + 2y'(\mathscr{L}'_{\downarrow \overline{F}}) \;\leq\; y'(\mathscr{L}' \setminus \mathscr{B})\;. \tag{3.21}$$

Note that y and y' differ only on sets in $\mathscr{A}' = \mathscr{L}'^* \setminus \mathscr{D}'$. Therefore, we have the following three equations quantifying the differences between the three terms in Equations (3.20) and (3.21):

- $$\sum_{e \in E_F} y(\mathscr{L}'(e)) - \sum_{e \in E_F} y'(\mathscr{L}'(e)) \;=\; \sum_{e \in E_F} \sum_{C \in \mathscr{A}'} \varepsilon \cdot \mathbb{1}[e \in \delta_F C] \;=\; \varepsilon \sum_{C \in \mathscr{A}'} |\delta_F C| \tag{3.22}$$

- $$y(\mathscr{L}'_{\downarrow \overline{F}}) - y'(\mathscr{L}'_{\downarrow \overline{F}}) \;=\; \sum_{C \in \mathscr{A}'} \varepsilon \cdot \mathbb{1}[C \in \mathscr{L}'_{\downarrow \overline{F}}] \;=\; \varepsilon |\{C \in \mathscr{A}' \mid C \in \mathscr{L}'_{\downarrow \overline{F}}\}| \tag{3.23}$$

- $$y(\mathscr{L}' \setminus \mathscr{B}) - y'(\mathscr{L}' \setminus \mathscr{B}) \;=\; \sum_{C \in \mathscr{A}'} \varepsilon \cdot \mathbb{1}[C \notin \mathscr{B}] \;=\; \varepsilon |\mathscr{A}'| - \varepsilon |\mathscr{A}' \cap \mathscr{B}| \;\geq\; \varepsilon |\mathscr{A}'| - \varepsilon g \tag{3.24}$$

In the last inequality, we used the fact that $|A| = g$ and hence $\mathscr{B}$ can contain at most g maximal sets in the laminar family $\mathscr{L}'$. Combining the three equations above with Equation (3.21) and Lemma 3.22 then gives:

$$\sum_{e \in E_F} y(\mathscr{L}'(e)) + 2y(\mathscr{L}'_{\downarrow \overline{F}}) \;\leq\; y(\mathscr{L}' \setminus \mathscr{B})\;. \tag{3.25}$$

Since $\mathscr{L}' = \mathscr{L}$, this is equivalent to Equation (3.20), completing the proof for this case.

Now we consider the cluster merge case. As before, let F be a g-forest satisfying the conditions of invariant (I). Since $\mathscr{L} = \mathscr{L}' \cup \{C_1 \cup C_2\}$ and $\mathscr{D} = \mathscr{D}'$, F is also $\mathscr{L}'$-connected, $\mathscr{L}'^*$-disjoint, has no leaf component in $\mathscr{D}'$, and is not contained in $\mathscr{D}'$. Therefore, we can invoke Equation (3.20) again. Moreover, Equations (3.22), (3.23), and (3.24) also hold in this case. Combining these equations with (3.21) and Lemma 3.22 then again results in Equation (3.25). Furthermore, we have $y(C_1 \cup C_2) = 0$ and thus $y(\mathscr{L}(e)) = y(\mathscr{L}'(e)$, $y(\mathscr{L}_{\downarrow \overline{F}}) = y(\mathscr{L}'_{\downarrow \overline{F}})$, and $y(\mathscr{L} \setminus \mathscr{B}) = y(\mathscr{L}' \setminus \mathscr{B})$. Applying these equalities to Equation (3.25) completes the proof. □

The following lemma is essential for proving a lower bound on the value of the optimal solution.

Lemma 3.24. *Let $\mathscr{L}$ be a laminar family with dual values y. Let F be a g-forest and let $\mathscr{B} = \bigcup_{T \in F} \mathscr{L}_{\uparrow T}$. Then*

$$c(E_F) + \pi(\overline{V_F}) \ \geq \ y(\mathscr{L} \setminus \mathscr{B}) \ .$$

Proof. Let $\mathscr{M} = \{C \in \mathscr{L} \mid \delta_F C \neq \{\}\}$ and $\mathscr{N} = \mathscr{L}_{\downarrow \overline{F}}$. Then $\mathscr{L} = \mathscr{M} \cup \mathscr{N} \cup \mathscr{B}$.

Since the y are dual values, we have $c(e) \geq y(\mathscr{L}(e))$ for every $e \in E_F$. Therefore,

$$\begin{aligned} c(E_F) \ &= \ \sum_{e \in E_F} c(e) \ \geq \ \sum_{e \in E_F} y(\mathscr{L}(e)) \ = \ \sum_{e \in E_F} \sum_{C \in \mathscr{L}(e)} y(C) \\ &= \ \sum_{C \in \mathscr{L}} \sum_{e \in \delta_F C} y(C) \ \geq \ \sum_{C \in \mathscr{M}} y(C) \ = \ y(\mathscr{M}) \ . \end{aligned}$$

Moreover, we have $\pi(C) \geq y(\mathscr{L}_{\downarrow C})$ for every $C \in \mathscr{L}$. Thus,

$$\pi(\overline{V_F}) \ \geq \ \sum_{\substack{C \in \mathscr{L}^* \\ C \subseteq \overline{V_F}}} \pi(C) \ \geq \ \sum_{\substack{C \in \mathscr{L}^* \\ C \subseteq \overline{V_F}}} y(\mathscr{L}_{\downarrow C}) \ = \ y(\mathscr{L}_{\downarrow \overline{V_F}}) \ = \ y(\mathscr{N}) \ .$$

Finally, we get

$$c(E_F) + \pi(\overline{V_F}) \ \geq \ y(\mathscr{M}) + y(\mathscr{N}) \ \geq \ y(\mathscr{M} \cup \mathscr{N}) \ = \ y(\mathscr{L} \setminus (\mathscr{L} \setminus (\mathscr{M} \cup \mathscr{N}))) \ \geq \ y(\mathscr{L} \setminus \mathscr{B}) \ ,$$

where we used $\mathscr{L} = \mathscr{M} \cup \mathscr{N} \cup \mathscr{B}$ in the final step. □

We can now prove the main theorem establishing an approximation guarantee for PCSF-GW, which also proves Theorem 3.5.

Theorem 3.25. *Let F be the result of* PCSF-GW(G, c, π, g). *Then F is a g-forest and*

$$c(F) + 2\pi(\overline{F}) \ \leq \ 2c(F_{OPT}) + 2\pi(\overline{F_{OPT}}) \ ,$$

where F_{OPT} is a g-forest minimizing $c(F_{OPT}) + \pi(\overline{F_{OPT}})$.

Proof. By construction in the growth phase of PCSF-GW (lines 7 to 24), F is a $\mathscr{L}$-connected forest at the end of the growth phase. Since at most one element is added to $\mathscr{D}$ in each iteration of the growth phase, we have $|\mathscr{A}| = g$ at the end of the growth phase. Hence restricting F to $\mathscr{A}$ in line 26 leads to F being a g-forest which is still $\mathscr{L}$-connected. Furthermore, F is $\mathscr{L}^*$-disjoint and no tree in F is contained in $\mathscr{D}$.

The pruning phase (lines 27 to 29) maintains that F is a g-forest, $\mathscr{L}$-connected, $\mathscr{L}^*$-disjoint, and not contained in $\mathscr{D}$. Moreover, the pruning phase removes all leaf components of F in $\mathscr{D}$. Hence at the end of the pruning phase, F satisfies the conditions of Lemma 3.23 ($\mathscr{L}$, $\mathscr{D}$, and $\mathscr{A}$ did not change in the pruning phase).

Now let $F_{OPT} = (T_1^{OPT}, \ldots, T_g^{OPT})$ be a g-forest minimizing $c(F_{OPT}) + \pi(\overline{F_{OPT}})$ and let $A = \{v_1, \ldots v_g\}$ with $v_i \in T_i^{OPT}$. Moreover, let $\mathscr{B}_1 = \bigcup_{v \in A} \mathscr{L}_{\uparrow \{v\}}$ as in Lemma 3.23.

Invoking the Lemma then gives

$$\sum_{e \in E_F} y(\mathscr{L}(e)) + 2y(\mathscr{L}_{\downarrow \overline{F}}) \;\leq\; 2y(\mathscr{L} \setminus \mathscr{B}_1) \; . \tag{3.26}$$

Now, note that every $e \in E_F$ was added to F when we had $c(e) = y(\mathscr{L}(e))$. Hence

$$\sum_{e \in E_F} y(\mathscr{L}(e)) \;=\; \sum_{e \in E_F} c(e) \;=\; c(F) \; . \tag{3.27}$$

Moreover, $\overline{V_F}$ can be decomposed into elements in $\mathscr{D}$: after restricting F to $\mathscr{A} = \mathscr{L}^* \setminus \mathscr{D}$ in line 26 this clearly holds. During the pruning phase, all subtrees that are removed from trees in F are elements of $\mathscr{D}$. Therefore, there is a family of pairwise disjoint sets $\mathscr{X} \subseteq \mathscr{D}$ such that $\bigcup_{C \in \mathscr{X}} = \overline{V_F}$. Note that for every $C \in \mathscr{D}$ we have $\pi(C) = y(\mathscr{L}_{\downarrow C})$ because C was deactivated at some point in the growth phase. Therefore,

$$\pi(\overline{F}) \;=\; \sum_{C \in \mathscr{X}} \pi(C) \;=\; \sum_{C \in \mathscr{X}} y(\mathscr{L}_{\downarrow C}) \;\leq\; y(\mathscr{L}_{\downarrow \overline{F}}) \; . \tag{3.28}$$

Combining Equations (3.26), (3.27), and (3.28) then gives

$$c(F) + 2\pi(\overline{F}) \;\leq\; 2y(\mathscr{L} \setminus \mathscr{B}_1) \; . \tag{3.29}$$

We now relate this upper bound to the optimal solution F_{OPT}. Let $\mathscr{B}_2 = \bigcup_{T \in F_{OPT}} \mathscr{L}_{\uparrow T}$ as in Lemma 3.24. The y are valid dual values due to their construction in PCSF-GW. Thus Lemma 3.24 gives

$$y(\mathscr{L} \setminus \mathscr{B}_2) \;\leq\; c(F_{OPT}) + \pi(\overline{F_{OPT}}) \; . \tag{3.30}$$

Note that $\mathscr{B}_2 \subseteq \mathscr{B}_1$ and therefore $y(\mathscr{L} \setminus \mathscr{B}_1) \leq y(\mathscr{L} \setminus \mathscr{B}_2)$. The guarantee in the theorem now follows directly from Equations (3.29) and (3.30). □

3.8.5 A fast algorithm for Goemans-Williamson

We now introduce our fast variant of the GW scheme. To the best of our knowledge, our algorithm is the first practical implementation of a GW-like algorithm that runs in nearly linear time.

On a high level, our algorithm uses a more aggressive and *adaptive* dynamic edge splitting scheme than [76]: our algorithm moves previously inserted splitting points in order to reach a tight edge constraint quicker than before. By analyzing the precision needed to represent merge and deactivation events in the GW algorithm, we prove that our algorithm runs in $O(\alpha \cdot |E| \log|V|)$ time, where α is the number of bits used to specify each value in the input. For constant bit precision α (as is often the case in practical applications) our algorithm hence has a running time of $O(|E| \log|V|)$. Furthermore, our algorithm achieves the approximation guarantee (3.11) exactly without the additional $\frac{2}{n^k}$ term present in the work of [76]. From an empirical point of view, our more aggressive edge splitting scheme produces only very few additional edge pieces: we observed that the number of processed edge events is usually close to $2|E|$, the number of edge events initially created. We demonstrate this empirical benefit in our experiments (see Section 3.8.5.3).

Since the pruning stage of the GW scheme can be implemented relatively easily in linear time [135], we focus on the moat growing stage here. We also remark that there are algorithms for the PCST problem that achieve a nearly-linear time for *planar* graphs [38, 95].

3.8.5.1 Algorithm

Similar to [76], our algorithm divides each edge $e = (u, v)$ into two *edge parts* e_u and e_v corresponding to the endpoints u and v. We say an edge part p is *active* if its endpoint is in an active cluster, otherwise the edge part p is *inactive*. The key advantage of this approach over considering entire edges is that *all active edge parts always grow at the same rate.* For each edge part p, we also maintain an *event value* $\mu(p)$. This event value is the total amount that the moats on edge part p are allowed to grow until the next event for this edge occurs. In order to ensure that the moats growing on the two corresponding edge parts e_u and e_v never overlap, we always set the event values so that $\mu(e_u) + \mu(e_v) = c(e)$. As for edges, we define the remaining slack of edge part e_u as $\mu(e_u) - \sum_{C \in \mathscr{C}} y_C$, where $\mathscr{C}$ is the set of clusters containing node u.

We say that an *edge event* occurs when an edge part has zero slack remaining. However, this does not necessarily mean that the corresponding edge constraint has become tight as the edge event might be "stale" since the other edge parts has become inactive and stopped growing since the last time the edge event was updated. Nevertheless, we will be able to show that the total number of edge events to be processed over the course of the algorithm is small. Note that we can find the next edge event by looking at the edge events with smallest remaining slack values in their clusters. This is an important property because it allows us to organize the edge parts in an efficient manner. In particular, we maintain a priority queue Q_C for each cluster C that contains the edge parts with endpoint in C, sorted by the time at which the next event on each edge part occurs. Furthermore, we arrange the cluster priority queues in an overall priority queue resulting in a "heap of heaps" data structure. This data structure allows us to quickly locate the next edge event and perform the necessary updates after cluster deactivation or merge events.

In addition to the edge events, we also maintain a priority queue of *cluster events*. This priority queue contains each active cluster with the time at which the corresponding cluster constraint becomes tight. Using these definitions, we can now state the high-level structure of our algorithm in pseudo code (see Algorithm 12) and then describe the two subroutines MergeClusters and GenerateNewEdgeEvents in more detail.

MergeClusters(C_u, C_v) : As a first step, we mark C_u and C_v as inactive and remove them from the priority queue keeping track of cluster deactivation events. Furthermore, we remove the priority queues Q_{C_u} and Q_{C_v} from the heap of heaps for edge events. Before we merge the heaps of C_u and C_v, we have to ensure that both heaps contain edge events on the "global" time frame. If C_u (or C_v) is inactive since time t' when the merge occurs, the edge event times in Q_{C_u} will have become "stale" because the moat on edge parts incident to C_u did not grow since t'. In order to correct for this offset and bring the keys in Q_{C_u} back to the global time frame, we first increase all keys in Q_{C_u} by $t - t'$. Then, we merge Q_{C_u} and Q_{C_v}, which results in the heap for the new merged cluster. Finally, we insert the new heap into the heap of heaps and add a new entry to the cluster deactivation heap.

Algorithm 12 Fast variant of the GW algorithm for PCSF.

```
function PCSF-Fast($G, c, \pi, g$)
    InitPCST($G, c, \pi$)
    $t \leftarrow 0$                                        ▷ Current time
    $g' \leftarrow |V|$                                     ▷ Number of active clusters
    while $g' > g$ do
        ▷ Returns event time and corresponding edge part
        $(t_e, p_u) \leftarrow$ GetNextEdgeEvent()
        ▷ Returns event time and corresponding cluster
        $(t_c, C) \leftarrow$ GetNextClusterEvent()
        if $t_e < t_c$ then
            $t \leftarrow t_e$
            RemoveNextEdgeEvent()
            $p_v \leftarrow$ GetOtherEdgePart($p_u$)
            ▷ GetSumOnEdgePart returns the current moat sum on the edge part
            ▷ $p_u$ and the maximal cluster containing $u$
            $(s, C_u) \leftarrow$ GetSumOnEdgePart($p_u$)
            $(s', C_v) \leftarrow$ GetSumOnEdgePart($p_v$)
            $r \leftarrow$ GetEdgeCost($p_u$) $- s - s'$           ▷ Remaining amount on the edge
            if $C_u = C_v$ then          ▷ The two endpoints are already in the same cluster
                continue                               ▷ Skip to beginning of while-loop
            end if
            if $r = 0$ then
                MergeClusters($C_u, C_v$)
            else
                GenerateNewEdgeEvents($p_u, p_v$)
            end if
        else
            $t \leftarrow t_c$
            RemoveNextClusterEvent()
            DeactivateCluster($C$)
            $g' \leftarrow g' - 1$
        end if
    end while
    Pruning()
end function
```

GenerateNewEdgeEvents(p_u, p_v) : This function is invoked when an edge event occurs, but the corresponding edge constraint is not yet tight. Since the edge part p_u has no slack left, this means that there is slack remaining on p_v. Let $\mathscr{C}_u$ and $\mathscr{C}_v$ be the set of clusters containing u and v, respectively. Then $r = c(e) - \sum_{C \in \mathscr{C}_u \cup \mathscr{C}_v} y_C$ is the length of the part of edge e not covered by moats yet. We distinguish two cases:

1. The cluster containing the endpoint v is active.
   Since both endpoints are active, we expect both edge parts to grow at the same rate until they meet and the edge constraint becomes tight. Therefore, we set the new event values to $\mu(p_u) = \sum_{C \in \mathscr{C}_u} + \frac{r}{2}$ and $\mu(p_v) = \sum_{C \in \mathscr{C}_v} + \frac{r}{2}$. Note that this maintains the invariant $\mu(p_u) + \mu(p_v) = c(e)$. Using the new event values for p_u and p_v, we update the priority queues Q_{C_u} and Q_{C_v} accordingly and then also update the heap of heaps.

2. The cluster containing the endpoint v is inactive.
   In this case, we assume that v stays inactive until the moat growing on edge part p_u makes the edge constraint for e tight. Hence, we set the new event values to $\mu(p_u) = \sum_{C \in \mathscr{C}_u} + r$ and $\mu(p_v) = \sum_{C \in \mathscr{C}_v}$. As in the previous case, this maintains the invariant $\mu(p_u) + \mu(p_v) = c(e)$ and we update the relevant heaps accordingly. It is worth noting our setting of $\mu(p_v)$ reduces the slack for p_v to zero. This ensures that as soon as the cluster C_v becomes active again, the edge event for p_v will be processed next.

Crucially, in GenerateNewEdgeEvents, we set the new event values for p_u and p_v so that the next edge event on e would merge the clusters C_u and C_v, *assuming both clusters maintain their current activity status.* If one of the two clusters changes its activity status, this will not hold:

1. If both clusters were active and cluster C_u has become inactive since then, the next event on edge e will be part p_v reaching the common midpoint. However, due to the deactivation of C_u, the edge part p_u will not have reached the common midpoint yet.

2. If C_v was inactive and becomes active before the edge event for p_u occurs, the edge event for p_v will also immediately occur after the activation for C_v. At this time, the moat on p_u has not reached the new, size-0 moat of C_v, and thus the edge constraint is not tight.

However, in the next section we show that if all input values are specified with d bits of precision then at most $O(d)$ edge events can occur per edge. Moreover, even in the general case our experiments in Section 3.6 show that the pathological cases described above occur very rarely in practice. In most instances, only two edge events are processed per edge on average.

3.8.5.2 Analysis

We now study the theoretical properties of our algorithm PCSF-Fast. Note that by construction, the result of our algorithm exactly matches the output of PCSF-GW and hence also satisfies guarantee (3.11).

First, we establish the following structural result for the growth stage of the GW algorithm (the “exact” algorithm PCSF-GW, not yet PCSF-Fast). Informally, we show that a single additional bit of precision suffices to exactly represent all important events in the moat growth process. The following result is equivalent to Theorem 3.6.

Theorem 3.26. *Let all node prizes $\pi(v)$ and edge costs $c(e)$ be even integers. Then all cluster merge and deactivation events occur at integer times.*

Proof. We prove the theorem by induction over the cluster merge and deactivation events occuring in the GW scheme, sorted by the time at which the events happen. We will show that the updates caused by every event maintain the following invariant:

Induction hypothesis Based on the current state of the algorithm, let t_e be the time at which the edge constraint for edge e becomes tight and t_C be the time at which the cluster constraint for cluster C becomes tight. Then t_e and t_C are integers. Moreover, if the merge event at t_e is a merge event between an active cluster and an inactive cluster C, then $t_e - t_{\text{inactive}(C)}$ is even, where $t_{\text{inactive}(C)}$ is the time at which cluster C became inactive.

Clearly, the induction hypothesis holds at the beginning of the algorithm: all edge costs are even, so $t_e = \frac{c(e)}{2}$ is an integer. Since the node prizes are integers, so are the t_C. The assumption on merge events with inactive clusters trivially holds because there are no inactive clusters at the beginning of the algorithm. Next, we perform the induction step by a case analysis over the possible events:

- **Active-active**: a merge event between two active clusters. Since this event modifies no edge events, we only have to consider the new deactivation event for the new cluster C. By the induction hypothesis, all events so far have occured at integer times, so all moats have integer size. Since the sum of prizes in C is also an integer, the new cluster constraint becomes tight at an integer time.

- **Active-inactive**: a merge event between an active cluster and an inactive cluster. Let e be the current edge, t_e be the current time, and C be the inactive cluster. The deactivation time for the new cluster is the same as that of the current active cluster, so it is also integer. Since every edge e' incident to C now has a new growing moat, we have to consider the change in the event time for e'. We denote the previous event time of e' with $t'_{e'}$. We distinguish two cases:

  - If the other endpoint of e' is in an active cluster, the part of e' remaining has size $t'_{e'} - t_e$ and e' becomes tight at time $t_e + \frac{t'_{e'} - t_e}{2}$ because e' has two growing moats. We have

    $$t'_{e'} - t_e \;=\; (t'_{e'} - t_{\text{inactive}(C)}) - (t_e - t_{\text{inactive}(C)}) \, .$$

    Note that both terms on the right hand side are even by the induction hypothesis, and therefore their difference is also even. Hence the new event time for edge e' is an integer.

  - If the other endpoint of e' is an inactive cluster, say C', we have to show that $t_{e'} - t_{\text{inactive}(C')}$ is even, where $t_{e'}$ is the new edge event time for e'. We consider whether C or C' became inactive last:

  - C became inactive last: from the time at which C became inactive we know that $t'_{e'} - t_{\text{inactive}(C')}$ is even. Moreover, we have that $t_{e'} = t'_{e'} + (t_e - t_{\text{inactive}(C)})$. Since $t_e - t_{\text{inactive}(C)}$ is even by the induction hypothesis, so is $t_{e'} - t_{\text{inactive}(C')}$.

  - C' became inactive last: from the time at which C' became inactive we know that $t'_{e'} - t_{\text{inactive}(C)}$ is even. The time of the new edge event can be written as $t_{e'} = t_e + t'_{e'} - t_{\text{inactive}(C')}$ (an integer by the induction hypothesis), which is equivalent to $t_{e'} - t'_{e'} = t_e - t_{\text{inactive}(C')}$. We now use this equality in the second line of the following derivation:

$$\begin{aligned} t_{e'} - t_{\text{inactive}(C')} &= t_{e'} - t'_{e'} + t'_{e'} - t_e + t_e - t_{\text{inactive}(C')} \\ &= 2(t_{e'} - t'_{e'}) + t'_{e'} - t_e \\ &= 2(t_{e'} - t'_{e'}) + (t'_{e'} - t_{\text{inactive}(C)}) - (t_e - t_{\text{inactive}(C)}) \,. \end{aligned}$$

    Since $t_e - t_{\text{inactive}(C)}$ is even by the induction hypothesis, all three terms on the right hand side are even.

- **Cluster deactivation**: Clearly, a deactivation of cluster C leads to no changes in other cluster deactivation times. Moreover, edges incident to C and another inactive cluster will never become tight based on the current state of the algorithm. The only quantities remaining are the edge event times for edges e with another cluster endpoint that is active. Note that up to time t_C, the edge e had two growing moats and t_e was an integer. Therefore, the part of e remaining has length $2(t_e - t_C)$, which is an even integer. The new value of t_e is $t_C + 2(t_e - t_C)$, and since $t_{\text{inactive}(C)} = t_C$ the induction hypothesis is restored.

Since the induction hypothesis is maintained throughout the algorithm and implies the statement of the theorem, the proof is complete. □

We now use this result to show that the number of edge part events occuring in PCSF-Fast is small.

Corollary 3.27. *Let all node prizes $\pi(v)$ and edge costs $c(e)$ be specified with α bits of precision. Then the number of edge part events processed in* PCSF-Fast *is bounded by $O(\alpha \cdot |E|)$.*

Proof. We look at each edge e individually. For every edge part event A on e that does not merge two clusters, the following holds: either A reduces the remaining slack of e by at least a factor of two or the event directly preceeding A reduced the remaining slack on e by at least a factor of two. In the second case, we charge A to the predecessor event of A.

So after $O(\alpha)$ edge parts events on e, the remaining slack on e is at most $\frac{c(e)}{2^\alpha}$. Theorem 3.26 implies that the minimum time between two cluster merge or deactivation events is $\frac{c(e)}{2^{\alpha+1}}$. So after a constant number of additional edge part events on e, the edge constraint of e must be the next constraint to become tight, which is the last edge part event on e to be processed. Therefore, the total number of edge part events on e is $O(\alpha)$. □

We now show that all subroutines in PCSF-Fast can be implemented in $O(\log|V|)$ amortized time, which leads to our final bound on the running time.

Theorem 3.28. *Let all node prizes $\pi(v)$ and edge costs $c(e)$ be specified with α bits of precision. Then* PCSF-Fast *runs in $O(\alpha \cdot |E| \log|V|)$ time.*

Proof. The requirements for the priority queue maintaining edge parts events are the standard operations of a mergeable heap data structure, combined with an operation that adds a constant offset to all elements in a heap in $O(\log|V|)$ amortized time. We can build such a data structure by augmenting a pairing heap with an offset value at each node. Due to space constraints, we omit the details of this construction here. For the outer heap in the heap of heaps and the priority queue containing cluster deactivation events, a standard binomial heap suffices.

We represent the laminar family of clusters in a tree structure: each cluster C is a node, the child nodes are the two clusters that were merged to form C, and the parent is the cluster C was merged into. The initial clusters, i.e., the individual nodes, form the leaves of the tree. By also storing the moat values at each node, the GetSumOnEdgePart operation for edge part p_u can be implemented by traversing the tree from leaf u to the root of its subtree. However, the depth of this tree can be up to $\Omega(|V|)$. In order to speed up the data structure, we use path compression in essentially the same way as standard union-find data structures. The resulting amortized running time for GetSumOnEdgePart and merging clusters then becomes $O(\log|V|)$ via a standard analysis of union-find data structures (with path compression only).

This shows that all subroutines in PCSF-Fast (Algorithm 12) can be implemented to run in $O(\log|V|)$ amortized time. Since there are at most $O(\alpha|E|)$ events to be processed in total, the overall running time bound of $O(\alpha \cdot |E| \log|V|)$ follows. □

3.8.5.3 Experimental results

We also investigate the performance of our algorithm PCSF-Fast outside sparse recovery. As test data, we use the public instances of the DIMACS challenge on Steiner tree problems[1]. We record both the total running times and the number of edge events processed by our algorithm. All experiments were conducted on a laptop computer from 2010 (Intel Core i7 with 2.66 GHz, 4 MB of cache, and 8 GB of RAM). All reported running times are averaged over 11 trials after removing the slowest run. Since the GW scheme has a provable approximation guarantee, we focus on the running time results here.

Running times Figure 3-6 shows the running times of our algorithm on the public DIMACS instances for the unrooted prize-collecting Steiner tree problem (PCSPG). For a single instance, the maximum running time of our algorithm is roughly 1.3 seconds and most instances are solved significantly faster. The scatter plots also demonstrates the nearly-linear scaling of our running time with respect to the input size.

Effectiveness of our edge splitting heuristic As pointed out in our running time analysis, the number of edge part events determines the overall running time of our algorithm. For input values specified with α bits of precision, our analysis shows that the algorithm encounters at most $O(\alpha)$ events per edge. In order to get a better understanding of our

[1] `http://dimacs11.cs.princeton.edu/`

empirical performance, we now look at the number of edge part events encountered by our algorithm (see Figure 3-7). The scatter plots show that the average number of events per edge is less than 3 for all instances. These results demonstrate the effectiveness of our more adaptive edge splitting heuristics. Moreover, the number of edge events encountered explains the small running times on the large i640 instances in Figure 3-6.

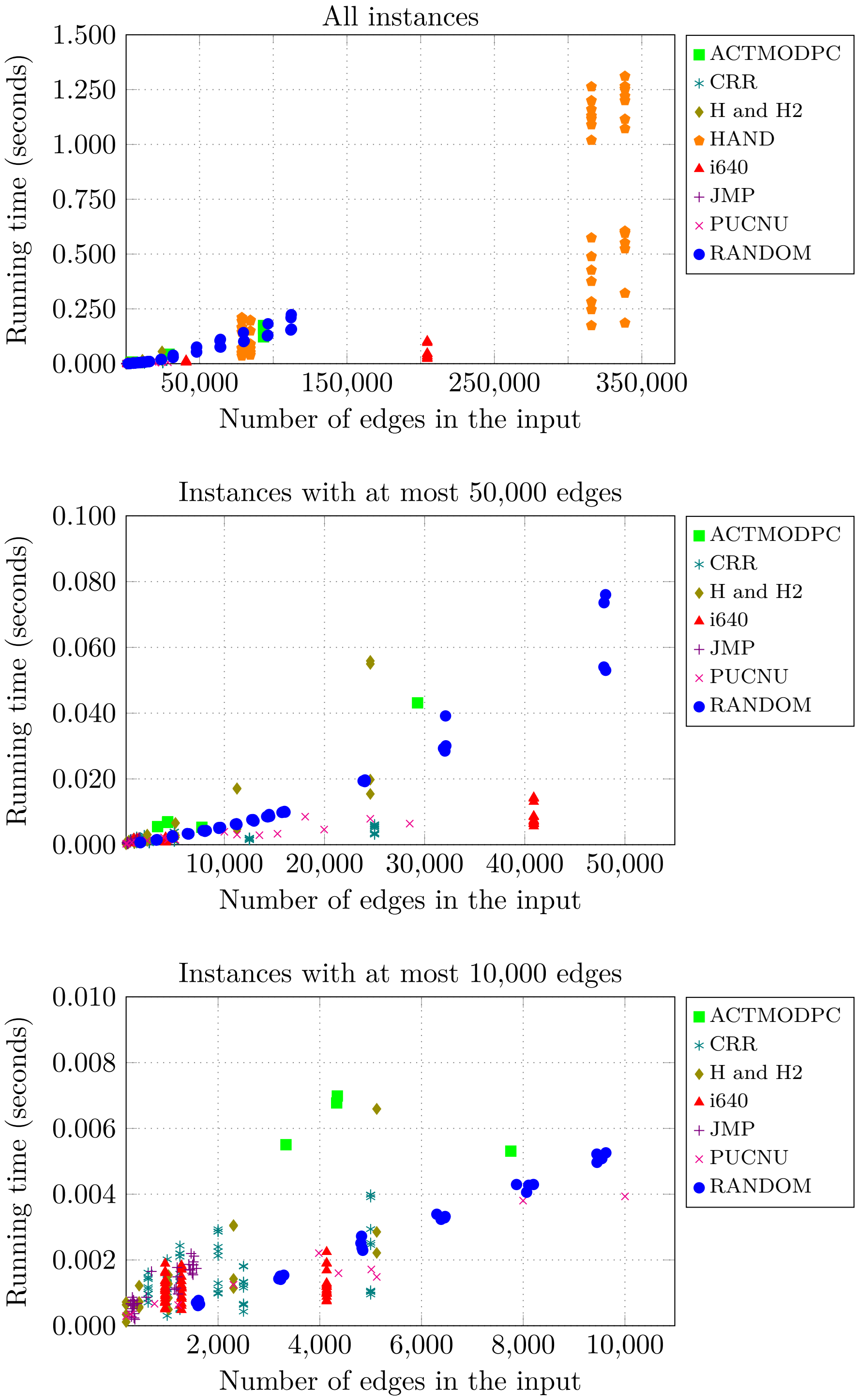


Figure (3-6): Running times for the PCSPG instances of the DIMACS challenge. Each color corresponds to one test case group. Our algorithm runs for at most 1.3s on any instance and clearly shows nearly-linear scaling with the input size.

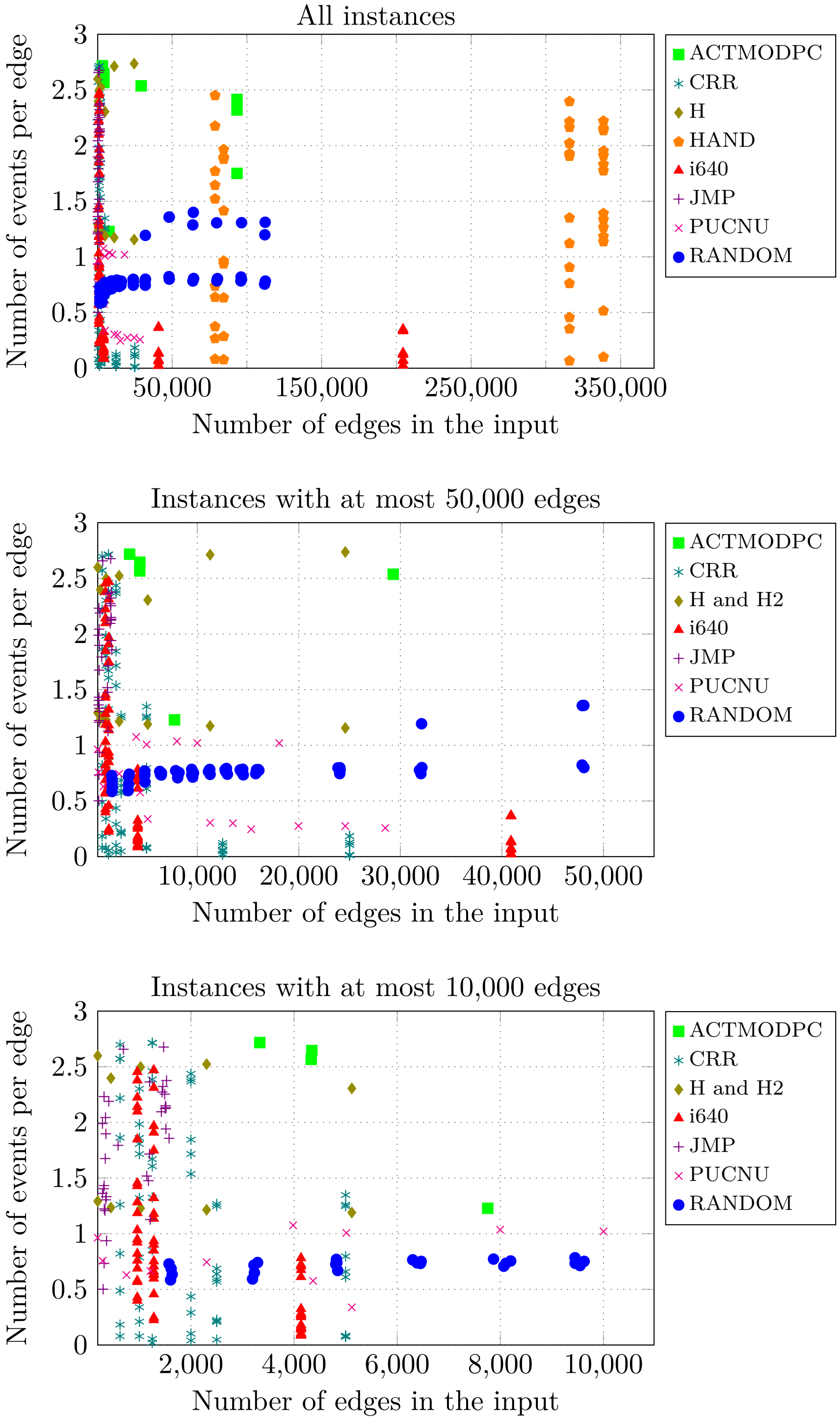


Figure (3-7): Average number of edge events processed per edge for the PCSPG instances of the DIMACS challenge. Each color corresponds to one test case group. The results demonstrate the effectiveness of our edge splitting approach and show that the average number of edge events is less than 3 for every instance.

Chapter 4

Group Sparsity

4.1 Introduction

In many regression problems, we posses prior information that certain features tend to be relevant or irrelevant at the same time. For instance, a genomics application may come with side information that certain genes belong to the same functional groups. If we select one gene from such a functional group, we then should also treat all other genes in this group as selected. *Group sparsity* allows us to incorporate this type of information in statistical learning problems [242]. In a nutshell, group sparsity requires that the selected support (i.e., the set of nonzero indices) can be written as the union of a few pre-defined groups. As we have seen with graph sparsity in the previous chapter, utilizing group sparsity can also lead to better sample complexity and to more interpretable results.

On the algorithmic side, there are two approaches to incorporating group constraints into estimation problems.

- On the convex relaxation side, researchers have proposed so-called *group norms*. One common example is the ℓ_1 / ℓ_2 group norm, which sums the ℓ_2-norms of the variables in each group. Hence this can be seen as an ℓ_1-norm on the group level, and an ℓ_2-norm within each group.
- For non-convex PGD approaches, we require an efficient projection operator onto the set of group sparse vectors.

One special case of group sparsity is its *non-overlapping* form. Here, every feature belongs to exactly one group, i.e., the group structure forms a partition of the coordinates. In this case, it is known that group norms have good statistical properties, and that efficient projection operators exist. This is easy to see for the projection approach: when the groups do not overlap, projecting onto the group sparse set corresponds to a standard knapsack problem.

However, non-overlapping group sparsity is somewhat restrictive. Ideally, we would be able to handle arbitrary group structures. Unfortunately, group sparsity turns out to be significantly more challenging in the general case. Using the aforementioned group norm approach leads to undesirable properties because the regularizer then encourages that the *complement* of the selected support has a group-sparse representation [232]. Moreover, projecting onto the set of vectors satisfying an overlapping group sparsity constraint is an NP-hard problem [33].

We circumvent these issues by introducing new *approximate* projections for the set of group sparse vectors. Our algorithms make no assumptions about the group structure and also work in the overlapping case. The head approximation turns out to be a standard problem in submodular maximization. Our main contribution here is to show that the submodular greedy algorithm can be implemented in *nearly-linear time* in this case. In particular, we efficiently implement the oracle calls of the greedy algorithm with a careful use of priority queues.

Our tail approximation algorithm is more technical and uses ideas from approximation algorithms for the set cover problem.[1] At a high level, our algorithm proceeds via a binary search over a Lagrangian relaxation of the non-convex sparsity constraint. For a each value of the trade-off parameter, we again use a greedy-like algorithm to solve the sub-problems in nearly-linear time. Here, we have to take special care to avoid a dependence on the dynamic range of the input values. We achieve this by “clamping” the input values for the greedy sub-routine appropriately.

4.1.1 Related work

Due to the large body of work on group sparsity, we only mention the most closely related publications here.

Jacob, Obozinski, and Vert [130] introduce a convex regularizer for overlapping group structures. This approach avoids some of the issues that arise when applying the group norms from the non-overlapping case. But to the best of our knowledge, the statistical properties of their regularizer are not yet fully understood. In [130], the authors give only asymptotic consistency results. One approach could be via the decomposable regularizer framework of Negahban et al. [171], which is one of the most general schemes for analyzing convex relaxations in statistical estimation tasks. However, the authors of [171] point out that their current analysis does not extend to the overlapping group regularizer of [130]. In contast, the statistical analysis of the PGD approach requires only a simple counting argument similar to model-based compressive sensing [36].

Baldassarre et al. [33] show that projecting onto a group-sparse set of vectors is NP-hard in the general case. Moreover, the authors propose a type of group structure that allows for a limited amount of overlap between the groups. For this type of group sparsity, the authors provide a polynomial time approach to constrained estimation problems. Our approach using approximate projections is more general and applies to arbitrary group structures. Moreover, our algorithms run in nearly-linear time, while their dynamic programming-based method requires (strongly super-linear) polynomial time.

Finally, Jain, Rao, and Dhillon [132] also introduce a head approximation for the set of group sparse vectors based on the submodular greedy algorithm. However, they do not utilize a tail approximation in their recovery algorithm, which leads to sub-optimal sample complexity bounds. Moreover, the authors do not provide a nearly-linear time head approximation. We

[1] Note that the hardness of projecting onto the group sparse set also stems from a connection with the set cover problem.

note that their greedy approach can likely be converted to a nearly-linear time algorithm as well by using priority queues as in our head approximation.

4.1.2 Preliminaries

We now formally define the notion of group sparsity we consider in this chapter.

We assume we are given a family of groups $\mathbb{G} = \{G_1, \ldots, G_{N_g}\}$ where each G_i contains a subset of the coordinates, i.e., $G_i \subseteq [d]$. Let m be the maximum size of a group:

$$m = \max_{G \in \mathbb{G}} |G| \; .$$

We are also given a target number of groups g. We denote subsets of $\mathbb{G}$ with calligraphic letters such as $\mathcal{G}$.

Subsets of $\mathbb{G}$ with a given cardinality will be particularly important for us. We write $\mathbb{G}_g$ for subsets of size at most g:

$$\mathbb{G}_g = \{\mathcal{G} \subseteq \mathbb{G} \mid |\mathcal{G}| \leq g\}.$$

Using this notation, we can now define the constraint set of group sparse vectors that we wish to project on:

$$\mathcal{C}_{\mathbb{G},g} \;=\; \left\{ \theta \in \mathbb{R}^d \;\middle|\; \operatorname{supp}(\theta) \subseteq \bigcup_{G \in \mathcal{G}} G \text{ for some } \mathcal{G} \in \mathbb{G}_g \right\} \; .$$

Before we proceed to our approximate projections, we introduce additional notation that will be useful. For a vector $b \in \mathbb{R}^d$ and a subset $\mathcal{G} \subseteq \mathbb{G}$, we write $b_{\mathcal{G}}$ for the restriction of b to elements contained in at least one of the $G \in \mathcal{G}$, i.e.,

$$(b_{\mathcal{G}})_i \;=\; \begin{cases} b_i & \text{if there is a } G \in \mathcal{G} \text{ such that } i \in G \\ 0 & \text{otherwise} \end{cases} \; .$$

Similarly, we sometimes use $\mathcal{G}$ as a shorthand for the union of its members $\bigcup_{G \in \mathcal{G}} G$, which will be clear from context.

4.2 Head approximation for group sparsity

For a given input $b \in \mathbb{R}^d$, the goal of head approximation is to find a set of groups $\mathcal{G}$ such that

$$\|b_{\mathcal{G}}\|_2^2 \;\geq\; c_{\mathcal{H}} \cdot \max_{\mathcal{G}' \in \mathbb{G}_g} \|b_{\mathcal{G}'}\|_2^2$$

for constant $0 < c_{\mathcal{H}} \leq 1$. To achieve a good sample complexity, it is critical to use a small number of groups $\mathcal{G}$. Ideally, $|\mathcal{G}|$ should be $O(g)$.

We can achieve such an approximation by observing that the above is a weighted coverage

Algorithm 13 Greedy algorithm for submodular maximization

```
function GreedyHead($b, h$)
    $\mathcal{G} \leftarrow \{\}$
    while $|\mathcal{G}| \leq h$ do
        $G \leftarrow \max_{G' \in \mathbb{G}} \|b_{G'}\|_2^2$
        $\mathcal{G} \leftarrow \mathcal{G} \cup \{G\}$
        $b_G \leftarrow 0$
    end while
    return $\mathcal{G}$
end function
```

maximization problem. In particular, if we define $h : 2^{\mathbb{G}} \to \mathbb{R}^+$ such that $h(\{\}) = 0$ and

$$h(\mathcal{G}) = \|b_{\bigcup_{G \in \mathcal{G}} G}\|^2,$$

then h is monotone-submodular. Hence the standard greedy algorithm can achieve a constant-factor head approximation. For completeness, we describe the procedure in Algorithm 13.

We now analyze this algorithm. Recall the following (classical) result [143]:

Lemma 4.1. *Let $\mathcal{G} = \{G_i\}_{i=1,\ldots,h}$ be the greedily selected groups by Algorithm 13. Then, we have:*

$$\|b_{\mathcal{G}}\|_2^2 \;\geq\; \left(1 - e^{-h/g}\right) \max_{\mathcal{G}' \in \mathbb{G}_g} \|b_{\mathcal{G}'}\|_2^2.$$

In particular, we get a constant-factor approximation for $h = O(g)$.

When naively implemented, the running time of the above algorithm is by $O(hN_g)$. However, we can speed up the algorithm if we maintain the ℓ_2 norms of the N_g groups as a heap. Constructing this heap in the beginning requires time that is nearly-linear in the size of the set-system describing $\mathbb{G}$. Each update to the heap touches an coordinate at most once (since b_G is zeroed out once G is added to $\mathcal{G}$). Therefore, the total cost of all the updates is nearly-linear in $O(N_g \max |G_i|)$.

4.3 Tail approximation for group sparsity

Overall, our goal in this section is to design an efficient tail approximation algorithm for group sparsity. We aim to find a set of groups $\mathcal{G}$ such that

$$\|b - b_{\mathcal{G}}\|_2^2 \;\leq\; c_{\mathcal{T}} \cdot \min_{\mathcal{G}' \in \mathbb{G}_g} \|b - b_{\mathcal{G}'}\|_2^2$$

where $c_{\mathcal{T}}$ is a fixed constant. The challenging part is to use few groups. Ideally (for sample-complexity reasons), we would have $|\mathcal{G}| = O(g)$. However, this is impossible due to approximation hardness results for set cover. Instead, we will give a tail approximation algorithm with $|\mathcal{G}| = O(g \log m)$. The $m = O(\log d)$ regime is already relevant for group sparsity in terms of sample complexity. In this regime, the increase in the number of groups

Algorithm 14 Greedy algorithm for the Lagrangian relaxation

```
function GreedyLagrange($b, \lambda$)
    $\mathcal{G} \leftarrow \{\}$
    while $\mathcal{G} \neq \mathbb{G}$ do
        $G \leftarrow \max_{G' \in \mathbb{G}} \|b_{G'}\|_2^2$
        if $\|b_G\|_2^2 < \lambda$ then
            return $\mathcal{G}$
        end if
        $\mathcal{G} \leftarrow \mathcal{G} \cup \{G\}$
        $b_G \leftarrow 0$
    end while
    return $\mathcal{G}$
end function
```

will only lead to a *doubly*-logarithmic increase in the sample complexity compared to the information-theoretic lower bound of $O(s)$ for an s-sparse vector.

A main ingredient in our approach is the following relaxation of the tail approximation problem. The goal is still to find a set of groups $\mathcal{G}$, but now with a penalty term for the number of groups instead of a given budget. The fixed parameter λ controls the trade-off between approximation quality and the number of groups. Formally, we want to find a set of groups $\mathcal{G} \subseteq \mathbb{G}$ such that

$$\log m \cdot \|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| \;\leq\; \log m \cdot \min_{\mathcal{G}'} \Big(\|b - b_{\mathcal{G}'}\|_2^2 + c\lambda|\mathcal{G}'| \Big) \tag{4.1}$$

where c is a fixed constant.

4.3.1 Lagrangian relaxation

We show that the following greedy algorithm produces a set of groups $\mathcal{G}$ satisfying the guarantee in Equation (4.1).

In this section, we will sometimes "clamp" a vector by restricting all entries to be within $-\tau$ and τ for a given parameter $\tau \in \mathbb{R}$. We denote this operation with $\mathrm{cl}_\tau(b) : \mathbb{R}^d \to \mathbb{R}^d$ and define it element-wise as $\mathrm{cl}_\tau(b)_i = \max(\min(b_i, \tau), -\tau)$. First, we show that running GreedyLagrange on the clamped version of b does not affect its solution quality.

Lemma 4.2. *Let $\lambda \in \mathbb{R}^+$, $\lambda' \leq \lambda$, $b \in \mathbb{R}^d$, $\tilde{b} = cl_{\sqrt{\lambda}}(b)$, and*

$$\mathcal{G} \;=\; \text{GreedyLagrange}(\tilde{b}, \lambda') \; .$$

Then we have

$$\|b - b_{\mathcal{G}}\|_2^2 \;=\; \|\tilde{b} - \tilde{b}_{\mathcal{G}}\|_2^2 \; .$$

Proof. We show that $b - b_{\mathcal{G}} = \tilde{b} - \tilde{b}_{\mathcal{G}}$, which clearly implies the equality in the theorem. For $i \in \mathcal{G}$, we have $(b - b_{\mathcal{G}})_i = 0 = (\tilde{b} - \tilde{b}_{\mathcal{G}})_i$. For $i \in [d] \setminus \mathcal{G}$, we have $b_i^2 < \lambda'$ because otherwise GreedyLagrange would have added a set containing i to $\mathcal{G}$. Hence also $b_i^2 < \lambda$. Due to the clamping threshold $\sqrt{\lambda}$, we have $b_i = \tilde{b}_i$, which implies $(b - b_{\mathcal{G}})_i = (\tilde{b} - \tilde{b}_{\mathcal{G}})_i$. □

Algorithm 15 Greedy algorithm with a fixed number of iterations

```
function GreedyFixed($(b, k)$)
    $\mathcal{G} \leftarrow \{\}$
    for $i \leftarrow 1, \ldots, k$ do
        $G \leftarrow \max_{G' \in \mathbb{G}} \|b_{G'}\|_2^2$
        $\mathcal{G} \leftarrow \mathcal{G} \cup \{G\}$
        $b_G \leftarrow 0$
    end for
    return $\mathcal{G}$
end function
```

Next, we relate the solution quality obtained by GreedyLagrange to a variant of the algorithm with a fixed number of iterations. In particular, we consider GreedyFixed as presented in Algorithm 15. We have the following result.

Lemma 4.3. *Let $k \in \mathbb{N}$, $\gamma \in \mathbb{R}^+$, $b \in \mathbb{R}^d$, $\mathcal{G} = \textsc{GreedyLagrange}(b, \lambda/\gamma)$. Moreover, let $\mathcal{G}' = \textsc{GreedyFixed}(b, k)$. Then we have*

$$\gamma \cdot \|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| \;\leq\; \gamma \cdot \|b - b_{\mathcal{G}'}\|_2^2 + \lambda|\mathcal{G}'| \,.$$

Proof. Note that GreedyFixed and GreedyLagrange consider the groups $G \in \mathbb{G}$ in the same order (in case of a tie in marginal gain, we define the algorithms so that they choose the same group). So if $|\mathcal{G}| = |\mathcal{G}'|$, the inequality in the lemma clearly holds with equality. We now consider two cases.

Case 1: $|\mathcal{G}| < |\mathcal{G}'|$. Let Ω be the indices in $\mathcal{G}'$ but not in $\mathcal{G}$. For each group $G \in \mathcal{G}' \setminus \mathcal{G}$ the marginal gain $\|b_G\|_2^2$ was less than λ/γ because GreedyLagrange did not add the group G to its solution. Hence we have $\gamma\|b_\Omega\|_2^2 < \lambda(|\mathcal{G}'| - |\mathcal{G}|)$. Moreover, we have

$$\gamma\|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| - \gamma\|b - b_{\mathcal{G}'}\|_2^2 - \lambda|\mathcal{G}'| \;=\; \gamma\|b_\Omega\|_2^2 - \lambda(|\mathcal{G}'| - |\mathcal{G}|) \;<\; 0$$

which implies the statement of the lemma.

Case 2: $|\mathcal{G}| > |\mathcal{G}'|$. Let Ω be the indices in $\mathcal{G}$ but not in $\mathcal{G}'$. For each group $G \in \mathcal{G} \setminus \mathcal{G}'$ the marginal gain $\|b_G\|_2^2$ was at least λ/γ because GreedyLagrange added the group G to its solution. Hence we have $\gamma\|b_\Omega\|_2^2 > \lambda(|\mathcal{G}| - |\mathcal{G}'|)$. Moreover, we have

$$\gamma\|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| - \gamma\|b - b_{\mathcal{G}'}\|_2^2 - \lambda|\mathcal{G}'| \;=\; -\gamma\|b_\Omega\|_2^2 + \lambda(|\mathcal{G}| - |\mathcal{G}'|) \;<\; 0$$

which implies the statement of the lemma.

□

We now show that GreedyFixed satisfies the guarantee of the Lagrangian relaxation when run for the right number of iterations. First, we briefly introduce a fact about GreedyFixed. It is an instantiation of the classical guarantee for the submodular greedy algorithm [143].

Fact 4.4. *Let $b \in \mathbb{R}^d$, $\alpha \in \mathbb{N}$, $\beta \in \mathbb{N}$, and $\mathcal{G} = \textsc{GreedyFixed}(b, \alpha \cdot \beta)$. Then we have*

$$\|b_{\mathcal{G}}\|_2^2 \;\geq\; (1 - e^{-\alpha}) \max_{\mathcal{G}' \in \mathbb{G}_\beta} \|b_{\mathcal{G}'}\|_2^2 \,.$$

Lemma 4.5. *Let $\lambda \in \mathbb{R}^+$ and $b \in \mathbb{R}^d$ with $b_{\max}^2 = \max_{i\in[d]} b_i^2 \le \lambda$. Moreover, let $\mathcal{G}^*$ be such that*

$$\|b - b_{\mathcal{G}^*}\|_2^2 + 2\lambda|\mathcal{G}^*| \;=\; \min_{\mathcal{G}'\subseteq\mathbb{G}}\Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'|\Big)\ .$$

Let $\mathcal{G} = \textsc{GreedyFixed}(b, |\mathcal{G}^*|\log m)$ where $m = \max_{G\in\mathbb{G}}|G|$. Then we have

$$\log m \cdot \|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| \;\le\; \log m \cdot \min_{\mathcal{G}'\subseteq\mathbb{G}}\Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'|\Big)\ .$$

Proof. We will show that

$$\log m \cdot \|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| \;\le\; \log m \cdot \Big(\|b - b_{\mathcal{G}^*}\|_2^2 + 2\lambda|\mathcal{G}^*|\Big) \tag{4.2}$$

which implies the guarantee in the theorem. By definition of $\mathcal{G}$ we have

$$\lambda|\mathcal{G}| \;\le\; \log m \cdot \lambda|\mathcal{G}^*|\ .$$

After subtracting this inequality from the Equation (4.2) and cancelling the $\log m$ term on both sides, it suffices to show that

$$\|b - b_{\mathcal{G}}\|_2^2 \;\le\; \|b - b_{\mathcal{G}^*}\|_2^2 + \lambda|\mathcal{G}^*|\ . \tag{4.3}$$

We now invoke Fact 4.4, which yields

$$\begin{aligned}
\|b_{\mathcal{G}}\|_2^2 \;&\ge\; (1 - e^{-\log m}) \max_{\mathcal{G}'\in\mathbb{G}_{|\mathcal{G}^*|}} \|b_{\mathcal{G}'}\|_2^2 \\
&\ge\; \Big(1 - \frac{1}{m}\Big)\|b_{\mathcal{G}^*}\|_2^2 \\
&\ge\; \|b_{\mathcal{G}^*}\|_2^2 - \frac{b_{\max}^2 m|\mathcal{G}^*|}{m} \\
&\ge\; \|b_{\mathcal{G}^*}\|_2^2 - \lambda|\mathcal{G}^*|
\end{aligned}$$

where the third line used that every element in $b_{\mathcal{G}^*}^2$ is bounded by $b_{\max}^2$, and $\mathcal{G}^*$ contains at most $|\mathcal{G}^*|$ groups of size m. The last line follows from the assumption that $b_{max}^2 \le \lambda$. Multiplying both sides with -1 yields

$$-\|b_{\mathcal{G}}\|_2^2 \;\le\; -\|b_{\mathcal{G}^*}\|_2^2 + \lambda|\mathcal{G}^*|\ .$$

Finally, note that $\|b - b_{\mathcal{G}}\|_2^2 = \|b\|_2^2 - \|b_{\mathcal{G}}\|_2^2$ and the same holds for $\mathcal{G}^*$. Hence adding $\|b\|_2^2$ to both sides of the above inequality yields the desired result in Equality (4.3). □

We now prove a final lemma about the effect of clamping on the optimal solution.

Lemma 4.6. *Let $b \in \mathbb{R}^d$, $\gamma \in \mathbb{R}^+$, $\lambda \in \mathbb{R}^+$, and $\tilde{b} = cl_\gamma(b)$. Then we have*

$$\min_{\mathcal{G}'\subseteq\mathbb{G}}\|\tilde{b} - \tilde{b}_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \;\le\; \min_{\mathcal{G}'\subseteq\mathbb{G}}\Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'|\Big)\ .$$

Proof. We consider the inequality for a fixed set of groups $\mathcal{G}' \subset \mathbb{G}$ on both sides. By proving the resulting inequality

$$\|\tilde{b} - \tilde{b}_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \ \leq \ \|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \ .$$

for all $\mathcal{G}'$, we establish the theorem.

Subtracting the term $2\lambda|\mathcal{G}'|$ on both sides leaves the following inequality

$$\|\tilde{b} - \tilde{b}_{\mathcal{G}'}\|_2^2 \ \leq \ \|b - b_{\mathcal{G}'}\|_2^2 \ .$$

For all $i \in \mathcal{G}'$, we have $(\tilde{b} - \tilde{b}_{\mathcal{G}'})_i = 0 = (b - b_{\mathcal{G}'})_i$. For the remaining indices, we have

$$(\tilde{b} - \tilde{b}_{\mathcal{G}'})_i \ = \ \tilde{b}_i \ \leq \ b_i \ = \ (b - b_{\mathcal{G}'})_i$$

which completes the proof. □

We now have all ingredients in place to prove our main result for the Lagrangian relaxation.

Theorem 4.7. *Let* $\lambda \in \mathbb{R}^+$, $b \in \mathbb{R}^d$, $\tilde{b} = cl_{\sqrt{\lambda}}(b)$, *and* $m = \max_{G \in \mathbb{G}} |G|$. *Then the set of groups* $\mathcal{G} = \textsc{GreedyLagrange}(\tilde{b}, \lambda / \log m)$ *satisfies*

$$\log m \cdot \|b - b_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| \ \leq \ \log m \cdot \min_{\mathcal{G}' \subseteq \mathbb{G}} \Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \Big)$$

Proof. Let $\mathcal{G}^*$ be such that

$$\|\tilde{b} - \tilde{b}_{\mathcal{G}^*}\|_2^2 + 2\lambda|\mathcal{G}^*| \ = \ \min_{\mathcal{G}' \subseteq \mathbb{G}} \Big(\|\tilde{b} - \tilde{b}_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \Big) \ .$$

Since we have clamped b to $\tilde{b}$, we can invoke Lemma 4.5 and get a

$$\mathcal{G}_F = \textsc{GreedyFixed}(\tilde{b}, |\mathcal{G}^*| \log m)$$

such that

$$\begin{aligned} \log m \cdot \|\tilde{b} - \tilde{b}_{\mathcal{G}_F}\|_2^2 + \lambda|\mathcal{G}_F| \ &\leq \ \log m \cdot \min_{\mathcal{G}' \subseteq \mathbb{G}} \Big(\|\tilde{b} - \tilde{b}_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \Big) \\ &\leq \ \log m \cdot \min_{\mathcal{G}' \subseteq \mathbb{G}} \Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \Big) \end{aligned}$$

where the second line follows from Lemma 4.6. Invoking Lemma 4.3 on the left hand side (using $\gamma = \log m$) then yields

$$\log m \cdot \|\tilde{b} - \tilde{b}_{\mathcal{G}}\|_2^2 + \lambda|\mathcal{G}| \ \leq \ \log m \cdot \min_{\mathcal{G}' \subseteq \mathbb{G}} \Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \Big) \ .$$

Finally applying Lemma 4.2 on the left hand side (using $\lambda' = \lambda / \log m$) completes the proof. □

4.3.1.1 Tail approximation algorithm

With the sub-routine for the Lagrangian relaxation in place, we can now state our main algorithm (see Algorithm 16).

The subroutine LagrangeApprox is the algorithm developed in the previous section. As a simple corollary of Theorem 4.7, the output $\mathcal{G}_\lambda$ satisfies

$$\begin{aligned}\log m \cdot \|b - b_{\mathcal{G}_\lambda}\|_2^2 + \lambda|\mathcal{G}_\lambda| \;&\leq\; \log m \cdot \min_{\mathcal{G}' \in \mathbb{G}} \Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda|\mathcal{G}'| \Big) \\ &\leq\; \log m \cdot \min_{\mathcal{G}' \in \mathbb{G}_g} \Big(\|b - b_{\mathcal{G}'}\|_2^2 + 2\lambda g \Big) \\ &\leq\; OPT_g \log m + 2\lambda g \log m \qquad (4.4)\end{aligned}$$

where we define $OPT_g = \min_{\mathcal{G}' \in \mathbb{G}_g} \|b - b_{\mathcal{G}'}\|_2^2$. We will use this inequality line repeatedly in the following argument.

We first prove a lemma about the initialization of λ_ℓ.

Lemma 4.8. *Let m, λ_ℓ, g, and $\mathcal{G}_\ell$ be as in Algorithm 16. Then the output satisfies $\|b - b_{\mathcal{G}_\ell}\|_2^2 \;\leq\; (1 + 1/\gamma) OPT_g$. Moreover, if $OPT_g = 0$, then we have $|\mathcal{G}_\ell| \leq 2g \log m$.*

Proof. We start with the assumption $OPT_g = 0$. Equation (4.4) then yields

$$\log m \cdot \|b - b_{\mathcal{G}_\ell}\|_2^2 + \lambda_\ell |\mathcal{G}_\ell| \;\leq\; 2\lambda_\ell g \log m$$

which directly implies $|\mathcal{G}_\ell| \leq 2g \log m$ as desired. Moreover, we have

$$\begin{aligned}\|b - b_{\mathcal{G}_\ell}\|_2^2 \;&\leq\; 2\lambda_\ell g \\ &=\; \frac{b_{\min}^2}{2} \;.\end{aligned}$$

Note that the smallest nonzero tail error is $b_{\min}^2$. Without loss of generality we can assume that there is a i such that $b_i \neq 0$ (otherwise the problem is trivial). Hence $\frac{b_{\min}^2}{2}$ is strictly smaller than $b_{\min}^2$ and we have $\|b - b_{\mathcal{G}_\ell}\|_2^2 = 0$ as required.

The $OPT_g \neq 0$ case remains. We want to prove that $\|b - b_{\mathcal{G}_\ell}\|_2^2 \;\leq\; (1 + 1/\gamma) OPT_g$. Invoking Equation (4.4) and dropping $\lambda_\ell|\mathcal{G}_\ell|$ from the left side gives

$$\log m \cdot \|b - b_{\mathcal{G}_\ell}\|_2^2 \;\leq\; OPT_g \log m + 2\lambda_\ell g \log m$$

or equivalently

$$\begin{aligned}\|b - b_{\mathcal{G}_\ell}\|_2^2 \;&\leq\; OPT_g + 2\lambda_\ell g \\ &=\; OPT_g + \frac{b_{\min}^2}{2\gamma} \\ &\leq\; (1 + 1/\gamma) OPT_g\end{aligned}$$

because $OPT_g \neq 0$ and hence $OPT_g \geq b_{\min}^2$ as before. □

We can now prove our main theorem.

Theorem 4.9. *Let $b \in \mathbb{R}^d$, $g \in \mathbb{N}$, $\gamma \in \mathbb{R}^+$ and $\mathcal{G} = \textsc{GroupTail}(b, g, \gamma)$. Moreover, let $m = \max_{G \in \mathbb{G}} |G|$. Then we have $|\mathcal{G}| \leq (2+\gamma) g \log m$ and*

$$\|b - b_{\mathcal{G}}\|_2^2 \;\leq\; \left(1 + \frac{3}{\gamma}\right) OPT_g \; .$$

Proof. We consider the three lines in which GroupTail can return the result $\mathcal{G}$.

First, consider Line 9. We clearly have $|\mathcal{G}| \leq (2+\gamma) g \log m$. Moreover, Lemma 4.8 implies that $\|b - b_{\mathcal{G}}\|_2^2 \leq (1 + 3/\gamma) OPT_g$ as stated in the theorem.

Next, we consider the remaining two cases. We can assume that $OPT_g \neq 0$ because the algorithm would have returned in Line 9 otherwise (contrapositive of Lemma 4.8). Hence we have $OPT_g \geq b_{\min}^2$.

If the algorithm returns in Line 15, we again have $|\mathcal{G}| \leq (2+\gamma) g \log m$ by definition. Furthermore, Equation (4.4) gives

$$\begin{aligned} \|b - b_{\mathcal{G}}\|_2^2 \log m \;&\leq\; OPT_g \log m + \lambda(2g \log m - |b_{\mathcal{G}}|) \\ &\leq\; OPT_g \log m \end{aligned}$$

because we also have $|b_{\mathcal{G}}| \geq 2g \log m$. Hence $\|b - b_{\mathcal{G}}\|_2^2$ satisfies the guarantee in the theorem.

Line 30 remains. To simplify notation, we use $g_\ell = |\mathcal{G}_\ell|$ and $t_\ell = \|b - b_{\mathcal{G}_\ell}\|_2^2$. We adopt the same notation for g_r and t_r. Note that binary search maintains the invariant that $g_r \leq 2g \log m$ and $g_\ell > (2+\gamma) g \log m$. So we have $|\mathcal{G}| \leq (2+\gamma) g \log m$ as desired and it remains to show that $t_r \leq (1 + 3/\gamma) OPT_g$ in this case.

Instantiating Equation (4.4) for λ_ℓ yields

$$t_\ell \log m + \lambda_\ell g_\ell \;\leq\; OPT_g \log m + \lambda_\ell g \log m \; .$$

Using $g_\ell > (2+\gamma) g \log m$ and simplifying gives

$$\begin{aligned} t_\ell \log m + \lambda_\ell (2+\gamma) g \log m \;&\leq\; OPT_g \log m + \lambda_\ell g \log m \\ t_\ell + \lambda_\ell \gamma g \;&\leq\; OPT_g \\ \lambda_\ell \;&\leq\; \frac{OPT_g}{\gamma g} \; . \end{aligned}$$

At the end of the binary search we also have $\lambda_r \leq \lambda_\ell + \varepsilon$. We use this to bound t_r as follows. Instantiating Equation (4.4) for λ_r and dividing both sides by $\log m$ gives

$$\begin{aligned} t_r \;&\leq\; OPT_g + 2\lambda_r g \\ &\leq\; OPT_g + 2\lambda_l g + 2g\varepsilon \\ &\leq\; OPT_g + \frac{2}{\gamma} OPT_g + \frac{b_{\min}^2}{\gamma} \; . \end{aligned}$$

In the last line we used the bound on λ_l and the definition of ε. Since $b_{\min}^2 \leq OPT_g$, the last line also gives the desired bound

$$\|b - b_{\mathcal{G}}\|_2^2 \ = \ t_r \ \leq \ \left(1 + \frac{3}{\gamma}\right) OPT_g \ .$$

□

Algorithm 16 Tail approximation for group sparsity

```
function GroupTail($b, g, \gamma$)
    $m \leftarrow \max_{G \in \mathbb{G}} |G|$
    $b_{\min} \leftarrow \min_{i \in [d], b_i \neq 0} |b_i|$
    $\lambda_\ell \leftarrow \frac{b_{\min}^2}{4g\gamma}$
    $\lambda_r \leftarrow 4\|b\|_2^2 \log m$
    $\varepsilon \leftarrow \frac{b_{\min}^2}{2g\gamma}$
    $\mathcal{G}_\ell \leftarrow$ LagrangeApprox($b, \lambda_\ell$)
    if $|\mathcal{G}_\ell| \leq (2+\gamma) g \log m$ then
        return $\mathcal{G}_\ell$
    end if
    while $\lambda_r - \lambda_\ell \geq \varepsilon$ do                          ▷ Binary search
        $\lambda_m \leftarrow \frac{\lambda_r + \lambda_\ell}{2}$
        $\mathcal{G}_\ell \leftarrow$ LagrangeApprox($b, \lambda_m$)
        if $2g \log m \ \leq \ |\mathcal{G}_m| \ \leq \ (2+\gamma) g \log m$ then
            return $\mathcal{G}_m$
        end if
        if $|\mathcal{G}_m| \leq 2g \log m$ then
            $\lambda_r \leftarrow \lambda_m$
        else
            $\lambda_\ell \leftarrow \lambda_m$
        end if
    end while
    $\mathcal{G}_r \leftarrow$ LagrangeApprox($b, \lambda_r$)
    return $\mathcal{G}_r$
end function

function LagrangeApprox($b, \lambda$)
    $\tilde{b} \leftarrow \mathrm{cl}_{\sqrt{\lambda}}(b)$
    $m \leftarrow \max_{G \in \mathbb{G}} |G|$
    $\mathcal{G}_\lambda \leftarrow$ GreedyLagrange($\tilde{b}, \lambda / \log m$)
    return $\mathcal{G}_\lambda$
end function
```

Chapter 5

Tree Sparsity

5.1 Introduction

Over the last decade, the concept of sparsity has attracted significant attention in signal processing, statistics, and machine learning. In addition to "standard" sparsity, several classes of real-world data possess additional structure. An important type of structure is *tree sparsity* (also known as *hierarchical sparsity*), which arises in multiple applications. A concrete example are wavelet representations of images [160]. The coefficients of the wavelet basis can naturally be arranged as a tree due to the multi-scale nature of wavelets. Moreover, the dominant coefficients in the wavelet representation often form a *rooted, connected tree* [34]. Using the fact that the large coefficients are connected (as opposed to being spread over the entire wavelet tree) leads to performance improvements when working with wavelet representations of images. Another example of tree sparsity occurs in genomics when features in a supervised learning problem can be arranged in a hierarchy [140].

As in previous chapters, our goal is to incorporate this type of constraint into optimization problems via projected gradient descent. Hence we are interested in the problem of projecting an arbitrary input to the set of tree-sparse vectors. More formally, we define the *tree sparsity* problem: given an arbitrary vector $x \in \mathbb{R}^n$, find the k-sparse and tree-structured vector $\widehat{x}$ that minimizes the error $\|x - \widehat{x}\|_2$.

Over the last two decades, the tree sparsity problem and its variants have been the subject of extensive studies in several areas. The problem arises in several sub-fields of signal processing such as compression and denoising [79, 209]. In combinatorial optimization, the problem constitutes an important special case of the rooted k-MST problem.[1] The problem is NP-hard in general, but solvable on trees in polynomial time [156, 183]. In machine learning, the problem was formulated in the context of optimal decision tree pruning [51]. Perhaps most prominently, the problem has been studied in the compressive sensing literature. Here, it is used to identify tree sparse approximations of signals represented in a wavelet basis (see e.g., [36, 37, 64, 89, 121] as well as the recent survey [120]).

[1] In the rooted k-MST problem we are given an edge-weighted graph G, the size parameter k, and a node v. The goal is to find a subtree of G with k edges that contains v and has the minimum weight. Note that for the case when G is a tree, rooted k-MST is computationally equivalent to tree sparsity: the weight of each edge can be assigned to one of its endpoints that points "away" from the root v, and minimization can be replaced by maximization by negating each weight and adding to it the sum of all weights W.

Head / Tail	Running time	Solution sparsity	Approximation ratio	Tree depth
Both [64]	$O(kn)$	k	1	$O(\log n)$
Tail	$O(n \log n)$	$2k$	≤ 2	$O(\log n)$
Head	$O(n \log n + k \log^2 n)$	$2k + O(\log n)$	$\geq 1/4$	$O(\log n)$
Tail	$n(\log n + 1/\epsilon)^{O(1)}$	k	$\leq (1+\epsilon)$	any
Head	$n(\log n + 1/\epsilon)^{O(1)}$	k	$\geq (1-\epsilon)$	any
Head	$O(n \log n)$	k	$\geq (k-d)/16k$	d

Table (5.1): A summary of our algorithms for the tree sparsity problem.

The best available (theoretical) performance for this problem is achieved by the dynamic-programming (DP) approach of [64], building upon the algorithm first developed in [51]. For an input vector of length n and target sparsity k, the algorithm has a runtime of $O(nk)$.[1] Unfortunately, this means that the algorithm does not easily scale to even moderate problem sizes. For example, a modestly-sized natural image (say, of size $n = 512 \times 512$) is usually only tree-sparse with parameter $k \approx 10^4$. In this case, nk exceeds 2.5×10^9. Considering that projected gradient descent invokes the projection oracle many times, it is easy to see that the running time quickly becomes impractical for megapixel-size images with $n = 10^6$ or $n = 10^7$.

Our contributions. In this chapter, we develop new approaches for the tree sparsity problem. As in previous chapters, we leverage our *approximation-tolerant* variant of projected gradient descent and introduce fast algorithms for head and tail approximations. At a high level, we introduce two classes of algorithms (also see Table 5.1 for an overview):

- Algorithms for *balanced trees* (Sections 5.2 and 5.3). The balanced setting is a relevant special case of the tree sparsity problem because wavelet decompositions used in compressive sensing give rise to balanced tree structures. Oure algorithms for this case are practical and especially the tail approximation algorithm performs well in experiments.

- Algorithms for *arbitrary tree* (Section 5.6). To understand the general version of the tree sparsity problem, we also provide algorithms that work for imbalanced trees. These algorithms still in run in nearly-linear time and achieve better approximation ratios than our algorithms for the balanced case. However, the theoretical power comes at a cost of extra logarithmic factors in the running time.

In addition, we also provide a conditional hardness result for the tree sparsity problem in arbitrary trees (see Section 5.6.4). In particular, we relate tree sparsity to the $(\max, +)$-convolution problem, which is a basic primitive in discrete algorithms. Our reduction is evidence that an exact projection for tree sparsity is indeed a challenging algorithmic task and that it may be impossible to achieve running times that are significantly faster than $O(nk)$. Hence approximate projections may again be necessary for large tree sparsity problems, even though an exact projection is possible in polynomial time.

[1] The algorithm in [CT13] was designed for *balanced* binary trees. However, in Section 5.6.7 we show the algorithm can be extended to arbitrary binary trees.

At a high level, our tail approximation can be seen as an extension of the complexity-penalized residual sum of squares (CPRSS) formulation proposed by Donoho [89]. We first pose the exact tree projection as a (nonconvex) sparsity-constrained optimization problem. We then perform a Lagrangian relaxation of the sparsity constraint and, similar to Donoho, solve the relaxed problem using a dynamic program (DP) with running time of $O(n)$. We then iterate this step a logarithmic number of times by conducting a binary search over the Lagrangian relaxation parameter until we arrive at the target sparsity k. A careful termination criterion for the binary search gives our desired approximation guarantee.

As a concrete application of our approximate projections, we instantiate them in a compressive sensing setting (Section 5.4). Moreover, we conduct numerical experiments with both synthetic and real data in Section 5.5. Our empirical results show two interesting phenomena:

- First, we see that our recovery algorithm with approximate projections has essentially no drawbacks in sample complexity. In particular, the empirical sample complexity of our algorithm matches that of exact projections.
- Moreover, our approximate projection has a very fast empirical running time. A single run of our approximate projection is comparable to the cost of computing a gradient. Hence running projected gradient descent with our approximate projection comes at essentially no computational cost compared to gradient descent without projections (or projected gradient descent for standard sparsity).

These findings show that approximate projections combine the statistical advantages of a sophisticated data model (i.e., constraint set) with the computational advantages of a simpler method. So at least for tree sparsity, the approximate projection framework allows us to get the best of both worlds. This is another concrete instance where the statistical setting leads to an algorithmic problem that is easier than what an arguably more classical worst-case perspective would suggest.

5.1.1 Related work

The problem of projecting into the tree-sparsity model has received a fair amount of attention in the literature over the last two decades. Researchers have proposed several algorithms such as the condensing sort-and-select algorithm (CSSA) [37], complexity-penalized residual sum-of-squares (CPRSS) [89], and optimal pruning [51]. However, all of these algorithms either run in time $\Omega(n^2)$ or fail to provide projection guarantees for general inputs. More recently, Cartis and Thompson [64] give a dynamic programming algorithm for exact projections running in time $O(nk)$. While this is an improvement, the running time can still be impractical for large inputs.

5.1.2 Preliminaries

Sparsity and structure. A vector $x \in \mathbb{R}^n$ is said to be k-sparse if no more than k of its coefficients are nonzero. The support of x, denoted by $\mathrm{supp}(x) \subseteq [n]$, indicates the locations of its nonzero entries.

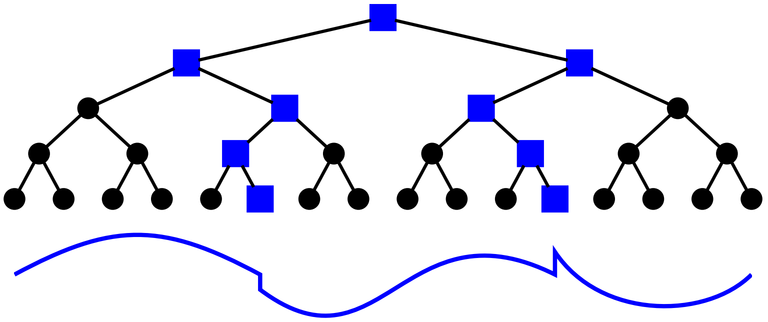


Figure (5-1): A binary wavelet tree for a one-dimensional signal. The squares denote the large wavelet coefficients that arise from the discontinuities in the piecewise smooth signal drawn below. The support of the large coefficients forms a rooted, connected tree.

Suppose that we possess some additional prior information about our vectors of interest. One way to model this information is as follows [36]: denote the set of *allowed supports* with $\mathbb{M}_k = \{\Omega_1, \Omega_2, \ldots, \Omega_L\}$, where $\Omega_i \subseteq [n]$ and $|\Omega_i| = k$. Often it is useful to work with the closure of $\mathbb{M}_k$ under taking subsets, which we denote with $\mathbb{M}_k^+ = \{\Omega \subseteq [n] \mid \Omega \subseteq S \text{ for some } S \in \mathbb{M}_k\}$. Then, we define a *structured sparsity model*, $\mathcal{M}_k \subseteq \mathbb{R}^n$, as the set of vectors $\mathcal{M}_k = \{x \in \mathbb{R}^n \mid \text{supp}(x) \in \mathbb{M}_k^+\}$. The number of allowed supports L is called the "size" of the model $\mathcal{M}_k$. Typically we have $L \ll \binom{n}{k}$, which leads to better sample complexity than "standard" sparsity (without extra structure).

Tree-sparsity Our focus in this chapter is the *tree-structured sparsity model*, or simply, tree sparsity. We assume that the n coefficients of a signal $x \in \mathbb{R}^n$ can be arranged as the nodes of a full d-ary tree (for simplicity, we mostly focus on the binary case here). Then the tree-sparsity model comprises the set of k-sparse vectors whose nonzero coefficients form a *rooted, connected* subtree. The size of this model is upper bounded by $L \leq (2e)^k/(k+1)$ [36].

For the rest of the chapter, we denote the set of supports corresponding to a subtree rooted at node i with $\mathbb{T}_i$. Then $\mathbb{M}_k = \{\Omega \subseteq [n] \mid \Omega \in \mathbb{T}_1 \text{ and } |\Omega| = k\}$ is the formal definition of the tree-sparsity model (we assume node 1 to be the root of the entire signal).

The tree-sparsity model can be used for modeling a variety of signal classes. A compelling application of this model emerges while studying the wavelet decomposition of piecewise-smooth signals and images. It is known that wavelets act as *local* discontinuity detectors [160]. Since the supports of wavelets at different scales are nested, a signal discontuinity will give rise to a chain of significant coefficients along a branch of the wavelet tree (see Figure 5-1).

Approximate projections. We briefly recall our definitions of approximate projections from Chapter 2 in the context of tree sparsity. First, we define the *model-projection* problem for $\mathcal{M}_k$ as follows: given $x \in \mathbb{R}^n$, determine a $x^* \in \mathcal{M}_k$ such that $\|x - x^*\|_2$ is minimized.[1] Since this problem is non-convex, it can be computationally demanding (see Section 5.6.4 for evidence). Our *approximation-tolerant* variant of projected gradient offers an alternative.

[1] Our algorithms for tree sparse projections also work for other norm parameters. For simplicity, we state the ℓ_2 case here.

Instead of a single *exact* projection algorithm, our framework requires two *approximate* projection algorithms with two complementary notions of approximation:

- A head approximation algorithm $H(x, k)$ that satisfies the following guarantee: Let $\widehat{\Omega} = H(x, k)$. Then $\widehat{\Omega} \in \mathbb{M}^+_{c_1 k}$ and
$$\|x_{\widehat{\Omega}}\|_2 \;\geq\; c_2 \cdot \max_{\Omega \in \mathbb{M}_k} \|x_\Omega\|_2$$
for some constants $c_1 \geq 1$ and $c_2 \leq 1$.
- A tail approximation algorithm $T(x, k)$ that satisfies the following guarantee: Let $\widehat{\Omega} = T(x, k)$. Then $\widehat{\Omega} \in \mathbb{M}^+_{c_1 k}$ and
$$\|x - x_{\widehat{\Omega}}\|_2 \;\leq\; c_2 \cdot \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_2$$
for some constants $c_1 \geq 1$ and $c_2 \geq 1$.

Using approximate model-projection algorithms, our framework from Chapter 2 achieves the same estimation guarantees as projected gradient descent with an exact model-projection.

5.2 Head approximation for the tree-sparsity model

We now introduce our head approximation algorithm for the tree-sparsity model. In order to simplify the analysis, we will assume that $k \geq \lceil \log_d n \rceil$. Note that we can always reduce the input to this case by removing layers of the tree with depth greater than k. Our approach is based on the following structural result about decompositions of d-ary trees.

Lemma 5.1. *Let T be a d-ary tree with $|T| = k$. Moreover, let $\alpha \in \mathbb{N}, \alpha \geq 1$. Then T can be decomposed into a set of disjoint, connected subtrees $S = \{T_1, \ldots, T_\beta\}$ such that $|T_i| \leq d\alpha$ for all $i \in [\beta]$ and $\beta = |S| \leq \lceil \frac{k}{\alpha} \rceil$.*

Proof. We first show that given a d-ary tree U with at least $d\alpha + 1$ nodes, we can find a subtree U' with $\alpha \leq |U'| \leq d\alpha$. Consider the following algorithm FindTree:

```
function FindTree($U$)
    Let $U'$ be the subtree of $U$ maximizing $|U'|$.
    if $|U'| \leq d\alpha$ then
        return $U'$
    else
        return FindTree($U'$)
    end if
end function
```

First, note that $|U'| \geq \alpha$ because of the pigeonhole principle and $|U| \geq d\alpha + 1$. Moreover, the size of U decreases with each recursive invocation. Since T is a finite tree, FindTree eventually terminates. When it does, the algorithm returns a subtree U' with $\alpha \leq |U'| \leq d\alpha$.

We use FindTree repeatedly on T in order to build a decomposition S with the desired properties. After identifying a subtree U', we remove it from T and find the next subtree in the remaining tree until at most $d\alpha$ nodes are left. We use this remaining subtree as the final subtree in S.

Algorithm 17 (HeadApprox) Head approximation for the tree-sparsity model

```
function HeadApprox($(x, k, d, p, \alpha)$)
    Run ETP on $x$ with sparsity parameter $k' = d\alpha$.
    $x^{(1)} \leftarrow x$
    for $i \leftarrow 1, \ldots, \lceil \frac{k}{\alpha} \rceil$ do
        $\widehat{\Omega}_i \leftarrow \operatorname*{arg\,max}_{\Omega \in \mathbb{T},\, |\Omega| = d\alpha} \left\| x^{(i)}_{\Omega} \right\|_p$
        $x^{(i+1)} \leftarrow x^{(i)}, \qquad x^{(i+1)}_{\widehat{\Omega}_i} \leftarrow 0$
        for $j \in \widehat{\Omega}_i \cup \text{root-path}(\widehat{\Omega}_i)$ (in bottom-up order) do
            Update the DP table for node $j$ up to sparsity $k' = d\alpha$.
        end for
    end for
    return $\widehat{\Omega} \leftarrow \bigcup_{i=1}^{\lceil \frac{k}{\alpha} \rceil} \widehat{\Omega}_i \cup \text{root-path}(\widehat{\Omega}_i)$
end function
```

By construction, S is a set of disjoint, connected subtrees. Moreover, the remaining subtree and each subtree returned by FindTree satisfy $|T_i| \leq d\alpha$. Finally, the decomposition contains at most $\lceil \frac{k}{\alpha} \rceil$ subtrees because every subtree we remove from T has at least α nodes. □

In addition to the tree decomposition, our head-approximation algorithm builds on the exact tree projection algorithm (ETP) introduced in [64]. The algorithm finds the best tree-sparse approximation for a given input via dynamic programming (DP) in $O(nkd)$ time.[1] We run ETP with a small sparsity value $k' < k$ in order to find optimal subtrees of size k'. We then assemble several such subtrees into a solution with a provable approximation guarantee. We use the fact that ETP calculates the DP table entries in the following way: if the DP tables corresponding to the children of node i are correct, the DP table for node i can be computed in $O(k'^2)$ time. The time complexity follows from the structure of the DP tables: for every node and $l \leq k'$, we store the value of the best subtree achievable at that node with sparsity exactly l. We can now state our head-approximation algorithm (Algorithm 17) and the corresponding guarantees.

Theorem 5.2. *Let $x \in \mathbb{R}^n$ be the coefficients corresponding to a d-ary tree rooted at node* 1*. Also, let $p \geq 1$ and $\alpha \geq 1$. Then* HeadApprox(x, k, d, p, α) *returns a support $\widehat{\Omega}$ satisfying $\|x_{\widehat{\Omega}}\|_p \geq \left(\frac{1}{4}\right)^{1/p} \max_{\Omega \in \mathbb{M}_k} \|x_\Omega\|_p$. Moreover, $\widehat{\Omega} \in \mathbb{M}^+_\gamma$ with $\gamma = \lceil \frac{k}{\alpha} \rceil (d\alpha + \lceil \log_d n \rceil)$.*

Proof. Let $\Omega^* \in \mathbb{M}_k$ be an optimal support, i.e.,

$$\|x_{\Omega^*}\|_p \;=\; \max_{\Omega \in \mathbb{M}_k} \|x_\Omega\|_p \,.$$

Using Lemma 5.1, there is a decomposition of Ω^* into disjoint sets $\Omega^*_1, \ldots, \Omega^*_\beta$ such that $\Omega^*_i \in \mathbb{T}$, $|\Omega^*_i| \leq d\alpha$ and $\beta \leq \lceil \frac{k}{\alpha} \rceil$. The contribution of Ω^*_i to the overall solution is $\left\| x_{\Omega^*_i} \right\|_p^p$.

[1] While ETP as stated in [64] works for $p = 2$ only, the algorithm can easily be extended to arbitrary norm parameters p.

Now, compare the contributions of our subtrees $\widehat{\Omega}_i$ to these quantities. When finding $\widehat{\Omega}_i$ for $i \in [\beta]$, one of the following two cases holds:

1. $\left\|x^{(i)}_{\Omega^*_i}\right\|_p^p \geq \frac{1}{2}\left\|x_{\Omega^*_i}\right\|_p^p$. Since Ω^*_i is a candidate in the search for $\widehat{\Omega}_i$ in line 5, we have $\left\|x^{(i)}_{\widehat{\Omega}_i}\right\|_p^p \geq \left\|x^{(i)}_{\Omega^*_i}\right\|_p^p \geq \frac{1}{2}\left\|x_{\Omega^*_i}\right\|_p^p$.
2. $\left\|x^{(i)}_{\Omega^*_i}\right\|_p^p < \frac{1}{2}\left\|x_{\Omega^*_i}\right\|_p^p$. Therefore, $\widehat{\Omega}_1, \ldots, \widehat{\Omega}_{i-1}$ have already covered at least half of the contribution of Ω^*_i. Formally, let $C_i = \Omega^*_i \cap \bigcup_{j=1}^{i-1} \widehat{\Omega}_j$. Then $\|x_{C_i}\|_p^p \geq \frac{1}{2}\left\|x_{\Omega^*_i}\right\|_p^p$.

Let $A = \{i \in [\beta] \,|\, \text{case 1 holds for}\, \widehat{\Omega}_i\}$ and $B = \{i \in [\beta] \,|\, \text{case 2 holds for}\, \widehat{\Omega}_i\}$. For the set A we have

$$\left\|x_{\widehat{\Omega}}\right\|_p^p \quad = \quad \sum_{i=1}^{\lceil \frac{k}{\alpha} \rceil} \left\|x^{(i)}_{\widehat{\Omega}_i}\right\|_p^p \quad \geq \quad \sum_{i \in A} \left\|x^{(i)}_{\widehat{\Omega}_i}\right\|_p^p + \sum_{i \in B} \left\|x^{(i)}_{\widehat{\Omega}_i}\right\|_p^p \quad \geq \quad \frac{1}{2} \sum_{i \in A} \left\|x_{\Omega^*_i}\right\|_p^p . \qquad (5.1)$$

Now, consider the set B. Since the Ω^*_i are disjoint, so are the C_i. Moreover, $C_i \subseteq \widehat{\Omega}$ and therefore

$$\left\|x_{\widehat{\Omega}}\right\|_p^p \quad \geq \quad \sum_{i=1}^{\beta} \|x_{C_i}\|_p^p \quad \geq \quad \sum_{i \in B} \|x_{C_i}\|_p^p \quad \geq \quad \frac{1}{2} \sum_{i \in B} \left\|x_{\Omega^*_i}\right\|_p^p . \qquad (5.2)$$

Combining (5.1) and (5.2), we get

$$2\left\|x_{\widehat{\Omega}}\right\|_p^p \quad \geq \quad \frac{1}{2} \sum_{i \in A} \left\|x_{\Omega^*_i}\right\|_p^p + \frac{1}{2} \sum_{i \in B} \left\|x_{\Omega^*_i}\right\|_p^p \quad \geq \quad \frac{1}{2} \|x_{\Omega^*}\|_p^p .$$

Raising both sides to power $1/p$ gives the guarantee in the theorem. For the sparsity bound, note that $\left|\widehat{\Omega}_i\right| \leq d\alpha$ and $\left|\text{root-path}(\widehat{\Omega}_i)\right| \leq \lceil \log_d n \rceil$. Since we take the union over $\lceil \frac{k}{\alpha} \rceil$ such sets, the theorem follows. □

Next, we analyze the running time of our head approximation algorithm. The proof is a direct consequence of Theorems 5.2 and 5.3.

Theorem 5.3. HEADAPPROX(x, k, d, p, α) *has a running time of*

$$O(nd^2\alpha + \frac{k}{\alpha}(d\alpha + \log n)(d^2\alpha^2 + \log n)) .$$

Proof. Since we invoke ETP with $k' = d\alpha$, the initial call to ETP takes $O(nd^2\alpha)$ time. We can implement the arg max in line 5 with a binary heap containing the DP table entries for all nodes and sparsity $d\alpha$. Therefore, this step takes $O(\log n)$ time per iteration of the outer loop.

The cost of the outer loop is dominated by the updates performed in line 8. For each node in $\widehat{\Omega}_i \cup \text{root-path}(\widehat{\Omega}_i)$, we have to update its DP table ($O(d^2\alpha^2)$ time) and then its entry in the binary heap ($O(\log n)$ time). Since $\left|\widehat{\Omega}_i\right| \leq d\alpha$ and $\left|\text{root-path}(\widehat{\Omega}_i)\right| \leq \lceil \log_d n \rceil$, the total cost over all iterations of the outer loop is $O(\frac{k}{\alpha}(d\alpha + \log n)(d^2\alpha^2 + \log n))$. Combining this with the cost of ETP gives the running time bound in the theorem. □

Overall, we arrive at the following final guarantee.

Corollary 5.4. *Let* $\alpha = \lceil \log_d n \rceil$. *Then* $\textsc{HeadApprox}(x, k, d, p, \alpha)$ *returns a support* $\widehat{\Omega} \in \mathbb{M}^+_{k(2d+2)}$ *satisfying* $\|x_{\widehat{\Omega}}\|_p \geq \left(\frac{1}{4}\right)^{1/p} \max_{\Omega \in \mathbb{M}_k} \|x_\Omega\|_p$. *Moreover, the algorithm runs in time* $O(n \log n + k \log^2 n)$ *for fixed* d.

5.3 Tail approximation for the tree-sparsity model

Recall that the main goal for an *exact* tree-projection is the following: given an input $x \in \mathbb{R}^n$, minimize the quantity

$$\|x - x_\Omega\|_p$$

over $\Omega \in \mathbb{M}_k$ for $1 \leq p < \infty$.[1] For a tail *approximation*, we are interested in the following relaxed problem: find a $\widehat{\Omega} \in \mathbb{M}_{k'}$ with $k' \leq c_2 k$ and

$$\|x - x_{\widehat{\Omega}}\|_p \;\leq\; c_1 \cdot \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p \, . \tag{5.3}$$

We highlight two aspects of the modified problem (5.3):

- Instead of projecting into the model $\mathcal{M}_k$, we project into the slightly larger model $\mathcal{M}_{k'}$.
- Instead of finding the best projection, we provide a multiplicative guarantee for the approximation error.

We propose an approximation algorithm that achieves (5.3). First, we approach the modified problem via a Lagrangian relaxation, i.e., we relax the sparsity constraint and keep the requirement that the support Ω forms a connected subtree:

$$\arg\min_{\Omega \in \mathbb{T}_1} \|x - x_\Omega\|_p^p + \lambda |\Omega| \, . \tag{5.4}$$

Note that the parameter λ controls the trade-off between the approximation error and the sparsity of the recovered support. Second, we use a binary search to find a suitable value of λ, resulting in an overall recovery guarantee of the form (5.3).

5.3.1 Solving the Lagrangian relaxation

We first transform the problem in (5.4) into a slightly different form. Note that (5.4) is equivalent to $\arg\max_{\Omega \in \mathbb{T}_1} \|x_\Omega\|_p^p - \lambda|\Omega|$. We can now rewrite this problem as

$$\arg\max_{\Omega \in \mathbb{T}_1} \sum_{i \in \Omega} y_i \tag{5.5}$$

where $y_i = |x_i|^p - \lambda$. Hence the goal is to find a rooted subtree which maximizes the sum of weights y_i contained in the subtree. In contrast to the original tree approximation problem,

[1] Our approximation algorithm also solves the ℓ_∞-version of the tree-sparse approximation problem (and, in this case, even identifies the optimal projection, not only a provably good one). Since the focus in constrained estimation problems usually lies on $p = 2$, we limit our proofs here to ℓ_p-norms with $p < \infty$.

there is no sparsity constraint, but the weights associated with nodes can now be negative.

Problem (5.5) is similar to the CPRSS formulation proposed by Donoho [89]. However, our technical development here is somewhat different because the underlying problems are not exactly identical. Therefore, we summarize our approach in Algorithm 18 and outline its proof for completeness.

Theorem 5.5. *Let $x \in \mathbb{R}^n$ be the coefficients corresponding to a tree rooted at node 1, and let $p \geq 1$. Then* FindTree(x, λ, p) *runs in linear time and returns a support $\widehat{\Omega} \in \mathbb{T}_1$ satisfying*

$$\|x - x_{\widehat{\Omega}}\|_p^p + \lambda|\widehat{\Omega}| = \min_{\Omega \in \mathbb{T}_1} \|x - x_\Omega\|_p^p + \lambda|\Omega| \ .$$

Proof. As above, let $y_i = |x_i|^p - \lambda$. For a support $\Omega \in [n]$, let $y(\Omega) = \sum_{i \in \Omega} y_i$. Futhermore, we denote the total weight of the best subtree rooted at node i with $b_i^* = \max_{\Omega \in \mathbb{T}_i} y(\Omega)$. Note that $b_i^* = \max(0, y_i + \sum_{j \in \text{children(i)}} b_j^*)$ because we can choose nodes in the subtrees of i independently. A simple inductive argument shows that after the call to CalculateBest$(1, x, \lambda, p)$, we have $b_i = b_i^*$ for $i \in [n]$.

Similarly, a proof by induction shows that after a call to FindSupport(i), we have $y(\Omega_i) = b_i^*$. So the support $\widehat{\Omega}$ returned by FindTree(x, λ, p) satisfies

$$y(\widehat{\Omega}) \ = \ \max_{\Omega \in \mathbb{T}_1} y(\Omega) \ .$$

This implies the guarantee in the theorem.

The time complexity follows from the fact that the algorithm makes a constant number of passes over the tree. □

5.3.2 Binary search

Next, we use Alg. 18 in order to achieve the desired approximation guarantee (5.3). Since the Lagrangian relaxation (5.4) gives us only indirect control over the sparsity k', we perform a binary search over λ to find a suitable value. Alg. 19 contains the corresponding pseudo code. In addition to x, k, c, and p, the algorithm takes an additional parameter δ, which controls the maximum number of iterations of the binary search.

Theorem 5.6. *Let $x \in R^n$ be the coefficients corresponding to a tree rooted at node 1. Moreover, let $c > 1$, $\delta > 0$, and $p \geq 1$. Then* TreeApprox(x, k, c, p, δ) *returns a support $\widehat{\Omega} \in \mathbb{M}_{ck}^+$ satisfying*

$$\|x - x_{\widehat{\Omega}}\|_p \leq \left(1 + \frac{1}{c-1} + \delta\right)^{1/p} \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p \ .$$

Moreover, the algorithm runs in time $O(n(\log \frac{k}{\delta} + p \log \frac{x_{\max}}{x_{\min}}))$, where $x_{\max} = \max_{i \in [n]} |x_i|$ and $x_{\min} = \min_{i \in [n], x_i > 0} |x_i|$.

Proof. We analyze the three cases in which TreeApprox returns a support. First, note that the condition in line 2 can be checked efficiently: connect all nonzero entries in x to

Algorithm 18 (FindTree) Solving the Lagrangian relaxation

```
function FindTree($x, \lambda, p$)
    CalculateBest($1, x, \lambda, p$)
    return $\widehat{\Omega} \leftarrow$ FindSupport($1$)
end function
function CalculateBest($i, x, \lambda, p$)
    $b_i \leftarrow |x_i|^p - \lambda$
    for $j \in \text{children}(i)$ do
        CalculateBest($j, x, \lambda, p$)
        $b_i \leftarrow b_i + b_j$
    end for
    $b_i \leftarrow \max(0, b_i)$
end function
function FindSupport($i$)
    if $b_i = 0$ then
        $\Omega_i \leftarrow \{\}$
    else
        $\Omega_i \leftarrow \{i\}$
        for $j \in \text{children}(i)$ do
            $\Omega_i \leftarrow \Omega_i \cup$ FindSuppport($j$)
        end for
    end if
    return $\Omega_i$
end function
```

the root node and denote the resulting support with $\widehat{\Omega}$. If $\left|\widehat{\Omega}\right| \leq k$, we have $\widehat{\Omega} \in \mathbb{M}_k^+$ and $x \in \mathcal{M}_k$. Otherwise, $x \notin \mathcal{M}_k$ and the tail approximation error is greater than zero.

Second, if the algorithm returns in line 11, we have $\left|\widehat{\Omega}\right| \leq ck$ and $\widehat{\Omega} \in \mathbb{T}_1$ (Theorem 5.5). Hence $\widehat{\Omega} \in \mathbb{M}_{ck}^+$. Moreover, Theorem 5.5 implies

$$\|x - x_{\widehat{\Omega}}\|_p^p + \lambda_m |\widehat{\Omega}| \leq \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p^p + \lambda_m |\Omega| \; .$$

Since $\left|\widehat{\Omega}\right| \geq k = |\Omega|$ for $\Omega \in \mathbb{M}_k$, we have $\|x - x_{\widehat{\Omega}}\|_p \leq \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p$.

Finally, consider the return statement in line 18. Let Ω_l and Ω_r be the supports corresponding to λ_l and λ_r, respectively. Moreover, let $\Omega_{OPT} \in \mathbb{M}_k$ be a support with $\|x - x_{\Omega_{OPT}}\|_p = \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p$. We use the shorthands $t_l = \|x - x_{\Omega_l}\|_p^p$, $k_l = |\Omega_l|$, and the corresponding definitions for t_r, k_r, t_{OPT}, and k_{OPT}. Note that throughout the binary search, we maintain the invariants $k_r \geq ck$ and $k_l \leq k$. The invariants also hold before the first iteration of the binary search due to our initial choices for λ_l and λ_r ($\lambda = 0$ implies $y_i \geq 0$ and $\lambda = x_{\max}^p$ implies $y_i \leq 0$, both for all $i \in [n]$).

Algorithm 19 (TreeApprox) Tree-sparse approximation

```
function TreeApprox($(x, k, c, p, \delta)$)
    if there is a $\widehat{\Omega} \in \mathbb{M}_k$ with $\mathrm{supp}(x) \subseteq \widehat{\Omega}$ then
        return $\widehat{\Omega}$
    end if
    $x_{\max} \leftarrow \max_{i \in [n]} |x_i|, \quad x_{\min} \leftarrow \min_{i \in [n], x_i > 0} |x_i|$
    $\lambda_l \leftarrow x_{\max}^p, \quad \lambda_r \leftarrow 0, \quad \varepsilon \leftarrow \frac{\delta\, x_{\min}^p}{k}$
    while $\lambda_l - \lambda_r > \varepsilon$ do
        $\lambda_m \leftarrow \frac{\lambda_l + \lambda_r}{2}$
        $\widehat{\Omega} \leftarrow$ FindTree$(x, \lambda_m, p)$
        if $\left|\widehat{\Omega}\right| \geq k$ and $\left|\widehat{\Omega}\right| \leq ck$ then
            return $\widehat{\Omega}$
        else if $\left|\widehat{\Omega}\right| < k$ then
            $\lambda_l \leftarrow \lambda_m$
        else
            $\lambda_r \leftarrow \lambda_m$
        end if
    end while
    return $\widehat{\Omega} \leftarrow$ FindTree$(x, \lambda_l, p)$
end function
```

From Theorem 5.5, we have $t_r + \lambda_r k_r \leq t_{OPT} + \lambda_r k_{OPT}$. This implies

$$\begin{aligned} \lambda_r(k_r - k_{OPT}) &\leq t_{OPT} - t_r \\ \lambda_r(ck - k) &\leq t_{OPT} \\ \lambda_r &\leq \frac{t_{OPT}}{k(c-1)} \; . \end{aligned}$$

At the end of the binary search we have $\lambda_l - \lambda_r \leq \varepsilon$, giving

$$\lambda_l \leq \frac{t_{OPT}}{k(c-1)} + \varepsilon \; . \tag{5.6}$$

Theorem 5.5 also implies $t_l + \lambda_l k_l \leq t_{OPT} + \lambda_l k_{OPT}$. Together with (5.6), we get

$$\begin{aligned} t_l &\leq t_{OPT} + \lambda_l k \\ &\leq t_{OPT} + \frac{t_{OPT}}{c-1} + \varepsilon k \\ &\leq t_{OPT}\left(1 + \frac{1}{c-1}\right) + \delta x_{\min}^p \\ &\leq \left(1 + \frac{1}{c-1} + \delta\right) t_{OPT} \; . \end{aligned}$$

The last line follows from the fact that $x \notin \mathcal{M}_k$ and hence $t_{OPT} \geq x_{\min}^p$. Taking the p-th root on both sides gives the guarantee in the theorem.

Each iteration of the binary search runs in linear time (Theorem 5.5). The difference $\lambda_l - \lambda_r$

initially is $x_{\max}^p$ and is then halved in every iteration until it reaches ε. Hence the total number of iterations is at most

$$\begin{aligned} \log \frac{x_{\max}^p}{\varepsilon} &= \log \frac{x_{\max}^p k}{\delta x_{\min}^p} \\ &= \log \frac{k}{\delta} + p \log \frac{x_{\max}}{x_{\min}} \ . \end{aligned}$$

□

In practical applications, δ, p, $x_{\max}$, and $x_{\min}$ can be considered constants, which gives a running time of $O(n \log k)$. Next, we remove the dependence on $\frac{x_{\max}}{x_{\min}}$ and give an algorithm running in *strongly polynomial* time $O(n \log n)$ while achieving the same approximation guarantee.

5.3.3 A strongly polynomial variant

The goal of this section is to develop a tail approximation algorithm for the tree sparsity model with a strongly polynomial running time. As for our previous algorithm, we consider the Lagrangian relaxation

$$\arg\min_{\Omega \in \mathbb{T}_1} \|x - x_\Omega\|_p^p + \lambda |\Omega| \ ,$$

where the parameter λ controls the trade-off between the approximation error and the sparsity of the identified support. Our previous algorithm runs a binary search over λ in order to explore the Pareto curve of this trade-off. However, the running time of this algorithm is only weakly polynomial because it depends on both

$$x_{\max} = \max_{i \in [n]} |x_i| \qquad \text{and} \qquad x_{\min} = \min_{i \in [n], |x_i| > 0} |x_i| \ .$$

Below, we introduce an algorithm that exploits the structure of the Pareto curve in more detail and runs in strongly polynomial time $O(n \log n)$. In fact, our new algorithm constructs the shape of the *entire* Pareto curve and not only a single trade-off.

The Lagrangian relaxation is equivalent to

$$\arg\max_{\Omega \in \mathbb{T}_1} \|x_\Omega\|_p^p - \lambda |\Omega| \ .$$

Hence, we can rewrite this problem as

$$\arg\max_{\Omega \in \mathbb{T}_1} \sum_{i \in \Omega} y_i \qquad \text{where} \quad y_i = |x_i|^p - \lambda \ .$$

So for a given value of λ, the goal is to find a subtree Ω rooted at node 1 which maximizes the sum of weights y_i associated with the nodes in Ω.

In the following, we analyze how the solution to this problem changes as a function of λ and use this structure in our tail-approximation algorithm. On a high level, the optimal contribution of a node i is positive and decreasing up to a certain value of $\lambda = \gamma_i$, after which the contribution stays 0. So for $\lambda < \gamma_i$, a subtree rooted at node i can contribute positively to an overall solution. For $\lambda \geq \gamma_i$, we can ignore the subtree rooted at node i.

5.3.3.1 Properties of the Pareto curve

Let $b_i(\lambda)$ denote the maximum value achievable with a subtree rooted at i:

$$b_i(\lambda) = \max_{\Omega \in \mathbb{T}_i} \|x_\Omega\|_p^p - \lambda|\Omega| \; .$$

Our algorithm relies on two main insights: (i) $b_i(\lambda)$ is a piecewise linear function with at most n non-differentiable points (or "corners"), which correspond to the values of λ at which the optimal support changes. (ii) Starting with $\lambda = 0$, $b_i(\lambda)$ is strictly decreasing up to a certain value of λ, after which $b_i(\lambda) = 0$. Formally, we can state the properties of the Pareto curve as follows.

Lemma 5.7. *$b_i(\lambda)$ is piecewise linear. There is a value γ_i such that $b_i(\lambda) = 0$ for $\lambda \geq \gamma_i$ and $b_i(\lambda)$ is strictly decreasing for $\lambda \leq \gamma_i$. The corners of $b_i(\lambda)$ are the points $D_i = \{\gamma_i\} \cup \{\gamma \in \bigcup_{j \in \mathit{children}(i)} D_j \mid \gamma < \gamma_i\}$.*

Proof. A simple inductive argument shows that $b_i(\lambda)$ can be recursively defined as

$$b_i(\lambda) = \max(0,\; |x_i|^p - \lambda + \sum_{j \in \text{children}(i)} b_j(\lambda)) \; .$$

Note that the theorem holds for the leaves of the tree. By induction over the tree, we also get the desired properties for all nodes in the tree. We are using the fact that piecewise linear functions and strictly decreasing functions are closed under addition. Moreover, the corners of a sum of piecewise linear functions are contained in the union of the corners of the individual functions. □

Our algorithm does not compute the $b_i(\lambda)$ directly but instead keeps track of the following two quantities $s_i(\lambda)$ and $c_i(\lambda)$. For a given value of λ, $s_i(\lambda)$ denotes the sum achieved by the best subtree rooted at node i. Similarly, $c_i(\lambda)$ denotes the cardinality of the best subtree rooted at node i. These two quantities are easier to maintain algorithmically because they are piecewise constant. The proof of the next lemma follows directly from Lemma 5.7 and a similar inductive argument.

Lemma 5.8. *Let*

$$s_i(\lambda) = |x_i|^p + \sum_{\substack{j \in \mathit{children}(i) \\ b_j(\lambda) > 0}} s_j(\lambda) \qquad \textit{and} \qquad c_i(\lambda) = 1 + \sum_{\substack{j \in \mathit{children}(i) \\ b_j(\lambda) > 0}} c_j(\lambda) \; .$$

Then $s_i(\lambda)$ and $c_i(\lambda)$ are piecewise constant and monotonically decreasing. The discontinuities of $s_i(\lambda)$ and $c_i(\lambda)$ are D_i (see Lemma 5.7). At a discontinuity $\gamma \in D_i$ we have $\lim_{\delta \to 0^+} s_i(\gamma + \delta) = s_i(\gamma)$ and $\lim_{\delta \to 0^+} c_i(\gamma + \delta) = c_i(\gamma)$.

The following is an alternative characterization of $s_i(\lambda)$ and $c_i(\lambda)$. The proof follows by an induction over the tree.

Lemma 5.9. *Let*

$$\Omega_\lambda = \mathop{\arg\max}_{\substack{\Omega \in \mathbb{T}_i,\, \Omega \neq \{\} \\ b_j(\lambda) > 0 \text{ for } j \in \Omega \setminus \{i\}}} \|x_\Omega\|_p^p - \lambda |\Omega| \; .$$

Then

$$\begin{aligned} s_i(\lambda) &= \|x_{\Omega_\lambda}\|_p^p \quad \textit{and} \\ c_i(\lambda) &= |\Omega_\lambda| \; . \end{aligned}$$

We now establish a link between $s_i(\lambda)$, $c_i(\lambda)$ and $b_i(\lambda)$.

Lemma 5.10. *Let $f_i(\lambda) = s_i(\lambda) - \lambda c_i(\lambda)$. Then for $\lambda > \gamma_i$, we have $f_i(\lambda) < 0$. For $\lambda \leq \gamma_i$, we have $f_i(\lambda) = b_i(\lambda)$.*

Proof. We use the characterization of $s_i(\lambda)$ and $c_i(\lambda)$ stated in Lemma 5.9. For $\lambda < \gamma_i$, we have $i \in \arg\max_{\Omega \in \mathbb{T}_i} \|x_\Omega\|_p^p - \lambda|\Omega|$ and hence

$$\begin{aligned} f_i(\lambda) &= s_i(\lambda) - \lambda c_i(\lambda) \\ &= \|x_{\Omega_\lambda}\|_p^p - \lambda|\Omega_\lambda| \\ &= \max_{\Omega \in \mathbb{T}_i} \|x_\Omega\|_p^p - \lambda|\Omega| \\ &= b_i(\lambda) \end{aligned}$$

Since $f_i(\gamma_i) = 0$, we get $f_i(\lambda) = b_i(\lambda)$ for $\lambda \leq \gamma_i$.

Note that for $\lambda > \gamma_i$, we have $\arg\max_{\Omega \in \mathbb{T}_i} \|x_\Omega\|_p^p - \lambda|\Omega| = \{\}$, and $\{\}$ is the unique maximizer. Since $i \in \Omega_\lambda$ for all values of λ, we get $f_i(\lambda) < b_i(\lambda) = 0$ for $\lambda > \gamma_i$. □

The following lemma shows that for a given value of λ, we can find the next smallest discontinuity in $s_i(\lambda)$ and $c_i(\lambda)$ based solely on the current values of $s_i(\lambda)$ and $c_i(\lambda)$. This is an important ingredient in our algorithm because it allows us to build the Pareto curve incrementally.

Lemma 5.11. *Let $\lambda \geq 0$ with $\lambda \neq \gamma_i$ for $i \in [n]$ and let*

$$a = \mathop{\arg\max}_{i \in [n],\, b_i(\lambda) = 0} \frac{s_i(\lambda)}{c_i(\lambda)} \; .$$

Then

$$\gamma_a = \max_{i \in [n],\, \gamma_i \leq \lambda} \gamma_i \; .$$

Proof. Since $b_a(\lambda) = 0$, we have $\gamma_a \leq \lambda$ and hence $\gamma_a \leq \max_{i \in [n],\, \gamma_i \leq \lambda} \gamma_i$.

For contradiction, assume that there is a $\lambda \geq \gamma_j > \gamma_a$ and let γ_j be the largest such γ_j. From Lemmas 5.10 and 5.7, we have $s_j(\gamma_j) - \gamma_j c_j(\gamma_j) = f_j(\gamma_j) = b_j(\gamma_j) = 0$ and hence $\frac{s_j(\gamma_j)}{c_j(\gamma_j)} = \gamma_j$. Since s_j and c_j are constant in $[\gamma_j, \lambda]$, we have

$$\gamma_j = \frac{s_j(\lambda)}{c_j(\lambda)} \leq \frac{s_a(\lambda)}{c_a(\lambda)} \; . \tag{5.7}$$

We have $\gamma_j > \gamma_a$ and hence $s_a(\gamma_j) - \gamma_j c_a(\gamma_j) = f_a(\gamma_j) < 0$. Thus, $\gamma_j > \frac{s_a(\gamma_j)}{c_a(\gamma_j)}$. Since s_a and c_a are constant in $[\gamma_j, \lambda]$, we have $\gamma_j > \frac{s_a(\lambda)}{c_a(\lambda)}$, which is a contradiction to (5.7). □

Let γ_1 and γ_2 be two adjacent discontinuities in $s_i(\lambda)$ and $c_i(\lambda)$ with $\gamma_1 < \gamma_2$. Note that Ω_λ is constant for $\gamma_1 \leq \lambda < \gamma_2$ but $\Omega_{\gamma_2} \neq \Omega_{\gamma_1}$. As our last lemma, we show that Ω_{γ_1} is still an optimal solution for $\lambda = \gamma_2$.

Lemma 5.12. *Let $\lambda' > 0$ and let*

$$\lim_{\delta \to 0^+} s_i(\lambda' - \delta) = u$$
$$\lim_{\delta \to 0^+} c_i(\lambda' - \delta) = v \; .$$

Then $u - \lambda' v = f_i(\lambda')$.

Proof. First, we show that $f_i(\lambda)$ is continuous. Let

$$b'_i(\lambda) = |x_i|^p - \lambda + \sum_{\substack{j \in \text{children}(i) \\ b_j(\lambda) > 0}} b'_j(\lambda) \; .$$

By definition, we have $b'_i(\lambda) = s_i(\lambda) - \lambda c_i(\lambda) = f_i(\lambda)$. Moreover, an inductive argument similar to the one used for $b_i(\lambda)$ in Lemma 5.7 shows that $b'_i(\lambda)$ is piecewise linear and continuous. Therefore, $f_i(\lambda)$ is also continuous.

Since $s_i(\lambda)$ and $c_i(\lambda)$ have only finitely many discontinuities, there is an $\lambda'' < \lambda'$ such that for $\lambda'' < \lambda < \lambda'$ we have $s_i(\lambda) = u$ and $c_i(\lambda) = v$. Therefore, we also have $f_i(\lambda) = s_i(\lambda) - \lambda c_i(\lambda) = u - \lambda v$. Moreover, $f_i(\lambda)$ is continuous, so $f_i(\lambda') = \lim_{\delta \to 0} f_i(\lambda' - \delta) = u - \lambda' v$. □

5.3.3.2 Constructing the Pareto curve

We now use the quantities introduced above in order to traverse the Pareto curve. We start with $\lambda = +\infty$, for which the values of the $s_i(\lambda)$ and $c_i(\lambda)$ are easy to determine. Then, we iterate the following two steps (see Algorithm 20): (i) Use the current values of the $s_i(\lambda)$ and $c_i(\lambda)$ to find the next discontinuity. (ii) Update the $s_i(\lambda)$ and $c_i(\lambda)$ based on the change in the optimal support. In order to simplify the analysis, we assume that the discontinuities γ_i are distinct.

Theorem 5.13 establishes a connection between the variables s_j and c_j in FindPareto and the functions $s_j(\lambda)$ and $c_j(\lambda)$.

Theorem 5.13. *Let $s_l^{(i)}$, $c_l^{(i)}$, $active_l^{(i)}$, and $j^{(i)}$ be the values of s_l, c_l, $active_l$, and j after line 9 in iteration i of* FindPareto. *Then $\widehat{\lambda}_i = \max \gamma_l$, where $l \in [n]$ and $\gamma_l < \widehat{\lambda}_{i-1}$. Also, $\widehat{\lambda}_i = \gamma_{j^{(i)}}$. For $\widehat{\lambda}_{i-1} > \lambda \geq \widehat{\lambda}_i$, we have $s_l^{(i)} = s_l(\lambda)$ and $c_l^{(i)} = c_l(\lambda)$. Furthermore, $active_l = true$ if $b_l(\lambda) > 0$ and $active_l = false$ otherwise.*

Proof. We prove the theorem by induction over i. For $i = 1$, the statement of the theorem follows directly from the initialization of the variables in FindPareto.

Now assume that the theorem holds for a given $i > 1$. We need to show that the theorem also holds for $i + 1$.

Algorithm 20 (FindPareto) Constructing the Pareto curve

```
 1: function FindPareto($x, p$)
 2:     for $i \leftarrow 1, \ldots, n$ do                                   ▷ Initialization
 3:         $s_i \leftarrow |x_i|^p, \quad c_i \leftarrow 1, \quad \text{active}_i \leftarrow \text{false}$
 4:     end for
 5:     $\widehat{\lambda}_0 \leftarrow +\infty$
 6:     $r_1 \leftarrow c_1$
 7:     for $i = 1, \ldots, n$ do                                   ▷ Iterate over the discontinuities
 8:         $j \leftarrow \operatorname*{arg\,max}_{l \in [n], \, \text{active}_l = \text{false}} \frac{s_l}{c_l}$            ▷ Find the next discontinuity
 9:         $\widehat{\lambda}_i \leftarrow \frac{s_j}{c_j}$
10:         $\text{active}_j \leftarrow \text{true}$
11:         $a \leftarrow j$
12:         while $a \neq 1$ do                                   ▷ Update the affected nodes
13:             $a \leftarrow \text{parent}(a)$
14:             $s_a \leftarrow |x_a|^p$
15:             $c_a \leftarrow 1$
16:             for $l \in \text{children}(a)$ with $\text{active}_l = \text{true}$ do
17:                 $s_a \leftarrow s_a + s_l$
18:                 $c_a \leftarrow c_a + c_l$
19:             end for
20:         end while
21:         $r_{i+1} \leftarrow c_1$
22:     end for
23:     $\widehat{\lambda}_{n+1} \leftarrow 0$
24:     return $(\widehat{\lambda}, r)$
25: end function
```

Since $\widehat{\lambda}_i = \gamma_{j^{(i)}}$, we have $b_{j^{(i)}}(\lambda) > 0$ for $\lambda < \widehat{\lambda}_i$ and hence the update to $\text{active}_{j^{(i)}}$ is correct.

The inner update loop (lines 12 to 20) corresponds directly to the definition of $s_i(\lambda)$ and $c_i(\lambda)$, respectively. Hence we have

$$\begin{aligned} s_l^{(i+1)} &= \lim_{\delta \to 0^+} s_l(\widehat{\lambda}_i - \delta) \\ c_l^{(i+1)} &= \lim_{\delta \to 0^+} c_l(\widehat{\lambda}_i - \delta) \ . \end{aligned}$$

Note that we only have to update the nodes on the path from $j^{(i)}$ to the root because the other nodes are not affected by the discontinuity $\gamma_{j^{(i)}}$.

$s_l(\lambda)$ and $c_l(\lambda)$ are constant up to the next discontinuity given by

$$\gamma' = \max_{\substack{l \in [n] \\ \gamma_l < \widehat{\lambda}_i}} \gamma_l \ .$$

Let $\lambda' = \frac{\widehat{\lambda}_i + \gamma'}{2}$. Applying Lemma 5.11 with $\lambda = \lambda'$ to line 8 of FindPareto shows that $\gamma' = \widehat{\lambda}_{i+1}$ and $\widehat{\lambda}_{i+1} = \gamma_{j^{(i+1)}}$. □

Using this connection, we can now show that the algorithm returns the shape of the Pareto curve.

Theorem 5.14. *Let $p \geq 1$ and $x \in \mathbb{R}^n$ and let $\widehat{\lambda}$ and r be the vectors returned by* FindPareto(x, p)*. Moreover, let $\lambda > 0$ such that $\widehat{\lambda}_{i-1} > \lambda \geq \widehat{\lambda}_i$. Then we have $r_i = |\Omega^*_\lambda|$ where*

$$\Omega^*_\lambda = \operatorname*{arg\,max}_{\substack{\Omega \in \mathbb{T}_1,\ 1 \in \Omega \\ b_j(\lambda) > 0 \ \mathit{for}\ j \in \Omega \setminus \{1\}}} \|x_\Omega\|_p^p - \lambda|\Omega| \ .$$

Proof. By the definition of FindPath and Theorem 5.13, we have $r_i = c_1(\lambda)$ for $\widehat{\lambda}_{i-1} > \lambda \geq \widehat{\lambda}_i$. The theorem then follows from Lemma 5.9. □

Moreover, FindPareto can be implemented to run in $O(n \log n)$ time using a priority queue.

Theorem 5.15. *Let $p \geq 1$ and let $x \in R^n$ be the coefficients of a perfect d-ary tree. Then* FindPareto(x, p) *runs in time $O(n \log n)$ for constant d.*

Proof. Since we have a perfect d-ary tree, the depth of any node is bounded by $O(\log n)$. Hence the work of the inner update loop (lines 12 to 20) is bounded by $O(\log n)$ for a single iteration of the outer loop.

We implement the $\arg\max$ in line 8 with a Fibonacci heap containing the nodes j with $\text{active}_j = \text{false}$. Hence the cost of the $\arg\max$ is $O(\log n)$ and the cost of the inner update loop remains $O(\log n)$, now in amortized time.

As a result, the total time complexity of all n iterations is $O(n \log n)$. □

5.3.3.3 Tail approximation algorithm

Given the shape of the Pareto curve, we want to find the best solution achievable with our extended sparsity budget ck. To do so, we have to find a suitable trade-off parameter $\widehat{\lambda}$ that achieves a constant-factor tail approximation. We implement this search with a single scan over the $\widehat{\lambda}_i$, starting at $\widehat{\lambda}_n$ so that λ is increasing and the corresponding sparsity r_i decreasing. The main idea of this last claim is similar to our earlier (weakly polynomial) tail approximation algorithm. Algorithm 21 contains the pseudo code for this approach.

We first show that FindSolution allows us to reconstruct the support corresponding to a $\widehat{\lambda}_i$ and r_i.

Lemma 5.16. *Let $x \in \mathbb{R}^n$ be the coefficients corresponding to a d-ary tree and let $p \geq 1$. Then* FindSolution$(x, \widehat{\lambda}_i, p)$ *returns a support $\widehat{\Omega} \in \mathbb{T}_1$ satisfying*

$$\|x - x_{\widehat{\Omega}}\|_p^p + \lambda\left|\widehat{\Omega}\right| = \min_{\substack{\Omega \in \mathbb{T}_1 \\ \Omega \neq \{\}}} \|x - x_\Omega\|_p^p + \lambda|\Omega|$$

$$\left\|x_{\widehat{\Omega}}\right\|_p^p = s_1(\lambda)$$

$$\left|\widehat{\Omega}\right| = c_1(\lambda) = r_i$$

for $\widehat{\lambda}_{i-1} > \lambda \geq \widehat{\lambda}_i$. Moreover, FindTree *runs in linear time.*

Algorithm 21 (TailApprox) Tail approximation for the tree sparsity model

```
function TailApprox($x, k, c, p$)
    $(\widehat{\lambda}, r) \leftarrow$ FindPareto($x, p$)
    for $i \leftarrow n, \ldots, 1$ do
        if $r_i \leq ck$ then
            return FindSolution($x, \widehat{\lambda}_i, p$)
        end if
    end for
end function

function FindSolution($x, \lambda, p$)
    CalculateB($1, x, \lambda, p$)
    return $\widehat{\Omega} \leftarrow$ FindSupport($1$)
end function

function CalculateB($i, x, \lambda, p$)
    $\widehat{b}_i \leftarrow |x_i|^p - \lambda$
    for $j \in \text{children}(i)$ do
        CalculateB($j, x, \lambda, p$)
        $\widehat{b}_i \leftarrow \widehat{b}_i + \widehat{b}_j$
    end for
    $\widehat{b}_i \leftarrow \max(0, \widehat{b}_i)$
end function

function FindSupport($i$)
    $\Omega_i \leftarrow \{i\}$
    for $j \in \text{children}(i)$ do
        if $\widehat{b}_j > 0$ then
            $\Omega_i \leftarrow \Omega_i \cup$ FindSuppport($j$)
        end if
    end for
    return $\Omega_i$
end function
```

Proof. After the call to CalculateB, we have $\widehat{b}_j = b_j(\lambda_i)$ for $j \in [n]$ (see Lemma 5.7). Note that FindSupport follows the definition of $s_j(\lambda)$ and $c_j(\lambda)$. Using Lemma 5.9, we get

$$\widehat{\Omega} = \underset{\substack{\Omega \in \mathbb{T}_1,\, \Omega \neq \{\} \\ b_j(\lambda) > 0 \text{ for } j \in \Omega \setminus \{1\}}}{\arg\max} \; \|x_\Omega\|_p^p - \lambda|\Omega|$$

for $\widehat{\lambda}_{i-1} > \lambda \geq \widehat{\lambda}_i$. Lemma 5.9 also implies $\left\|x_{\widehat{\Omega}}\right\|_p^p = s_1(\lambda)$ and $\left|\widehat{\Omega}\right| = c_1(\lambda)$. Applying Theorem 5.14 then gives $\left|\widehat{\Omega}\right| = r_i$.

Negating the above objective function and using $\|x - x_\Omega\|_p^p = \|x\|_p^p - \|x_\Omega\|_p^p$, we get

$$\widehat{\Omega} = \underset{\substack{\Omega \in \mathbb{T}_1,\, \Omega \neq \{\} \\ b_j(\lambda) > 0 \text{ for } j \in \Omega \setminus \{1\}}}{\arg\min} \; \|x - x_\Omega\|_p^p + \lambda|\Omega| \; .$$

Finally, FindSolution makes a constant number of passes over the tree and consequently runs in time $O(n)$. □

We now prove the main result for the tail approximation algorithm.

Theorem 5.17. *Let $x \in \mathbb{R}^n$ be the coefficients corresponding to a d-ary tree rooted at node 1. Moreover, let $k \geq 1$, $c > 1$ and $p \geq 1$. Then* TailApprox(x, k, c, p) *returns a support $\widehat{\Omega} \in \mathbb{M}^+_{ck}$ satisfying*

$$\|x - x_{\widehat{\Omega}}\|_p \leq (1 + \frac{1}{c-1})^{1/p} \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p \, .$$

Furthermore, TailApprox *runs in time $O(n \log n)$.*

Proof. First, note that TailApprox always returns because $ck \geq 1 = r_1$. Moreover, the algorithm only returns if $r_i \leq ck$, so $\widehat{\Omega} \in \mathbb{M}^+_{ck}$ (Lemma 5.16).

We consider two cases based on $\left|\widehat{\Omega}\right|$. If $\left|\widehat{\Omega}\right| \geq k$, Lemma 5.16 implies

$$\begin{aligned} \|x - x_{\widehat{\Omega}}\|_p^p + \widehat{\lambda}_i \left|\widehat{\Omega}\right| &= \min_{\substack{\Omega \in \mathbb{T}_1 \\ \Omega \neq \{\}}} \|x - x_\Omega\|_p^p + \widehat{\lambda}_i |\Omega| \\ &\leq \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p^p + \widehat{\lambda}_i |\Omega| \, , \end{aligned}$$

where the last line uses $k \geq 1$. Since $\widehat{\lambda}_i \geq 0$ and $\left|\widehat{\Omega}\right| \geq k = |\Omega|$ for $\Omega \in \mathbb{M}_k$, we have

$$\|x - x_{\widehat{\Omega}}\|_p \leq \min_{\Omega \in \mathbb{M}_k} \|x - x_\Omega\|_p \, .$$

For the case of $\left|\widehat{\Omega}\right|$, let i be the final value of the loop counter in TailApprox. In order to establish an approximation guarantee for $\widehat{\Omega}$, we consider the support Ω' corresponding to r_{i+1}. By Theorem 5.14, this is

$$\Omega' = \operatorname*{arg\,max}_{\substack{\Omega \in \mathbb{T}_1, \, \Omega \neq \{\} \\ b_j(\lambda) > 0 \text{ for } j \in \Omega \setminus \{1\}}} \|x_\Omega\|_p^p - \lambda |\Omega|$$

for $\widehat{\lambda}_i > \lambda \geq \widehat{\lambda}_{i+1}$. Since the loop in TailApprox continued beyond r_{i+1}, we have $|\Omega'| = r_{i+1} > ck$.

Note that $s_1(\widehat{\lambda}_i) = \left\|x_{\widehat{\Omega}}\right\|_p^p$ and $c_1(\widehat{\lambda}_i) = \left|\widehat{\Omega}\right|$ (Lemma 5.16). Moreover, we have

$$\begin{aligned} \lim_{\delta \to 0^+} s_1(\widehat{\lambda}_i - \delta) &= \|x_{\Omega'}\|_p^p \\ \lim_{\delta \to 0^+} c_1(\widehat{\lambda}_i - \delta) &= |x_{\Omega'}| \, . \end{aligned}$$

Using Lemma 5.12 we get

$$\begin{aligned}\|x_{\Omega'}\|_p^p - \widehat{\lambda}_i|\Omega'| &= f_1(\widehat{\lambda}_i) \\ &= s_1(\widehat{\lambda}_i) - \widehat{\lambda}_i c_1(\widehat{\lambda}_i) \\ &= \left\|x_{\widehat{\Omega}}\right\|_p^p - \widehat{\lambda}_i\left|\widehat{\Omega}\right| .\end{aligned}$$

Equivalently, we have

$$\begin{aligned}\|x - x_{\Omega'}\|_p^p + \widehat{\lambda}_i|\Omega'| &= \|x - x_{\widehat{\Omega}}\|_p^p + \widehat{\lambda}_i\left|\widehat{\Omega}\right| \\ &= \min_{\substack{\Omega\in\mathbb{T}_1 \\ \Omega\neq\{\}}} \|x - x_\Omega\|_p^p + \widehat{\lambda}_i|\Omega| ,\end{aligned} \tag{5.8}$$

where the second line follows from Lemma 5.16. Now let $\Omega^* \in \mathbb{M}_k$ be a support with $\|x - x_{\Omega^*}\|_p = \min_{\Omega\in\mathbb{M}_k}\|x - x_\Omega\|_p$. Since $k \geq 1$, we have

$$\min_{\substack{\Omega\in\mathbb{T}_1 \\ \Omega\neq\{\}}} \|x - x_\Omega\|_p^p + \widehat{\lambda}_i|\Omega| \;\leq\; \|x - x_{\Omega^*}\|_p^p + \widehat{\lambda}_i|\Omega^*| . \tag{5.9}$$

Combining equations (5.8) and (5.9), we get

$$\begin{aligned}\|x - x_{\Omega'}\|_p^p + \widehat{\lambda}_i|\Omega'| &\leq \|x - x_{\Omega^*}\|_p^p + \widehat{\lambda}_i|\Omega^*| \\ \widehat{\lambda}_i(|\Omega'| - |\Omega^*|) &\leq \|x - x_{\Omega^*}\|_p^p - \|x - x_{\Omega'}\|_p^p \\ \widehat{\lambda}_i(ck - k) &\leq \|x - x_{\Omega^*}\|_p^p \\ \widehat{\lambda}_i &\leq \frac{\|x - x_{\Omega^*}\|_p^p}{k(c-1)} .\end{aligned}$$

We combine equations (5.8) and (5.9) again, this time for $\widehat{\Omega}$. Moreover, we use our new bound on $\widehat{\lambda}_i$.

$$\begin{aligned}\|x - x_{\widehat{\Omega}}\|_p^p + \widehat{\lambda}_i\left|\widehat{\Omega}\right| &\leq \|x - x_{\Omega^*}\|_p^p + \widehat{\lambda}_i|\Omega^*| \\ \|x - x_{\widehat{\Omega}}\|_p^p &\leq \|x - x_{\Omega^*}\|_p^p + \widehat{\lambda}_i k \\ &\leq \|x - x_{\Omega^*}\|_p^p + \frac{\|x - x_{\Omega^*}\|_p^p}{c-1} \\ &\leq \|x - x_{\Omega^*}\|_p^p \left(1 + \frac{1}{c-1}\right) .\end{aligned}$$

Taking the p-th root on both sides gives the guarantee in the theorem.

The running time bound for TailApprox follows directly from the time complexity of FindPareto and FindSolution. □

5.4 Compressive sensing recovery

As a concrete example for how to use our new approximate projections in an estimation problem, we instantiate our algorithms in a standard compressive sensing setup. Our example here is derived from the model-based CS framework proposed in [36] which uses exact projections.

Recall that the goal in compressive sensing is to estimate a vector $x \in \mathbb{R}^n$ from a small number of linear observations. These linear observation take the form

$$y \;=\; Ax + e \;,$$

where $A \in \mathbb{R}^{m \times n}$ is the measurement matrix and $e \in \mathbb{R}^n$ is a noise vector. The central result of compressive sensing is that under certain assumptions on the matrix A, it is possible to obtain a good estimate $\widehat{x}$ even though the matrix A is rank-deficient (and the system hence underdetermined).

Measurement matrices. Many algorithms for compressive sensing assume that the measurement matrix satisfies the *restricted isometry property* (RIP). The matrix A has the (δ, k)-RIP if the following inequalities hold for all k-sparse vectors $x \in \mathbb{R}^n$:

$$(1-\delta)\|x\|_2^2 \;\le\; \|Ax\|_2^2 \;\le\; (1+\delta)\|x\|_2^2 \;. \tag{5.10}$$

There exist measurement matrices satisfying the RIP with only $m = O(k \log \frac{n}{k})$ rows [35]. By using additional structure, we can improve over this bound. In particular, there exist matrices satisfying the RIP for tree sparse vectors with only $m = O(k)$ rows [36]. This logarithmically better sample complexity leads to clear improvements in numerical experiments (see Section 5.5).

Recovery algorithm. In order to use our approximate projections, we need a version of projected gradient descent that is robust to the approximation errors. We utilize our *approximate model-iterative hard thresholding* (AM-IHT) algorithm from Section 2.6.2. With the RIP assumption, this algorithm recovers a signal estimate $\widehat{x}$ satisfying

$$\|x - \widehat{x}\|_2 \;\le\; O(\|e\|_2) \;.$$

The algorithm repeatedly applies the following update rule, which is inspired by the well-known iterative hard thresholding (IHT) algorithm [50]:

$$x^{(i+1)} \leftarrow T(x^{(i)} + H(A^T(y - Ax^{(i)}))) \;. \tag{5.11}$$

It is possible to show that $O(\log \frac{\|x\|_2}{\|e\|_2})$ iterations suffice for guaranteed recovery. Therefore, the overall time complexity of AM-IHT is governed by the running times of $H(\cdot)$, $T(\cdot)$, and the cost of matrix-vector multiplication with A and A^T.

Putting things together. We can now instantiate AM-IHT with our new head and tail approximations. Recall that both approximate projection algorithms run in time $O(n \log n)$.

Publication	Measurement bound	Recovery time	Matrix-vector multiplication time	Recovery guarantee
[36]	$O(k)$	$O(nk)$	$O(nk)$	ℓ_2
[126]	$O(k\,\frac{\log n}{\log\log n})$	exponential	$O(n\log n)$	ℓ_1
[32]	$O(k\,\frac{\log n}{\log\log n})$	$O(nk)$	$O(n\log n)$	ℓ_1
This chapter	$O(k)$	$O(n\log n)$	$O(n\log n)$	ℓ_2

Table (5.2): Comparison of our results with previous recovery schemes for the tree-sparsity model. In order to simplify the presentation, all stated bounds are for the regime of $k \leq n^{1/2-\mu}$ with $\mu > 0$. We also omit a factor of $\log \frac{\|x\|_2}{\|e\|_2}$ from all recovery times. An ℓ_p-recovery guarantee is of the form $\|x - \widehat{x}\|_p \leq c\|e\|_p$, where x is the original signal, $\widehat{x}$ is the recovery result, e is the measurement noise, and c is a fixed constant.

For the overall running time, we also need a measurement matrix A that supports fast matrix-vector multiplication. In Section 2.6.6, we describe a new construction of such a matrix that satisfies the model-RIP for the tree-sparsity model $\mathcal{M}_k$ and in addition supports fast matrix-vector multiplication in the range $k \leq n^{1/2-\mu}$ with $\mu > 0$. Combining these ingredients with AM-IHT, we obtain an algorithm for recovering tree-sparse signals from (noisy) linear measurements.[1]

Theorem 5.18. *Let $A \in \mathbb{R}^{m\times n}$ be a model-RIP matrix as constructed in the proof of Theorem 2.29. Let $x \in \mathbb{R}^n$ be a signal with $x \in \mathcal{M}_k$ and let $y = Ax + e$ be the noisy measurements. Then, there exists an algorithm to recover a signal estimate $\widehat{x} \in \mathcal{M}_{c'k}$ from y such that*

$$\|x - \widehat{x}\|_2 \;\leq\; c\|e\|_2$$

for some constants $c, c' > 1$. The algorithm runs in time

$$O\Big((n\log n + k^2 \log n \log^2(k\log n)) \log \frac{\|x\|_2}{\|e\|_2}\Big)$$

for general k. This becomes time $O(n\log n)$ for the range $k \leq n^{1/2-\mu}$ with $\mu > 0$.

Note that this significantly improves over the time complexity of the original model-based compressive sensing framework [36], which required

$$O\Big(nk \log \frac{\|x\|_2}{\|e\|_2}\Big)$$

time for recovery. Moreover, our algorithm also improves over subsequenty work uses spase measurement matrices. We refer the reader to Table 5.2 for details.

[1] To be precise, the AM-IHT algorithm in Section 2.6.2 imposes additional restrictions on the approximation factors of the head and tail algorithms. However, it is possible to modify AM-IHT to work with arbitrary constant factors by "boosting" the head approximation ratio. See Section 2.6.5 for details.

Algorithm 22 (Tree-CoSaMP) Signal recovery

```
1: function Tree-CoSaMP($y, k, A, t$)
2:     $\widehat{x}_0 \leftarrow 0$
3:     for $i \leftarrow 1, \ldots, t$ do
4:         $v \leftarrow A^T(y - A\widehat{x}_{i-1})$
5:         $\Gamma \leftarrow \mathrm{supp}(\widehat{x}_{i-1}) \cup \textsc{TreeApprox}(v, 2k, c, 2, \delta)$
6:         $z_\Gamma \leftarrow A_\Gamma^\dagger y, \quad z_{\Gamma^C} \leftarrow 0$
7:         $\Omega \leftarrow \textsc{TreeApprox}(z, k, c, 2, \delta)$
8:         $\widehat{x}_i \leftarrow z_\Omega$
9:     end for
10:    return $\widehat{x} \leftarrow \widehat{x}_i$
11: end function
```

5.5 Numerical experiments

To complement our theoretical contributions, we now experimentally demonstrate the benefits of our approximate tree-projection algorithm in the context of compressive sensing. All experiments were conducted on a laptop computer equipped with an Intel Core i7 processor (2.66 GHz) and 8GB of RAM. Corresponding code is available on `http://people.csail.mit.edu/ludwigs/code.html`.

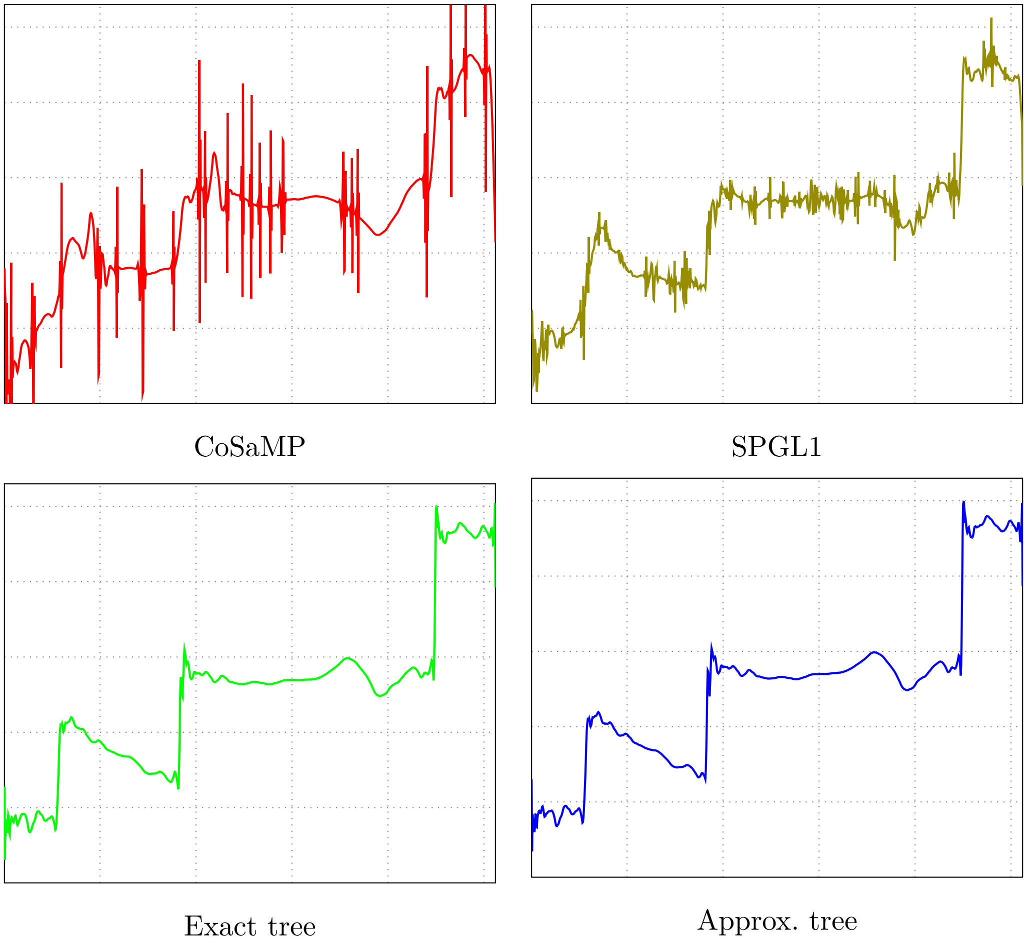
5.5.1 A modified algorithm

Since CoSaMP typically has better empirical performance than IHT, we use this variant of projected gradient descent for our numerical experiments. An approximation-tolerant version of the algorithm can be found in Section 2.6.3. We remark that it achieves the same theoretical guarantees as stated for AM-IHT in Section 5.4. The main difference to standard (model-)CoSaMP is that we replace the exact projections in Lines 5 and 7 with our approximate projection.

5.5.2 Results

First, we run model-CoSaMP with the exact tree projection approach in [64], as well as our Algorithm 22, on a piecewise polynomial signal. Such signals are well-modeled as tree-sparse in the wavelet domain [34]. The unknown vector (signal) is of length $n = 1024$ and is *exactly* tree-sparse with sparsity parameter $k = \lceil 0.04n \rceil = 41$. We record $m = \lceil 3.5k \rceil = 144$ random Gaussian measurements and perform CS recovery. For comparison, we also include recovery results using CoSaMP and ℓ_1-minimization (implemented by SPGL1).

As predicted by our theoretical results, Figure 5-2 demonstrates that our Algorithm 22 achieves accurate signal recovery, while standard CS approaches offer worse recovery quality. In fact, the performance of our algorithm is comparable to that of model-CoSaMP with exact projections. Figure 5-3 demonstrates a similar improvement in the case of a much larger signal. For this experiment, the input signal is a tree-sparse image of size $n = 512 \times 512$ with sparsity parameter $k = 0.04n \approx 10{,}000$, and we use $m = 3.3k \approx 35{,}000$ random Fourier measurements.



CoSaMP

SPGL1

Exact tree

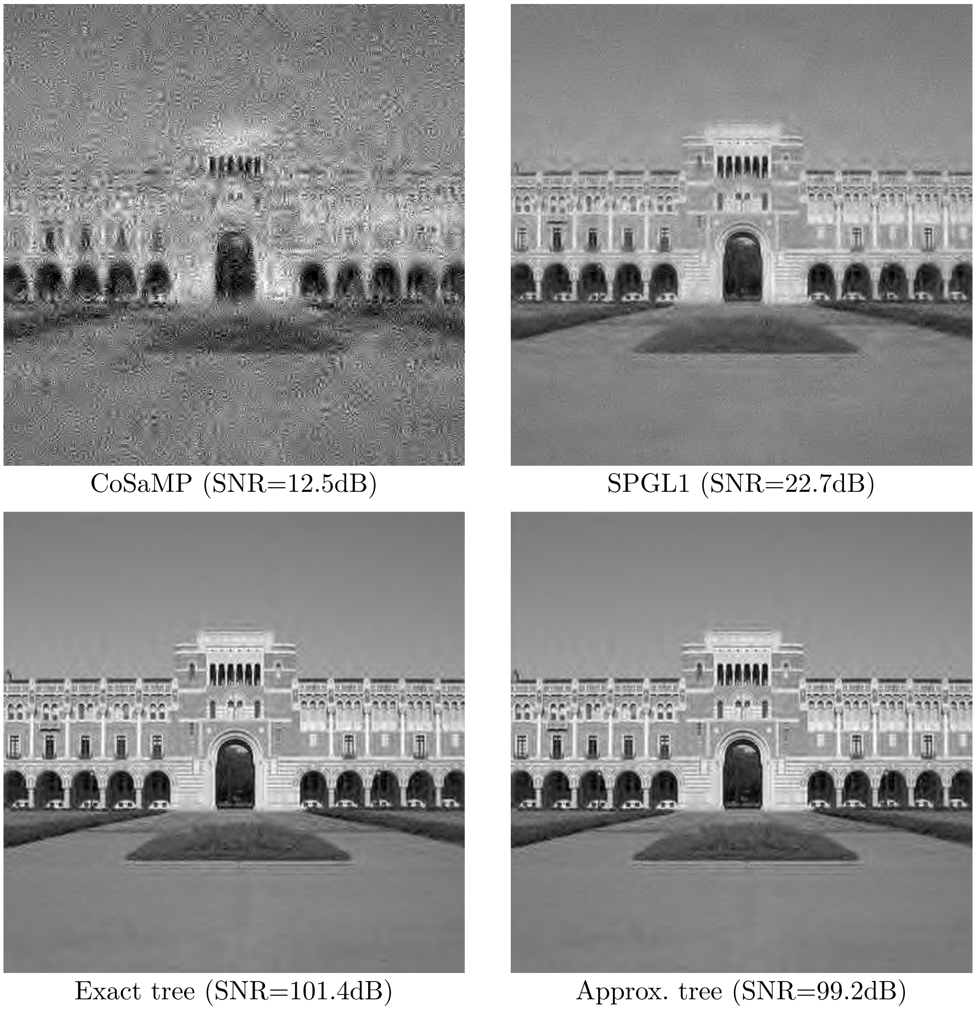
Approx. tree

Figure (5-2): Compressive sensing recovery of a 1D signal using various algorithms (signal parameters: $n = 1024$, $k = \lceil 0.04n \rceil = 41$, $m = \lceil 3.5k \rceil = 144$). Both tree-based algorithms accurately recover the ground truth signal.

Figure 5-4 plots the results of a Monte Carlo experiment that quantifies the performance of the different recovery algorithms in terms of the number of measurements m. The input test signal is a piecewise polynomial signal, similar to the one in Figure 5-2. Each data point in this plot was generated by averaging over 100 sample trials using different measurement matrices. For this plot, a "successful recovery" is defined as the event when the ℓ_2-error of the final signal estimate was within 5% of the ℓ_2-norm of the original signal. The success probability of Algorithm 22 almost matches the performance of model-CoSaMP with the exact tree-projection.

Table 5.3 demonstrates the computational efficiency of our tree-projection algorithm. Using the wavelet coefficients from Figure 5-3 as input, our tree-projection algorithm is more than two orders of magnitude faster than the exact tree-projection algorithm (400× speedup). Moreover, the tree-projection step is not a bottleneck in the overall recovery algorithm since the time spent on multiplying with A and A^T (at least one FFT each) dominates the runtime of our algorithm.

We also compare our algorithm with the greedy tree approximation algorithm described in [34]. While the greedy algorithm offers good recovery and runtime performance in the



CoSaMP (SNR=12.5dB)

SPGL1 (SNR=22.7dB)

Exact tree (SNR=101.4dB)

Approx. tree (SNR=99.2dB)

Figure (5-3): Compressive sensing recovery of a 2D image using various algorithms (signal parameters: $n = 512 \times 512$, $k = 0.04n \approx 10{,}000$, $m = 3.3k \approx 35{,}000$). Both tree-based algorithms accurately recover the ground truth image.

noiseless setting (see Figure 5-4 and Table 5.3), it is susceptible to shot noise (Figure 5-5). In contrast, our algorithm has rigorous approximation guarantees and demonstrates robust behavior similar to the exact tree-projection algorithm.

5.6 Algorithms for arbitrary trees

The algorithms from the previous sections run in nearly-linear time and suffice as head and tail approximations. However, they have three important shortcomings:

- The algorithms apply only to balanced trees.
- The algorithms increase the size of the output tree by a constant factor.
- The algorithms have constant approximation ratios (as opposed to tighter $1 + \varepsilon$ guarantees).

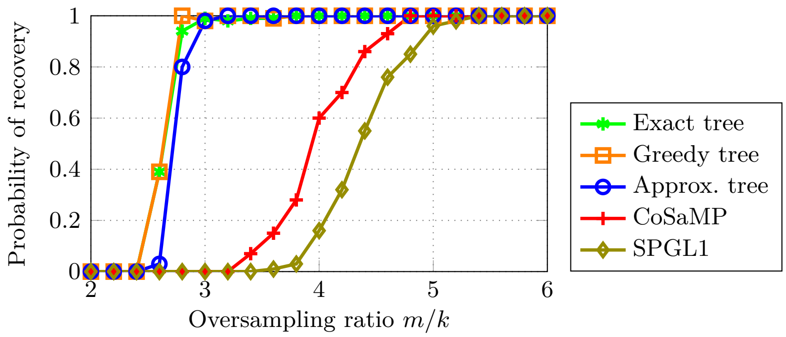


Figure (5-4): Comparison of compressiv sensing recovery algorithms. The probability of recovery is with respect to the measurement matrix and generated using 100 trial runs.

Algorithm	Exact tree	Approx. tree	Greedy tree	FFT
Runtime (sec)	4.4175	0.0109	0.0092	0.0075

Table (5.3): Runtimes of various algorithms on input data from Figure 5-3
(a single run of the projection algorithm without CoSaMP).
The times are averaged over 10 trials. $n \approx 260{,}000$, $k \approx 35{,}000$.

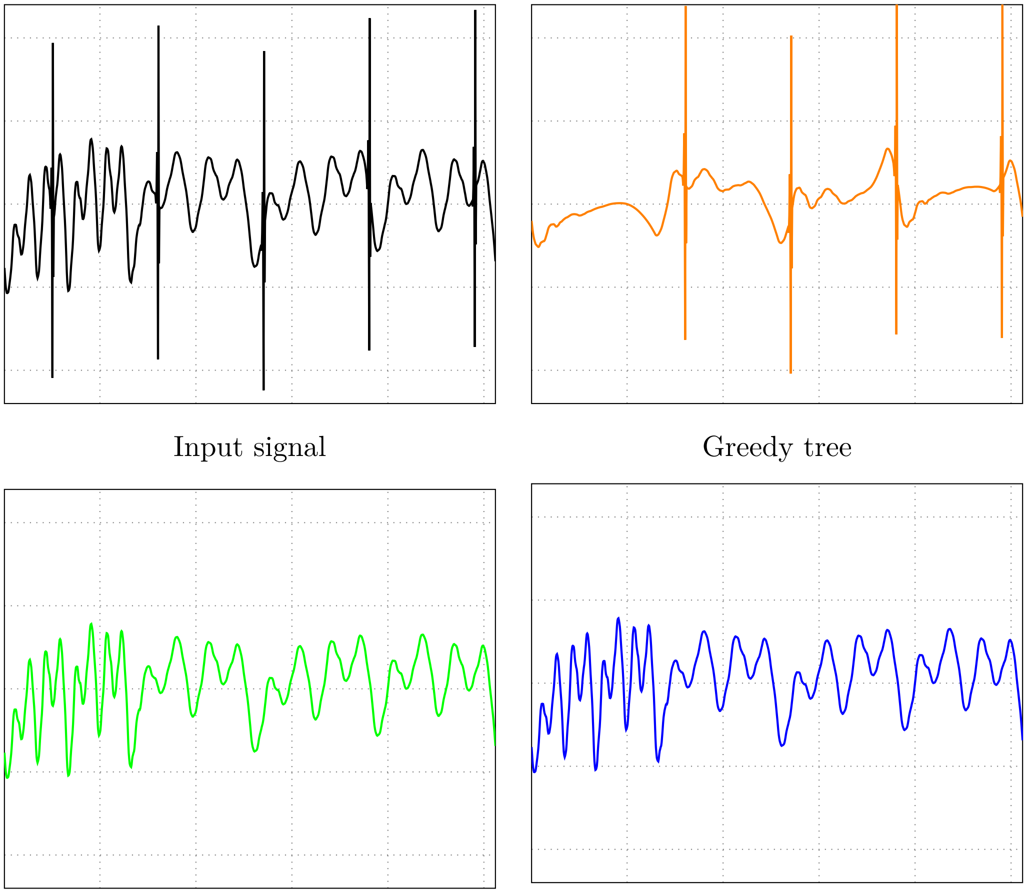
While our tail approximation works well in our numerical experiments, an exact algorithm is usually preferable if it is fast enough. For instance, exact algorithms tend to be more reliable and often require less tuning of hyperparameters. Hence it is important to understand whether the theoretical weaknesses mentioned above are necessary. To this end, we study the tree sparsity problem in more detail.

Our results. In this section, we contribute the following results:

- Our new head and tail projections run in nearly-linear time, do not increase the sparsity of the output, and work for arbitrary (binary) trees. Moreover, the approximation ratios of our algorithms are close to 1.
- We show that, for $k \approx n$, a strongly sub-quadratic time algorithm for tree sparsity would imply a strongly sub-quadratic time algorithm for $(\min, +)$-*convolution*. The latter is a well-studied problem (see e.g., [55]) for which no strongly sub-quadratic time algorithm is known. The problem bears strong similarity to the $(\min, +)$-matrix product, which has been used as the basis for several conditional hardness results [4, 5, 6, 29, 190, 237]. In particular, our reduction uses the techniques of [237].

From a theoretical perspective, our new approximation algorithms remove most drawbacks of our previous head / tail projections. Moreover, our conditional hardness result provides evidence that an exact tree projection might be impossible in time that is significantly faster than $O(nk)$.

From a practical perspective, a downside of our new algorithms is that their running time contains a large number of logarithmic factors. Hence we also give a simple algorithm for the head approximation problem whose running time $O(n \log n)$ has only a single logarithmic factor. The approximation factor of the algorithm is constant for trees whose depth is



Input signal

Greedy tree

Exact tree

Approx. tree

Figure (5-5): Tree projection for an input signal with shot noise ($n = 1024$). In order to simplify the example, we show the results of a single tree-projection without compressive sensing recovery. The greedy algorithm fails to find a good support while our approximation algorithm matches the output of the exact tree-projection algorithm.

a constant factor smaller than the sparsity k. We also show that a "boosted" variant of the algorithm increases the sparsity bound by a logarithmic factor while achieving an approximation ratio of 1.

Our techniques Our algorithms and lower bounds exploit the close connection between tree sparsity and convolutions. This can be illustrated by recalling the $O(kn)$-time exact algorithm for tree sparsity. The algorithm is based on dynamic programming. Specifically, for each node i and sparsity t, we compute $W[i, t]$, defined as the value of a sub-tree of size t, rooted at node i, and maximizing the total weight of the selected nodes. We compute this quantity recursively by selecting i and then, for an appropriate parameter r, selecting the maximum weight subtree of size r rooted in the left child of i, as well as the maximum weight subtree of size $t - r - 1$ rooted in the right child of i. Since we do not know the optimal value of r in advance, the algorithm enumerates all possible values. This leads to the following recursive formula:

$$W[i, t] = x_i + \max_r W[\mathit{left}(i), r] + W[\mathit{right}(i), t - r - 1] \tag{5.12}$$

where x_i denotes the weight of the node i.

Equation 5.12, when executed for all $t = 0 \ldots k$, can be seen as corresponding to the $(\max, +)$ convolution of a sequence

$$W[\mathit{left}(i), 0] \ldots W[\mathit{left}(i), t - 1]$$

and

$$W[\mathit{right}(i), 0] \ldots W[\mathit{right}(i), t - 1]$$

Our lower bound proceeds by showing that this correspondence is in some sense inherent, as we reduce $(\max, +)$ convolution to tree sparsity over a binary tree of depth $\Theta(k)$ with three long paths from the root to the leaves. In turn, our approximation algorithms are obtained by solving *approximate* $(\max, +)$ convolutions or $(\min, +)$-convolutions. Such approximations are known to be computable in nearly-linear time [178, 245]. For completeness, we include simpler (albeit slightly slower) algorithms performing those tasks in Section 5.6.6.

The first step in our approximation algorithms is to replace the exact convolution in Equation 5.12 by its approximate counterparts. However, this works only if the underlying tree has bounded (say, polylogarithmic) depth. This is because the approximation errors incurred at each level are cumulative, i.e., the final error bound is exponential in the tree depth. In order to make the algorithms applicable for trees of arbitrary depth, we must ensure that the error accumulates only a limited number of times. To this end, we utilize the *heavy-light decomposition* [215] of the underlying tree. On a high level, this approach decomposes the tree into a set of (possibly long) paths called *spines*. The decomposition ensures that there are at most $O(\log n)$ such spines on a path from the root to any leaf. We then design an algorithm that processes each spine in "one shot", as opposed to node by node. In this way, we ensure that the error is accumulated only a poly-logarithmic number of times.

5.6.1 Preliminaries

We start by setting up basic notation. We generally identify the node weights of a given binary tree with an n-dimensional vector $x \in \mathbb{R}^n$: each of the n coefficients in x corresponds to a node in the tree. Similarly, we identify a subtree with the corresponding *support* $\Omega \subseteq [n]$ in the vector x, i.e., the set of indices belonging to the subtree. We denote the set of supports forming valid rooted subtrees of size k with $\mathbb{T}_k$. For an arbitrary vector / tree x, we write x_Ω for the restriction of x to the support / subtree Ω: we have $(x_\Omega)_i = x_i$ if $i \in \Omega$ and $(x_\Omega) = 0$ otherwise. For a given support Ω, we denote its complement with $\overline{\Omega} = [n] \setminus \Omega$. Finally, the ℓ_1-norm of a vector x is $\|x\|_1 = \sum_{i=1}^n \|x_i\|$.

5.6.1.1 Head and tail approximation

With the new notation in place, we now formally state our main algorithmic problems. The "head" and "tail" terminology comes from the application of tree sparsity in sparse recovery [119]. In the following problems, the input is a tree with node weights $x \in \mathbb{R}^n$ and a sparsity parameter k.

Exact tree sparsity Find a subtree $\Omega \in \mathbb{T}_k$ such that $\|x - x_\Omega\|_1$ is minimized. Note that this is equivalent to maximizing $\|x_\Omega\|_1$.

Tail approximation Find a subtree $\Omega \in \mathbb{T}_k$ such that $\|x - x_\Omega\|_1 \leq c_T \cdot \min_{\Omega' \in \mathbb{T}_k} \|x - x_{\Omega'}\|_1$, where c_T is a constant (ideally close to 1).

Head approximation Find a subtree $\Omega \in \mathbb{T}_k$ such that $\|x_\Omega\|_1 \geq c_H \cdot \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1$, where c_H is a constant (ideally close to 1).

The tail approximation problem can be seen as an approximate minimization problem, while the head approximation problem is an approximate maximization problem. Note that a constant-factor tail approximation does not imply a constant-factor head approximation, and vice versa.

5.6.1.2 Generalized convolutions

Our algorithms build on sub-routines for various convolution problems, which we now review.

Definition 5.19 (($\otimes, \oplus$)-Convolution problem)**.** *Given two vectors $A = (A_0, \ldots, A_{n-1})^T \in \mathbb{Z}^n$ and $B = (B_0, \ldots, B_{n-1})^T \in \mathbb{Z}^n$, output the vector $C = (C_0, \ldots, C_{n-1})^T \in \mathbb{Z}^n$ defined as:*

$$C_k = \otimes_{i=0}^{k} (A_i \oplus B_{k-i}) \qquad \text{for all } k = 0, \ldots, n-1 \ .$$

We consider three instances of the general convolution problem: $(\min, +)$, $(\max, +)$, and $(\mathit{OR}, \mathit{AND})$-convolution. The latter is also called *Boolean Convolution*. Note that one can solve the Boolean Convolution problem in time $O(n \log n)$ using the Fast Fourier Transform on indicator vectors.

We will repeatedly use the following two theorems about $(\min, +)$ and $(\max, +)$-convolutions that we prove in Section 5.6.6.

Let $A, B \in \mathbb{N}^n$ be two integer vectors with positive entries. Let W be the largest value of an entry in A or B. Let $C \in \mathbb{N}^n$ be the $(\max, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1+\varepsilon)C'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W\right).$$

An analogous statement holds if C is the $(\min, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1+\varepsilon)C'_k$ for all $k = 0, \ldots, n-1$ in the same runtime.

An slightly faster algorithm for $(\min, +)$ matrix products was shown by Uri Zwick [245]. We also note than a different algorithm for approximating $(\max, +)$-convolution was given in [204], but no theoretical guarantees on the approximation factor were provided.

Let $A^1 \in \mathbb{N}^{n_1}, \ldots, A^l \in \mathbb{N}^{n_l}$ be $l \geq 2$ vectors with positive entries. We assume that the entries of the vectors are indexed starting with 0, i.e., $A^i = \left(A^i_0, \ldots, A^i_{n_i-1}\right)^T$ for every $i = 1, \ldots, l$. Let $B \in \mathbb{N}^n$, $n = n_1 + \ldots + n_l$ be the $(\min, +)$-convolution between the l vectors $A^1, \ldots, A^{n_l}$ defined as follows:

$$B_j = \min_{\substack{m_1 + \ldots + m_l = j \\ \text{s.t. } 0 \leq m_t \leq n_t - 1 \ \forall t = 1, \ldots, l}} \left(A^1_{m_1} + \ldots + A^l_{m_l}\right)$$

for every $j = 0, \ldots, n-1$. We can output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W \cdot \log^3 l\right),$$

where W denotes the largest entry in the vectors $A^1, \ldots, A^l$.

An analogous statement holds for $(\max, +)$-convolution (we replace min in the equation for B_j with max). The runtime stays the same and we output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$.

5.6.2 $1+\varepsilon$ tail approximation in nearly linear time

Our algorithm utilizes a heavy-light decomposition [215], which we define first.

Definition 5.20 (Heavy-light decomposition [215])**.** *Let x be a binary tree, and let node v be a child of node u (the parent). We call an edge $e = (u, v)$* heavy *if one of the following holds:*

- *$\mathit{size}(v) > \frac{\mathit{size}(u)-1}{2}$;*
- *$\mathit{size}(v) = \frac{\mathit{size}(u)-1}{2}$ and v is the right child of the parent u.*

We call all other edges light*.*

We extend the above definition of light and heavy edges to nodes as follows: If a parent u is connected to a child v by a heavy edge, we call v heavy. We can easily check that every node has at most one heavy child and that only leaves can have no heavy child. We call a node *special* if it is not a heavy child of its parent. Special nodes are relevant for our definition of a *spine*:

Definition 5.21 (Spine)**.** *Consider the following process. Pick any special node u. If u is not a leaf, it has a unique heavy child. Choose this child and repeat. We will end up in a leaf. This defines a path in the tree corresponding to the node u. We call this path the spine corresponding to the node u. The spine includes the node u and the final leaf.*

By iterating over all special nodes, we can decompose the tree into a set of spines. This decomposition has desirable properties because we can only have a small number of spines on any path from a leaf to the root. More formally, consider any node u in the tree. The number of light edges on the path from the node u to the root of the tree is upper bounded by $O(\log n)$. The reason is that every light edge at least halves the size of the subtree and the number of nodes in the tree is upper bounded by n. Since every two light edges on the path are separated by one spine, the number of different spines on the path from the node u to the root is also upper bounded by $O(\log n)$.

With our definition of a spine set up, we now proceed to our main result of this section.

Theorem 5.22. *Given a vector x and an integer $k \geq 1$, we can find a subtree $\Omega \in \mathbb{T}_k$ such that*

$$\|x_{\overline{\Omega}}\|_1 \;\leq\; (1+\varepsilon) \cdot \|x_{\overline{\Omega^*_k}}\|_1$$

where

$$\Omega^*_k \;=\; \operatorname*{arg\,min}_{\Omega' \in \mathbb{T}_k} \|x_{\overline{\Omega'}}\|_1 \; .$$

Moreover, the algorithm runs in time $O(\frac{n}{\varepsilon^3} \cdot \log^9 n \cdot \log^3 x_{\max})$.

Proof. For a tree x of size $n \geq 0$, we define the *tail tree sparsity vector* to be the vector

$$\left(\|x_{\overline{\Omega^*_0}}\|_1, \|x_{\overline{\Omega^*_1}}\|_1, \ldots, \|x_{\overline{\Omega^*_n}}\|_1\right)^T \in \mathbb{N}^{n+1}.$$

To simplify the exposition, in this section we will skip the word “tail”.

We will recursively compute approximate tree sparsity vectors for all subtrees rooted at the special nodes of the tree x. As we compute these approximations, the sizes of the subtrees become smaller and the lengths of the corresponding approximate sparsity vectors (the number of entries in the vectors) also become smaller. To compute the approximate sparsity vector for a special note u, we need to know the approximate sparsity vectors for all special nodes in the subtree rooted at u. Without loss of generality, we assume that for every leaf l, we have $x_l > 0$.

Our recursive algorithm works as follows.

- For an integer $l \geq 1$, let $p_1, p_2, \ldots, p_l$ be the spine that starts at the root p_1 of the tree x and ends in the leaf p_l of the tree. For simplicity, we renumber vertices of the tree x so that the spine consists of vertices $1, \ldots, l$ in this order (1 is the root of the tree and l is the leaf).
- For $i = 1, \ldots, l-1$, the node i has $i+1$ as a child. Let x^i denote the subtree rooted at the other child node (recall that x is a binary tree). Let $n_i \geq 0$ denote the size of the subtree x^i. Note that n_i can be zero if node i has only one child, i.e., the node $i+1$.

For simplicity of exposition, let x^l be an empty tree of size $n_l := 0$. It corresponds to a child of leaf l. Since leaf l has no children, x^l is of size 0. Let

$$s^i := \left(s_0^i, s_1^i, s_2^i, \ldots, s_{n_i}^i\right)^T \in \mathbb{N}^{n_i+1}$$

be the approximate sparsity vector for the subtree x^i computed recursively. We assume that s^i is an approximate sparsity vector in the following sense. Let $\hat{s}^i$ be the (exact) sparsity vector for the subtree x^i. Then $\hat{s}_j^i \leq s_j^i \leq (1+\delta)\hat{s}_j^i$ for some $\delta > 0$ and for all $j = 0, 1, \ldots, n_i$.

- For a node $i = 1, \ldots, l$, let w_i denote the total weight of the subtree rooted at i. We define a set L as follows. Initially $L = \{l\}$. We set $l_{\min} = l$. While $l_{\min} > 1$, choose the smallest $l' \geq 1$ such that $w_{l'+1} \leq (1+\delta')w_{l_{\min}}$ for some $\delta' > 0$ that we will choose later. Then we add l' to L, set $l_{\min} = l'$, and repeat. We note that the size of the set L is upper bounded by $|L| \leq \log_{1+\delta'}(nx_{\max}) \leq O(\log(nx_{\max})/\delta')$.

  We need the set L for the following reason: We want to approximate the sparsity vector for the tree x. Fix an arbitrary sparsity $t \geq 1$ that we want to approximate. The optimal tree will pick a number $l'' \geq 1$ of nodes from the spine and rooted subtrees from the trees $x^1, \ldots, x^{l''}$. The main idea is that l'' is as good as l''' for some $l''' \in L$ up to a factor $(1+\delta')$. That is, we can assume that $l'' \in L$ and we loose at most a factor of $(1+\delta')$ in the approximation ratio (we make this precise in the analysis below). This implies that it is sufficient to compute the sparsity vectors for all different $l'' \in L$, i.e., assuming that we have to pick the first l'' nodes from the spine. In particular, we do *not* require to compute sparsity vectors for all values in $[l]$, which leads to an important speed-up and enables our algorithm to run in nearly-linear time. To get an approximation for sparsity t, we take the minimum tail (for sparsity t) that we achieve over all the computed sparsity vectors for different $l'' \in L$.

- For every $l'' \in L$, we compute a sparsity vector $r^{l''} \in \mathbb{N}^{n+1}$. Let t be the sparsity that we want to approximate. We want to pick up the first l'' nodes from the spine. We also have to pick up $t - l''$ nodes from the trees $x^1, \ldots, x^{l''}$. Therefore, we set

$$r_t^{l''} := x_1 + \ldots + x_{l''} + \min_{\substack{m_1, \ldots, m_{l''} \geq 0 \\ \text{s.t. } m_1 + \ldots + m_{l''} = t - l''}} \left(s_{m_1}^1 + \ldots + s_{m_{l''}}^{l''}\right).$$

  It remains to approximately compute $\min\left(s_{m_1}^1 + \ldots + s_{m_{l''}}^{l''}\right)$. This is exactly the problem stated in Theorem 5.6.1.2. We run the corresponding algorithm with approximation factor $(1+\delta')$. The approximate sparsity vector $r \in \mathbb{N}^{n+1}$ for the tree x is then computed as $r_t := \min_{l'' \in L} r_t^{l''}$ for all $t = 0, 1, \ldots, n$.

Correctness of the algorithm There are three sources of error in one recursive call of the algorithm. First, we do not have the exact sparsity vectors for the subtrees x^i. Instead, we have $(1+\delta)$-approximate sparsity vectors. This introduces $1+\delta$ multiplicative error in our approximation of the sparsity vector for the tree x. Second, we will show below that working with the prefixes of the spine of length $l'' \in L$ introduces a multiplicative error of $(1+\delta')$. Finally, since we perform an *approximate* (min, +)-convolution in the final step of the recursive call, we get additional multiplicative error of $(1+\delta')$. Therefore, the multiplicative step gives an approximate sparsity vector for the tree x with error $(1+\delta)(1+\delta')^2$. From Definition 5.20, we know that the number of different spines on any path to the root is upper bounded by $O(\log n)$. This implies that the recursive depth of the algorithm is $O(\log n)$, which leads to a final approximation error of $(1+\delta')^{O(\log n)}$. Choosing $\delta' = \Theta\left(\frac{\varepsilon}{\log n}\right)$ gives the promised $(1+\varepsilon)$ approximation factor.

It remains to show that working with the prefixes of the spine of length $l'' \in L$ introduces a multiplicative error of at most $(1+\delta')$. Fix an arbitrary sparsity $t = 0, \ldots, n$. Let l' be the number of nodes from the spine that an optimal subtree (with the smallest possible tail error) of size t picks up. We assume that $l' \notin L$ since otherwise we compute an exact (sub-)solution. We have that $l''_1 < l' < l''_2$ for some $l''_1, l''_2 \in L$ with $w_{l''_1+1} \leq (1+\delta')w_{l''_2}$ by the construction of the set L. Let Ω^* be the support of the optimal subtree (Ω^* picks up l' nodes from the spine). Since $l' < l''_2$, we have

$$w_{l''_2} \leq \|x_{\overline{\Omega^*}}\|_1. \tag{5.13}$$

Let Ω_t be Ω^* but with all the nodes from the subtree rooted at $l''_1 + 1$ removed from it. Then Ω_t picks up l''_1 nodes from the spine as required. We have to show that we did not increase the tail too much by removing vertices from the support Ω^* to get the support Ω_t. To this end, we observe that we increased the tail by at most $w_{l''_1+1} - w_{l''_2}$, and therefore:

$$\begin{aligned}
\|x_{\overline{\Omega_t}}\|_1 &\leq \|x_{\overline{\Omega^*}}\|_1 + w_{l''_1+1} - w_{l''_2} \\
&\leq \|x_{\overline{\Omega^*}}\|_1 + \delta' w_{l''_2} \\
&\leq (1+\delta')\|x_{\overline{\Omega^*}}\|_1,
\end{aligned}$$

where we use (5.13) in the last inequality.

The runtime of the algorithm There are $S_1 := O(\log n)$ levels of recursion in the algorithm. In each level, we perform $|L| \leq O(\log(nx_{\max})/\delta') =: S_2$ approximate $(\min,+)$-convolutions between multiple sequences with approximation factor $(1+\delta')$. From Theorem 5.6.1.2, we know that one $(\min,+)$-convolution takes $S_3 := O\left(\frac{n}{\delta'^2} \cdot \log^4 n \cdot \log^2 x_{\max}\right)$ time. Since we chose $\delta' = \left(\frac{\varepsilon}{\log n}\right)$, we get the final runtime stated in the theorem:

$$S_1 \cdot S_2 \cdot S_3 \leq O\left(\frac{n}{\varepsilon^3} \cdot \log^9 n \cdot \log^3 x_{\max}\right)$$

□

5.6.3 $1+\varepsilon$ head approximation in nearly linear time

In this section, we prove the following theorem.

Theorem 5.23. *Given a vector x and an integer $k \geq 1$, we can find a subtree $\Omega \in \mathbb{T}_k$ such that*

$$\|x_\Omega\|_1 \;\geq\; \|x_{\Omega^*_k}\|_1/(1+\varepsilon)$$

where

$$\Omega_k^* \;=\; \operatorname*{arg\,max}_{\Omega_k \in \mathbb{T}_k} \|x_{\Omega_k}\|_1 \;.$$

Moreover, the algorithm runs in time $O(\frac{n}{\varepsilon^2} \cdot \log^{12} n \cdot \log^2 x_{\max})$.

We solve the problem in two attempts. In the first attempt, we construct an algorithm that runs in the required nearly-linear time but the approximation error is too large. This will allow us to introduce ideas used in the second (and final) algorithm that achieves both the desired running time and approximation factor.

5.6.3.1 First attempt to solve the problem

For a tree x of size $n \geq 0$, we define the *head tree sparsity vector* to be the vector

$$\left(\|x_{\Omega_1^*}\|_1, \ldots, \|x_{\Omega_n^*}\|_1\right)^T \in \mathbb{N}^n.$$

Notice that this definition is different from the *tail* tree sparsity vector in Section 5.6.2. To simplify the exposition, we omit the word "head".

Our plan is to compute an approximation to the tree sparsity vector for every tree rooted at a special node, i.e., for every tree rooted at the first node of every spine. Since the root of the tree is a special node, this will also solve our original problem.

Similarly to the previous section, we perform this computation recursively.

- For an integer $l \geq 1$, let $p_1, p_2, \ldots, p_l$ be the spine that starts at the root p_1 of the tree x and ends in a leaf (p_l) of the tree. To simplify notation, we renumber the vertices of the tree x so that the spine consists of vertices $1, \ldots, l$ in this order.

- For $i = 1, \ldots, l-1$, node i has child node $i+1$. Let x^i denote the subtree rooted at the other child node. Let $n_i \geq 0$ denote the size of the subtree x^i. To simplify notation, let x^l be an empty tree of size $n_l := 0$. It corresponds to a child of leaf l. Let

  $$s^i := \left(s_1^i, s_2^i, \ldots, s_{n_i}^i\right)^T \in \mathbb{N}^{n_i}$$

  be the approximate sparsity vector for the subtree x^i computed recursively. We assume that s^i is a $(1+\delta)$-approximate sparsity vector in the following sense. Let $\hat{s}^i$ be the (exact) sparsity vector for the subtree x^i. Then $\hat{s}_j^i/(1+\delta) \leq s_j^i \leq \hat{s}_j^i$ for all $j = 1, \ldots, n_i$. We will determine $\delta > 0$ later.

- Let $t = (t_1, \ldots, t_m)^T \in \mathbb{N}^m$ be the approximate sparsity vector corresponding to a tree of size $m \geq 0$. We would like to represent t with much fewer than m entries so that we can work with the sparsity vector faster. For this, we define the *compressed sparsity vector* $C(t) \in \mathbb{N}^{\log_{1+\delta'}(m x_{\max})}$ as follows.[1] For $j = 0, \ldots, \log_{1+\delta'}(m x_{\max}) - 1$, we set $C(t)_j$ to the minimum $j' \geq 1$ such that $t_{j'} \geq (1+\delta')^j$. Notice that the sparsity vector is non-decreasing (that is, $t_k \leq t_{k+1}$ for every $k = 1, \ldots, m-1$). Intuitively, $C(t)$ stores the indices of the sparsity vector t where the value changes by a lot, i.e., by at least a factor of $1+\delta'$. We decreased the number of entries from m to

[1] We will choose $\delta' > 0$ later. $x_{\max}$ upper bounds the maximum value in the vector t.

$\log_{1+\delta'}(mx_{\max}) \leq O(\log(mx_{\max})/\delta')$ by using $C(t)$ instead of t. Note that, given $C(t)$, we can recover vector $r = (r_1, \ldots, r_m)^T \in \mathbb{N}^m$ such that $t_j/(1+\delta') \leq r_j \leq t_j$ for every $j \in [m]$. We do this as follows: we set $r_j := \max_{j':C(t)_{j'} \leq j}(1+\delta')^{j'}$ for every $j \in [m]$.

- For every $i \in 1 \ldots l$, we compute the compressed sparsity vector $C(s^i)$ corresponding to the subtree x^i. Then, for every $i = l, l-1, \ldots, 1$ (in this order), we compute the compressed sparsity vector c^i for the tree rooted at node i. To compute c^i we need $C(s^i)$ and c^{i-1}. Given $C(s^i)$ and c^{i-1}, we can first compute approximate sparsity vectors corresponding to the tree x^i and to the tree rooted at node $i-1$ as described in the definition of compressed sparsity vectors above. Then we could use $(\max, +)$-convolution to compute the approximate sparsity pattern for the tree rooted at node i from which we can obtain c^i. However, the time complexity of this approach is too large. Instead, given c^{i-1} and $C(s^i)$, we can compute c^i directly in time $O(\log^2(nx_{\max})/\delta'^2)$. We give more details below.

  We output c^1, which is the compressed sparsity vector corresponding to the input tree x.

Why this approach does not quite work Consider the stage of the algorithm when we are given compressed sparsity vectors c^{i-1} and $C(s^i)$ and we compute c^i for some fixed $i = 1, \ldots, l$. To compute c^i, we can (implicitly) compute an approximate sparsity vector a corresponding to c^{i-1} and an approximate sparsity vector b corresponding to $C(s^i)$ as described in the definition of the compressed sparsity vectors. The vectors a and b consists of at most $O(\log(nx_{\max})/\delta')$ different entries. It is not hard to see that the approximate sparsity vector for the tree rooted at node i consists of at most $O(\log^2(nx_{\max})/\delta'^2)$ different values (it is essentially a $(\max, +)$-product of a and b). We obtain c^i from the resulting approximate sparsity vector as in the definition of the compressed sparsity vectors. Again, it is easy to check that this can be done implicitly in the stated runtime. As observed in the definition, the step of obtaining the compressed sparsity vector c^i introduces a $1+\delta'$ multiplicative error because we round the values of the vector to an integer power of $1+\delta'$. Since we compute l compressed sparsity vectors $c^l, c^{l-1}, \ldots, c^1$, the total error that we collect is $(1+\delta')^l$. Since we want the final error to be small (at most $1+\varepsilon$), and l can be large (as large as $\Omega(n)$), we have to choose $\delta' = O(1/n)$. This is prohibitively small because even the size of the compressed sparsity vectors becomes $\Omega(n)$.

5.6.3.2 Second attempt to solve the problem

In our first attempt, we constructed an algorithm in which we had to choose δ' to be too very small. In this section, we change the algorithm in a way that will allow us to pick δ' to be much larger. The resulting algorithm then runs in the promised time complexity and achieves a $1+\varepsilon$ approximation. The algorithm stays the same as in attempt one, except for the last (fourth) step. We now describe how to modify this last step.

W.l.o.g. we assume that l is an integer power of 2. We can do that since otherwise we can add $l - 2^j\lfloor l/2^j \rfloor$ nodes to the spine with the corresponding values equal to 0. For every integer $j \geq 0$ such that $2^j \leq l$, we split the nodes $1, 2, \ldots, l$ on the spine into $l/2^j$ groups, each containing 2^j nodes. For $y = 1, \ldots, l/2^j$, the y-th group consists of 2^j nodes $(y-1)2^j + 1, (y-1)2^j + 2, \ldots, y2^j$. For a fixed $j \geq 0$ and for a fixed group y, we want to compute an approximate sparsity vector $r^{j,y}$ corresponding to subtrees in which we choose only nodes $(y-1)2^j+1, (y-1)2^j+2, \ldots, y2^j$ from the spine. Namely, we want to compute

$$\begin{aligned} r_t^{j,y} :=& x_{(y-1)2^j+1} + \ldots + x_{y2^j} \\ &+ \max_{\substack{m_1,\ldots,m_{2^j} \geq 0 \\ \text{s.t. } m_1+\ldots+m_{2^j}=t-2^j}} \left(s_{m_1}^{(y-1)2^j+1} + \ldots + s_{m_{2^j}}^{y2^j} \right). \end{aligned}$$

for all $t = 0, 1, \ldots, 2^j + n_{(y-1)2^j+1} + n_{(y-1)2^j+2} + \ldots + n_{y2^j}$. To approximately compute the quantity $\max \left(s_{m_1}^{(y-1)2^j+1} + \ldots + s_{m_{2^j}}^{y2^j} \right)$, we run the algorithm from Theorem 5.6.1.2 with approximation factor $1+\delta'$. Let $n^{j,y} := 2^j + n_{(y-1)2^j+1} + \ldots + n_{y2^j}$. The runtime for $(1+\delta')$-approximately computing $r^{j,y}$ is

$$O\left(\frac{n^{j,y}}{\delta'^2} \cdot \log^6 n \cdot \log^2 x_{\max} \right)$$

by Theorem 5.6.1.2. When using Theorem 5.6.1.2 we note that the largest value in vectors s^i can be $\Omega(nW)$. Since $n^{j,1} + \ldots + n^{j,l/2^j} \leq n$, we get that the total runtime for computing the approximations $\hat{r}^{j,y}$ for all vectors $r^{j,y}$ is upper bounded by

$$O\left(\frac{n}{\delta'^2} \cdot \log^7 n \cdot \log^2 x_{\max} \right). \tag{5.14}$$

Now we will describe how to use the vectors $\hat{r}^{j,y}$ to get an approximation to the sparsity vector of the tree x. We start with computing the compressed sparsity vectors $C(\hat{r}^{j,y})$ for all j and y.

For every $i = 1, 2, \ldots, l$, we consider trees in which we choose nodes $1, 2, \ldots, i$ from the spine and we do not choose the node $i+1$. For such trees, we compute the compressed sparsity vector v^i. We want to use at most $O(\log n)$ compressed sparsity vectors $C(\hat{r}^{j_0,y_0}), C(\hat{r}^{j_1,y_1}), C(\hat{r}^{j_2,y_2}), C(\hat{r}^{j_3,y_3}), \ldots$ to compute v^i. We choose the pairs j_p, y_p according to the binary expansion of i: we choose different integers j_p such that $i = 2^{j_0} + 2^{j_1} + 2^{j_2} + 2^{j_3} + \ldots$. We also choose the groups y_p so that different groups do not share nodes and together the groups cover all elements $1, 2, \ldots, i$ from the spine. To compute v^i, we could compute the $(\max, +)$-convolution of the vectors $\hat{r}^{j_0,y_0}, \hat{r}^{j_1,y_1}, \hat{r}^{j_2,y_2}, \ldots$ from which we can obtain the compressed vector v^i. However, this would take too much time. Instead, we observed in our previous attempt that we can compute v^i directly from the compressed vectors $C(r^{j_p,y_p})$. Thus, the compressed sparsity vector v^i can be computed in time $O(\log n \cdot \log^2(nx_{\max})/\delta'^2)$.

Now we have l compressed sparsity vectors $v^1, v^2, \ldots, v^l$. To compute the sparsity vector for the tree x we could do the following: get sparsity vectors corresponding to vectors $v^1, v^2, \ldots, v^l$ and compute entry-wise minimum of the l vectors. This has too large time complexity and instead we compute the answer by computing the entry-wise minimum without explicitly computing the sparsity vectors. More precisely, let $v' \in \mathbb{N}^n$ be a vector consisting of only $+\infty$ initially. For every $i \in [l]$ and for every entry v_j^i, we set $v'_{v_j^i}$ to be equal to $\min(v'_{v_j^i}, j)$. We set $z = 1$. For every $j = 1, \ldots, n$ in this order, we set $v_j := (1+\delta')^z$ and, if $v'_j \neq +\infty$, we update $z = \max(z, v'_j)$. We output v as the sparsity vector for the tree

x.

The approximation factor of the algorithm We assume that the vectors s^i (that we compute recursively) are $(1+\delta)$-approximations to the exact sparsity vectors. We compute the vectors $\hat{r}^{j,y}$ that introduce another $1+\delta'$ multiplicative error in the approximation. We then get compressed sparsity vectors $C(\hat{r}^{j,y})$, which gives another multiplicative error factor $1+\delta'$. To compute every v^i, we need to compute a $(\max,+)$-convolution between compressed sparsity vectors $O(\log n)$ times, which gives $(1+\delta')^{O(\log n)}$ error. The total error from the recursive call is bounded by

$$(1+\delta)\cdot(1+\delta')\cdot(1+\delta')\cdot(1+\delta')^{O(\log n)}$$
$$=(1+\delta)\cdot(1+\delta')^{O(\log n)}\ .$$

Since the depth of the recursion is $O(\log n)$ (from the definition of the heavy-light decomposition), the error of the algorithm is $(1+\delta')^{O(\log^2 n)}$. To make it smaller than $1+\varepsilon$, we set $\delta'=\Theta(\varepsilon/\log^2 n)$.

The runtime of the algorithm The runtime of the recursive step is dominated by computing the vectors $\hat{r}^{j,y}$. Plugging $\delta'=\Theta(\varepsilon/\log^2 n)$ into (5.14), we get that the runtime of the recursive step is

$$O(\frac{n}{\varepsilon^2}\cdot\log^{11} n\cdot\log^2 x_{\max})\ .$$

Since the depth of the recursion of the algorithm is $O(\log n)$, the final runtime is

$$O(\frac{n}{\varepsilon^2}\cdot\log^{12} n\cdot\log^2 x_{\max})\ .$$

5.6.4 $(\max,+)$-Convolution hardness for the tree sparsity problem

In this section is we provide an evidence that the exact tree sparsity requires nearly quadratic time. Recall that, given a binary node-weighted tree x of size m, we want to output the largest sum of weights of nodes of x that we can pick up by choosing a rooted subtree of x of size $k\le m$. The best known algorithm for this problem runs in time $\Theta(m^2)$ (see Section 5.6.7). In this section we show that this problem cannot be solved in a strongly subquadratic time unless the $(\max,+)$-convolution problem can be solved in a strongly subquadratic time. This is a well studied problem [55, 68] which is known to be at least as hard as the *polyhedral 3SUM* problem[1] [55].

Theorem 5.24. *Let x be a binary node-weighted tree of size m. If $(\max,+)$convolution cannot be computed in $m^{2-\Omega(1)}$ time, then the sparsity cannot be computed in $m^{2-\Omega(1)}$ time either.*

Proof. Follows from Theorems 5.28 and 5.32 below. □

[1] Given three vectors $A=(A_0,\ldots,A_{n-1})^T$, $B=(B_0,\ldots,B_{n-1})^T$, and $C=(C_0,\ldots,C_{n-1})^T$, such that $A_i+B_j\le C_{i+j}$ for all $0\le i,j<n$, decide whether $A_i+B_j=C_{i+j}$ for any $0\le i,j<n$. No strongly subquadratic time algorithm is known for this problem.

Definition 5.25 (SUM_1 problem)**.** *Given three vectors $A, B, C \in \mathbb{Z}^n$, output*

$$\{k \mid \exists i, j \ : \ k = i + j \textit{ and } A_i + B_j + C_k \geq 0\}.$$

Definition 5.26 (SUM_2 problem)**.** *Given three vectors $A, B, C \in \mathbb{Z}^n$, output $t = 0, \dots, n-1$ such that*

$$t \in \{k \mid \exists i, j \ : \ k = i + j \textit{ and } A_i + B_j + C_k \geq 0\}$$

or report that there is no such an integer.

Definition 5.27 (SUM_3 problem)**.** *Given three vectors $A, B, C \in \mathbb{Z}^n$, decide if the following statement is true:*

$$\exists i, j \ : \ A_i + B_j + C_{i+j} \geq 0.$$

If $(\max, +)$-Convolution problem can be solved in strongly subquadratic time, so can SUM_3 problem. We will show the opposite direction - if SUM_3 problem can be solved in strongly subquadratic time, then $(\max, +)$-Convolution problem can be solved in strongly subquadratic time. We show this by reducing $(\max, +)$-Convolution problem to SUM_1 problem, SUM_1 problem to SUM_2 problem, SUM_2 problem to SUM_3 problem. Our proof of the following theorem uses the approach from [237] (see Sections 4.2 and 8).

Theorem 5.28. *If SUM_3 problem can be solved in a strongly subquadratic time, then* $(\max, +)$*-Convolution can be solved in a strongly subquadratic time.*

Proof. Follows from Lemmas 5.29, 5.30 and 5.31 below. □

It can be shown that SUM_3 problem is no harder than the $3SUM$ problem (given a set of integers, decide if it contains three integers that sum up to 0) [2]. This can be done using the techniques from [169] and [236] (see Theorem 3.3). This together with Theorem 5.28 implies that $(\max, +)$-convolution problem is no harder than the $3SUM$ problem (up to a factor that is logarithmic in the largest absolute value of an integer in the input).

Lemma 5.29. *Let $A, B \in \mathbb{Z}^n$ be input for* $(\max, +)$*-Convolution problem. Let W be the largest absolute value of an integer appearing in vector A or B. If SUM_1 can be solved in time $O(n^{2-\varepsilon})$, then* $(\max, +)$*-Convolution can be solved in $O(n^{2-\varepsilon} \cdot \log W)$ time.*

Proof. To solve $(\max, +)$-Convolution, we perform $O(\log W)$ steps of binary search for all indices in parallel.

We define two vectors $L_k = -10W$ and $H_k = 10W$ for all $k = 0, \dots, n-1$. We perform the following sequence of steps $100 \cdot \log W$ times:

1. Set $A' \leftarrow A$, $B' \leftarrow B$ and $C'_k \leftarrow -\left\lfloor \frac{L_k + H_k}{2} \right\rfloor$ for all $k = 0, \dots, n-1$.
2. Run the algorithm for SUM_1 on vectors A', B' and C'. Let R be the output set.
3. For every $k = 0, \dots, n-1$, consider two cases. If $k \in R$, then update $L_k \leftarrow \left\lfloor \frac{L_k + H_k}{2} \right\rfloor$. If $k \notin R$, then update $H_k \leftarrow \left\lfloor \frac{L_k + H_k}{2} \right\rfloor$.

Let C be the output of $(\max, +)$-Convolution instance that we need to output. During the execution of the algorithm above, we always maintain property that $L_k \leq C_k < H_k$ for all

$k = 0, \ldots, n-1$. After $100 \cdot \log W$ iterations, we have $L_k = H_k - 1$. This implies that $L_k = C_k$ for all $k = 0, \ldots, n-1$. Therefore, we output L_k as the answer. □

Lemma 5.30. *If SUM_2 can be solved in time $O(n^{2-\varepsilon})$, then SUM_1 can be solved in time $O(n^{2-(\varepsilon/2)})$.*

Proof. The idea is to run the algorithm for SUM_2 problem to find element t of interest, remove C_t from vector C and repeat until we found all elements that need to be reported. To achieve the stated runtime, we split both vectors A and B in $\sqrt{n}$ vectors each having $\sqrt{n}$ entries. Then we run the algorithm for SUM_2 for each pair of shorter vectors at least once. Below we provide more details.

Let $A = (A_0, \ldots, A_{n-1})^T$, $B = (B_0, \ldots, B_{n-1})^T$ and $C = (C_0, \ldots, C_{n-1})^T$ be the input vectors for SUM_1 problem. Let W be the largest absolute value of an integer in vector A or B. Let $T = \emptyset$ be the set that the algorithm will output. Initially the set is empty set. Let a be parameter that we will set later (we will set $a = 1/2$). For each pair of integer (i', j'), $i' = 0, \ldots, n^{1-a} - 1$, $j' = 0, \ldots, n^{1-a} - 1$, we perform the following sequence of steps (in total, we do the sequence of steps n^{2-2a} times).

1. For $i = 0, \ldots, n^a - 1$, we set $A'_i = A_{(i' \cdot n^a)+i}$. For $i = n^a, \ldots, 2n^a - 2$, set $A'_i = -10W$.
2. For $j = 0, \ldots, n^a - 1$, we set $B'_j = A_{(j' \cdot n^a)+j}$. For $j = n^a, \ldots, 2n^a - 2$, set $B'_j = -10W$.
3. For $k = 0, \ldots, 2n^a - 2$, we set $C'_k = C_{((i'+j') \cdot n^a)+k}$.
4. Run the algorithm for SUM_2 problem on vectors A', B' and C'. If the algorithm outputs an integer t, add integer t to set T and set $C_t = -10W$, and go to Step 1.

The correctness of the algorithm follows from its description. It remains to analyse its runtime. Suppose that the algorithm for SUM_2 runs in time $O(n^{2-\varepsilon})$. Since the length of vectors A', B' and C' is $O(n^a)$, every invocation of SUM_2 algorithm takes $S_1 := O(n^{a(2-\varepsilon)})$ time. We run algorithm for SUM_2 for every tuple (i', j'). The number of tuples is $S_2 := O(n^{2-2a})$. In Step 4 we might need to reiterate execution of the sequence of steps if we received some integer t. However, notice that no integer t can be reported twice by the definition of SUM_2 problem and because we set $C_t = -10W$. Therefore, the number of times we might need to reiterate the sequence of steps, is upper bounded by $S_3 := n$. Thus, the total runtime is upper bounded by

$$O(S_1 \cdot S_2 + S_1 \cdot S_3) = O\left(n^{a(2-\varepsilon)} \cdot n^{2-2a} + n^{a(2-\varepsilon)} \cdot n\right).$$

By setting $a = 1/2$, we get that the runtime is upper bounded by $O(n^{2-(\varepsilon/2)})$, as required. □

Lemma 5.31. *If SUM_3 problem can be solved in time $O(n^{2-\varepsilon})$, then SUM_2 problem can be solved in time $O(n^{2-\varepsilon} \cdot \log n)$.*

Proof. This follows by binary search. □

Theorem 5.32. *If tree sparsity can be solved in strongly subquadratic time, then SUM_3 problem can be solved in a strongly subquadratic time as well.*

Proof. Let $A, B, C \in \mathbb{Z}^n$ be the input to the SUM_3 problem. Let W be equal to 10 times the largest absolute value of an entry in A, B, C. Consider $A \in \mathbb{Z}^n$. We first construct a

Algorithm 23 Extracting a dense subtree.

```
function DenseSubtree($x, \Omega, k'$)
    Let $T$ be a tour through the nodes in $\Omega$ as they appear in a depth-first traversal.
    Let $I = (i_1, \ldots, i_{2|\Omega|-1})$ be the node indices in order of the tour $T$.
    Let $x'_j = \begin{cases} x_{i_j} & \text{if position } j \text{ is the first appearance of } i_j \text{ in } I \\ 0 & \text{otherwise} \end{cases}$
    Let $S = (i_j, \ldots, i_{j+k'-1})$ be a contiguous subsequence of $I$ with $\frac{1}{k'} \sum_{\ell=j}^{j+k'-1} x'_\ell \geq \frac{1}{2|\Omega|} \|x_\Omega\|_1$.
    return $\Omega'$, the set of nodes in $S$.
end function
```

path p_A of n nodes such that the weight of the first node is $11W + A_0$ and the weight of the i-th nodes is $W + A_{i-1} - A_{i-2}$ for $i = 2, \ldots, n$. p_B is constructed in the same way. Finally, we construct a path p_C such that the weight of the first node is $11W + C_{n-1}$ and the weight of the i-th nodes is $W + C_{n-i} - C_{n-i+1}$ for $i = 2, \ldots, n$. We then build the tree x as follows. The root of the tree is equal to the first node of p_A. The left child of the root is equal to the second node of p_A. The right child of the root is equal to the first node of p_B. (call it v). The left child of v is equal to the second node of p_B. Finally, the right child of v is equal to the first node of p_C. We set m to be the size of the tree (which is $3n$), and set k to be equal to $n + 2$. Observe that

$$\max_{\Omega \in \mathbb{T}_k} \|x_\Omega\|_1 \geq (2 + n)W + 30W$$

iff there are $i, j \in \{0, 1, \ldots, n-1\}$ such that $A_i + B_j + C_{i+j} \geq 0$.

To show the above inequality, let $k_A, k_B, k_C \geq 1$ be the number of nodes that we pick up from paths p_A, p_B, p_C, respectively, in the tree x in the optimal support Ω. k_A, k_B, k_C are positive because we assign very large weights to the first nodes of the paths. We have that $k_A + k_B + k_C = k = 2 + n$. It is easy to verify that the contributions from paths p_A, p_B, p_C are $10W + k_A W + A_{k_A-1}, 10W + k_B W + B_{k_B-1}, 10W + k_C W + C_{n-k_C}$, respectively. Since $k_A + k_B + k_C = 2 + n$, the total contribution from the three paths is $30W + (2 + n)W + A_{k_A-1} + B_{k_B-1} + C_{(k_A-1)+(k_B-1)}$, as required. □

5.6.5 Constant factor head approximation in nearly linear time

Before we give the main algorithm of this section, we setup some auxiliary sub-routines.

5.6.5.1 Subroutines

First, we state the following sub-routine which extracts a subtree of bounded size and proportional "density" from a given tree. A similar sub-routine has appeared before in [118], but here we give a variant that maintains exact sparsity.

Lemma 5.33. *Let $x \in \mathbb{R}^n_+$ be a vector of node weights and let Ω be a subtree. Moreover, let $k' \in \mathbb{N}$ be the target sparsity. Then* DenseSubtree(x, Ω, k') *returns a subtree Ω' of size*

$|\Omega'| \leq k'$ such that

$$\|x_{\Omega'}\|_1 \;\geq\; \frac{k'}{2|\Omega|}\|x_\Omega\|_1 \; .$$

Moreover, DenseSubtree *runs in time $O(|\Omega|)$.*

Proof. We use the notation set up in the algorithm DenseSubtree (see Algorithm 23). By a simple averaging argument, we know that at least one contiguous length-k' subsequence of I achieves at least the density of the sequence I, which is

$$\frac{1}{|I|}\sum_{j=1}^{|I|} x'_j \;>\; \frac{1}{2|\Omega|}\sum_{j=1}^{|I|} x'_j \;=\; \frac{1}{2|\Omega|}\|x_\Omega\|_1 \; .$$

The second inequality follows from the definition of x' (every node value in Ω appears exactly once in the sum).

Therefore, Line 5 of the algorithm always succeeds and we find a subset $\Omega' \subseteq \Omega$ corresponding to a subsequence $S = (i_j, \ldots, i_{j+k'-1})$ such that

$$\|x_{\Omega'}\|_1 \;=\; \sum_{\ell=j}^{j+k'-1} x'_\ell \;\geq\; \frac{k'}{2|\Omega|}\|x_\Omega\|_1 \; .$$

Finally, we can find such a dense contiguous sequence in linear time by maintaining a sliding window over the sequence I. □

Moreover, we use the following sub-routine for solving the Lagrangian relaxation of the tree sparsity problem [117].

Fact 5.34 ([117])**.** *There is an algorithm* SolveRelaxation *with the following guarantee. Let $x \in \mathbb{R}^n_+$ be a vector of node weights and let λ be the Lagrangian trade-off parameter. Then* SolveRelaxation(x, λ) *returns a subtree Ω such that*

$$\|x_\Omega\|_1 - \lambda|\Omega| \;\geq\; \max_{\Omega' \in \mathbb{T}} \|x_{\Omega'}\|_1 - \lambda|\Omega'| \; . \tag{5.15}$$

Moreover, SolveRelaxation *runs in time $O(n)$.*

5.6.5.2 Unrooted head approximation

We now give an algorithm for finding an *unrooted* subtree that achieves a constant-factor head approximation. While the resulting subtree is not connected to the root, we have sufficiently tight control over the sparsity so that we can later post-process the subtree by connecting it to the root.

Theorem 5.35. *Let $x \in \mathbb{R}^n_+$ be a vector of node weights, let $k \in \mathbb{N}$ be the target sparsity, and let $0 < \alpha < 1$ be a sparsity control parameter. Then* UnrootedHead(x, k, α) *returns a subtree Ω of size $|\Omega| \leq \alpha k$ such that*

$$\|x_\Omega\|_1 \;\geq\; \frac{\alpha}{16} \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1 \; .$$

Algorithm 24 Finding an unrooted head approximation.

```
1: function UnrootedHead($(x, k, \alpha)$)
2:     $x_{\max} \leftarrow \max_{i \in [n]} x_i$
3:     $\lambda_l \leftarrow \frac{x_{\max}}{2k}$
4:     $\Omega_l \leftarrow$ SolveRelaxation$(x, \lambda_l)$
5:     if $|\Omega_l| \leq 2k$ then
6:         return DenseSubtree$(x, \Omega_l, \alpha k)$
7:     end if
8:     $\lambda_r \leftarrow 2\|x\|_1$
9:     $\varepsilon \leftarrow \frac{x_{\max}}{4k}$
10:    while $\lambda_r - \lambda_l > \varepsilon$ do
11:        $\lambda_m \leftarrow \frac{\lambda_l + \lambda_r}{2}$
12:        $\Omega_m \leftarrow$ SolveRelaxation$(x, \lambda_m)$
13:        if $|\Omega_m| > 2k$ then
14:            $\lambda_l \leftarrow \lambda_m$
15:        else
16:            $\lambda_r \leftarrow \lambda_m$
17:        end if
18:    end while
19:    $\Omega_l \leftarrow$ SolveRelaxation$(x, \lambda_l)$
20:    $\Omega'_l \leftarrow$ DenseSubtree$(x, \Omega_l, \alpha k)$
21:    $\Omega_r \leftarrow$ SolveRelaxation$(x, \lambda_r)$
22:    $\Omega'_r \leftarrow$ DenseSubtree$(x, \Omega_r, \alpha k)$
23:    if $\|\Omega'_l\|_1 \geq \|\Omega'_r\|_1$ then
24:        return $\Omega'_l$
25:    else
26:        return $\Omega'_r$
27:    end if
28: end function
```

Moreover, UnrootedHead *runs in time $O(n \log n)$.*

Proof. We adopt the notation of Algorithm 24. Moreover, let Ω^* be an optimal subtree of size k, i.e., we have $\Omega^* \in \mathbb{T}_k$ and $\|x_{\Omega^*}\|_1 = \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1$.

There are three cases in which UnrootedHead returns a subtree: Lines 6, 24, and 26. We consider these three cases separately. Note that in every case, the final subtree is the result of a call to DenseSubtree with sparsity parameter αk. Hence the final subtree Ω returned by UnrootedHead always satisfies $|\Omega| \leq \alpha k$ (see Lemma 5.33). It remains to show that the subtree Ω also satisfies the desired head approximation guarantee.

Case 1: We start with Line 6. Substituting Ω^* into the guarantee of SolveRelaxation (see Fact 5.34), we get the following inequalities:

$$\begin{aligned} \|x_{\Omega_l}\|_1 - \lambda_l |\Omega_l| \;&\geq\; \max_{\Omega' \in \mathbb{T}} \|x_{\Omega'}\|_1 - \lambda_l |\Omega'| \;\geq\; \|x_{\Omega^*}\|_1 - \lambda_l k \\ \|x_{\Omega_l}\|_1 \;&\geq\; \|x_{\Omega^*}\|_1 - \lambda_l (k - |\Omega_l|) \; . \end{aligned} \tag{5.16}$$

Equation (5.16) and its variant for λ_r will become useful again later in the proof. Now, we substitute $\lambda_l = \frac{x_{\max}}{2k}$ and get

$$\|x_{\Omega_l}\|_1 \;\geq\; \|x_{\Omega^*}\|_1 - \frac{1}{2} x_{\max} \,.$$

Since we can assume that the sparsity k is at least the depth of the tree, a k-sparse rooted subtree can always pick up the largest node weight $x_{\max}$. So $\|x_{\Omega^*}\|_1 \geq x_{\max}$ and hence $\|x_{\Omega_l}\|_1 \geq \frac{1}{2}\|x_{\Omega^*}\|_1$.

To complete this case, we invoke the guarantee of DenseSubtree (Lemma 5.33) to get

$$\begin{aligned}\|x_{\Omega}\|_1 \;&\geq\; \frac{\alpha k}{2|\Omega_l|}\|x_{\Omega_l}\|_1 \\ &\geq\; \frac{\alpha}{4}\|x_{\Omega_l}\|_1 \\ &\geq\; \frac{\alpha}{8}\|x_{\Omega^*}\|_1 \,.\end{aligned}$$

Case 2: We now consider the case that the algorithm performs the binary search over λ and returns in Line 24 or 26. Note that we initialize the binary search so that we always have $|\Omega_l| > 2k$ and $|\Omega_r| \leq 2k$. Moreover, at the end of the binary search we also have $\lambda_l \leq \lambda_r \leq \lambda_l + \varepsilon$.

We now distinguish two sub-cases: in the first sub-case, we assume that the solution $|\Omega_l|$ has a good density $\frac{\|x_{\Omega_l}\|_1}{|\Omega_l|}$, which implies that the subtree Ω_l' is sufficiently good. In the complementary case, we can then use the low density of Ω_l to show that Ω_r' is a good solution.

Sub-case 2a: $\frac{\|x_{\Omega_l}\|_1}{|\Omega_l|} \geq \frac{1}{4}\frac{\|x_{\Omega^*}\|_1}{k}$. Substituting this inequality into the guarantee provided by DenseSubtree gives

$$\|x_{\Omega_l'}\|_1 \;\geq\; \frac{\alpha k}{2|\Omega_l|}\|x_{\Omega_l}\|_1 \;\geq\; \frac{\alpha}{8}\|x_{\Omega^*}\|_1$$

as desired.

Sub-case 2b: $\frac{\|x_{\Omega_l}\|_1}{|\Omega_l|} < \frac{1}{4}\frac{\|x_{\Omega^*}\|_1}{k}$. We lower bound $\|x_{\Omega_r}\|_1$ via the guarantee provided by SolveRelaxation, which we re-arrange as in Case 1 to get:

$$\begin{aligned}\|x_{\Omega_r}\|_1 \;&\geq\; \|x_{\Omega^*}\|_1 - \lambda_r(k - |\Omega_r|) \\ &\geq\; \|x_{\Omega^*}\|_1 - \lambda_r k \,. \end{aligned} \tag{5.17}$$

In order to control the RHS above, we need an upper bound on λ_r (note that $|\Omega_r|$ can be less than k). We establish this via an upper bound on λ_l and using that λ_l and λ_r are close at the end of the binary search. Re-arranging Equation (5.16) gives:

$$\begin{aligned}\lambda_l &\leq \frac{\|x_{\Omega_l}\|_1 - \|x_{\Omega^*}\|_1}{|\Omega_l| - k} \\ &\leq \frac{\|x_{\Omega_l}\|_1}{|\Omega_l| - k} \\ &< \frac{2\|x_{\Omega_l}\|_1}{|\Omega_l|} \\ &\leq \frac{1}{2}\frac{\|x_{\Omega^*}\|_1}{k}\end{aligned}$$

where we used $|\Omega_l| > 2k$ and the low-density assumption for Ω_l in this sub-case.

Substituting this upper bound and $\lambda_r \leq \lambda_l + \varepsilon$ back into Equation (5.17) gives

$$\begin{aligned}\|x_{\Omega_r}\|_1 &\geq \|x_{\Omega^*}\|_1 - \frac{1}{2}\|x_{\Omega^*}\|_1 - \varepsilon k \\ &\geq \frac{1}{4}\|x_{\Omega^*}\|_1\end{aligned}$$

where we used $x_{\max} \leq \|x_{\Omega^*}\|_1$ as in Case 1.

We now invoke the guarantee of DENSESUBTREE. As mentioned above, we maintain $|\Omega_r| \leq 2k$ as an invariant in the binary search. Hence we get

$$\|x_{\Omega'_r}\|_1 \geq \frac{\alpha k}{2|\Omega_r|}\|x_{\Omega_r}\|_1 \geq \frac{\alpha}{16}\|x_{\Omega^*}\|_1 .$$

Finally, we prove the running time of UNROOTEDHEAD. Since both subroutines SOLVERELAXATION and DENSESUBTREE run in linear time, the overall time complexity is dominated by the binary search. We can upper bound the number of iterations by the logarithm of

$$\frac{\lambda_r}{\varepsilon} = \frac{8k\|x\|_1}{x_{\max}} \leq 8nk \leq 8n^2 ,$$

which implies the running time bound in the theorem. □

5.6.5.3 Final head approximation algorithm

We now state our overall head approximation algorithm. The main idea is to invoke UNROOTEDHEAD form the previous subsection with a sufficiently small sparsity control parameter α so that we can connect the resulting subtree to the root without violating our sparsity constraint.

Theorem 5.36. *There is an algorithm* HEADAPPROX *with the following guarantee. Let $x \in \mathbb{R}^n_+$ be a vector of node weights, let d be the depth of the tree, and let $k \in \mathbb{N}$ be the target sparsity. Then* HEADAPPROX *returns a rooted subtree Ω of size $|\Omega| \leq k$ such that*

$$\|x_\Omega\|_1 \geq \frac{k-d}{16k} \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1 .$$

Moreover, HeadApprox *runs in time $O(n \log n)$.*

Proof. Let Ω' be the subtree returned by UnrootedHead$(x, k, \frac{k-d}{k})$. Hence Ω' satisfies $|\Omega'| \le k - d$ and

$$\|x_{\Omega'}\|_1 \;\ge\; \frac{k-d}{16k}$$

Next, let Ω_p be the path from the root of the subtree Ω' to the root of the overall tree. Since the depth of the overall tree is d, we have $|\Omega_p| \le d$.

We now let $\Omega = \Omega' \cup \Omega_p$ be the final subtree. Clearly, Ω is a rooted subtree and still satisfies the same head approximation guarantee as Ω'. Moreover, we have $|\Omega| = |\Omega'| + |\Omega_p| \le d$ as desired.

The running time of HeadApprox follows directly from Theorem 5.35. □

Corollary 5.37. *Assume that the depth of the input tree is at most a constant fraction of the sparsity k. Then* HeadApprox *is a constant-factor head-approximation algorithm with exact sparsity k.*

5.6.5.4 A boosted algorithm

Prior work shows that it is possible to "boost" a head-approximation algorithm so that the approximation ratio improves arbitrarily close to one while only incurring a small increase in sparsity [119].

Fact 5.38 ([119])**.** *Let* Head *be a head-approximation algorithm with approximation ratio c_H and output sparsity αk. Then* BoostedHead$_t$ *is a head-approximation algorithm with approximation ratio $c'_H = 1 - (1 - c_H)^t$ and output sparsity $t\alpha k$. Moreover,* BoostedHead *runs in time $O(t \cdot T_{\text{HEAD}})$, where T_{HEAD} is the time complexity of a single invocation of the algorithm* Head*.*

We can invoke Fact 5.38 and our new head approximation algorithm to give a range of trade-offs between sparsity increase and head-approximation ratio. It is worth noting that Fact 5.38 has only a logarithmic dependence on the gap between c_H and an exact head "approximation" with factor 1. For node weights x coming from a bounded range, this allows us to achieve an exact head (and hence also tail) approximation with only logarithmic increase in sparsity. More precisely, we get the following theorem, where we assume that the depth of the tree d is at most $k/2$ in order to simplify the bounds.

Theorem 5.39. *There is an algorithm* BoostedHeadApprox *with the following guarantee. Let $x \in \mathbb{N}_+^n$ be a vector of node weights bounded as $x_i \le \Delta$, and let $k \in \mathbb{N}$ be the target sparsity. Then* BoostedHeadApprox(x, k) *returns a rooted subtree Ω such that $|\Omega| \le 33k \log 2k\Delta$ and*

$$\|x_\Omega\|_1 \;\ge\; \max_{\Omega' \in \mathbb{T}_k} \|x_{\Omega'}\|_1 \,.$$

Moreover, BoostedHeadApprox *runs in time $O(n\,(\log n)\,(\log k\Delta))$.*

Proof. First, note that we can restrict our attention to trees with depth at most k. Any node that has larger distance from the root cannot participate in the optimal solution due to the sparsity constraint.

Next, we show that increasing our output sparsity by a constant factor allows us to get a result independent of the depth of the tree (this is in contrast to Theorem 5.36 and Corollary 5.37). We can construct an algorithm HeadApprox' that achieves a head approximation ratio of $\frac{1}{32}$ by invoking UnrootedHead from Theorem 5.35 with parameter $\alpha = \frac{1}{2}$. We ensure that the output is an unrooted tree by connecting it to the root. This increases the sparsity by at most the depth, which we just bounded by k. Hence the total sparsity of the output is $\alpha k + k = \frac{3}{2}\alpha$. Moreover, HeadApprox' runs in time $O(n \log n)$.

We now boost our new head approximation algorithm HeadApprox'. Rearranging the guarantee in Fact 5.38 shows that

$$t \;=\; \frac{\log \frac{1}{\varepsilon}}{\log \frac{1}{1-c_H}} \;\leq\; 22 \log \frac{1}{\varepsilon}$$

suffices for a boosted head approximation ratio of $c'_H = 1 - \varepsilon$.

Since we have integer node weights, the gap between an optimal head approximation and the second-best possible head approximation is at least 1. Moreover, the total weight of an optimal k-sparse subtree is at most $k\Delta$. Hence it suffices to set $\varepsilon = \frac{1}{2k\Delta}$ in order to guarantee that a $(1-\varepsilon)$-head approximation is exact. As a result, invoking BoostedHead with $t = 22 \log 2k\Delta$ produces a subtree Ω with the desired properties.

Each iteration of BoostedHead invokes HeadApprox' once. So the running time of HeadApprox' (see Theorem 5.36) and our bound on t imply the running time bound of the theorem. □

5.6.6 Approximating $(\max, +)$ and $(\min, +)$-convolutions

5.6.6.1 Approximating the $(\max, +)$ and $(\min, +)$-convolutions between two sequences

Let $A, B \in \mathbb{N}^n$ be two integer vectors with positive entries. Let W be the largest value of an entry in A or B. Let $C \in \mathbb{N}^n$ be the $(\max, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1+\varepsilon)C'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W\right).$$

An analogous statement holds if C is the $(\min, +)$-convolution of A and B. We can output a vector $C' \in \mathbb{N}^n$ such that $C'_k \leq C_k \leq (1+\varepsilon)C'_k$ for all $k = 0, \ldots, n-1$ in the same runtime.

Proof. Given a vector $D \in \mathbb{N}^n$ with positive entries and an integer $i \geq 0$, we define a binary vector $\chi(D, i)$:

$$\chi(D, i)_k := \begin{cases} 1 & \text{if } (1+\varepsilon)^i \leq D_k < (1+\varepsilon)^{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

For all pairs of integers $0 \leq i, j \leq \log_{1+\varepsilon} W$, define vector $\chi_{i,j} := \chi(A, i) * \chi(B, j)$. Computing all vectors $\chi_{i,j}$ takes total time

$$O\left((\log^2_{1+\varepsilon} W) \cdot n \log n\right).$$

Finally, we set

$$C'_k := \max_{\substack{0 \leq i,j \leq \log_{1+\varepsilon} W \\ \text{s.t. } (\chi_{i,j})_k = 1}} (1+\varepsilon)^i + (1+\varepsilon)^j \tag{5.18}$$

for all $k = 0, \ldots, n-1$. The runtime and the correctness follows from the description.

The proof for $(\min, +)$ convolution is analogous. □

5.6.6.2 Approximating the $(\max, +)$ and $(\min, +)$-convolutions between multiple sequences

Let $A^1 \in \mathbb{N}^{n_1}, \ldots, A^l \in \mathbb{N}^{n_l}$ be $l \geq 2$ vectors with positive entries. We assume that the entries of the vectors are indexed starting with 0, i.e., $A^i = \left(A^i_0, \ldots, A^i_{n_i-1}\right)^T$ for every $i = 1, \ldots, l$. Let $B \in \mathbb{N}^n$, $n = n_1 + \ldots + n_l$ be the $(\min, +)$-convolution between the l vectors $A^1, \ldots, A^{n_l}$ defined as follows:

$$B_j = \min_{\substack{m_1 + \ldots + m_l = j \\ \text{s.t. } 0 \leq m_t \leq n_t - 1 \ \forall t = 1, \ldots, l}} \left(A^1_{m_1} + \ldots + A^l_{m_l}\right)$$

for every $j = 0, \ldots, n-1$. We can output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$ in time

$$O\left(\frac{n}{\varepsilon^2} \cdot \log n \cdot \log^2 W \cdot \log^3 l\right),$$

where W denotes the largest entry in the vectors $A^1, \ldots, A^l$.

An analogous statement holds for $(\max, +)$-convolution (we replace min in the equation for B_j with max). The runtime stays the same and we output a vector $B' \in \mathbb{N}^n$ such that $B'_k \leq B_k \leq (1+\varepsilon)B'_k$ for all $k = 0, \ldots, n-1$.

Proof. We repeatedly use the fast algorithm for approximately computing $(\min, +)$-convolutions. Let B'' be the $(\min, +)$-convolution between the vectors $A_1, \ldots, A_{\lfloor l/2 \rfloor}$ and let B''' be the $(\min, +)$convolution between the vectors $A_{\lfloor l/2 \rfloor + 1}, \ldots, A_l$. Then B is the $(\min, +)$convolution between the *two* vectors B'' and B'''. This gives a natural recursive algorithm for computing B': recursively approximate B'' and B''' and use the approximation algorithm from Theorem 5.6.1.2 to compute get B'. If B'' and B''' are approximated within factor $(1+\varepsilon')$ and we set approximation factor to be $(1+\varepsilon'')$ in the algorithm from Theorem 5.6.1.2, we get $(1+\varepsilon')(1+\varepsilon'')$ approximation factor for B'. Since the depth of the recursion is $\log_2 l$, we get that the final approximation factor of B is $(1+\varepsilon'')^{\log_2 l}$. Setting $\varepsilon'' := O\left(\frac{\varepsilon}{\log l}\right)$ gives the required approximation factor for B'. Since the number of the recursion levels is $O(\log l)$ and every level takes $O\left(\frac{n}{\varepsilon''^2} \cdot \log n \cdot \log^2 W\right)$, we get the required runtime.

The proof for $(\max, +)$is analogous. □

5.6.7 Tree Sparsity in rectangular time on unbalanced trees

Theorem 5.40. *Given a binary tree of size n and an integer k, we can solve the Tree Sparsity problem in time $O(n^2)$.*

Proof. We show this inductively.

Consider the recursive algorithm for solving the Tree Sparsity problem. Consider a tree with the left subtree is of size L and the right subtree is of size R. The size of the tree is $1 + L + R$. For some constant C we can compute the sparsity vector of size $L + 1$ for the left subtree in time $C \cdot L^2$ and the sparsity vector of size $R + 1$ for the right subtree in time $C \cdot R^2$. These two sparsity vectors can be combined to get the sparsity vector for the root. As can be easily verified, combining the sparsity vectors takes time $C' \cdot LR$ for some constant C'. Thus, overall time to compute the sparsity vector for the root is

$$C \cdot L^2 + C \cdot R^2 + C' \cdot LR \leq C \cdot (1 + L + R)^2$$

if $C \geq C'$ as required. □

Theorem 5.41. *Given a binary tree of size n and an integer k, we can solve the Tree Sparsity problem in time $O(kn)$.*

Proof. We observe that for every node we need to compute $k + 1$ entries of the sparsity vector.

Consider the original tree. A node it *heavy* if both left and the right subtrees are of size $\geq k$. If both subtrees are of size $< k$, the node is *light*. Otherwise, the node is *average*.

The runtime corresponding to heavy, average and light vertices can be analyzed as follows.

- A simple counting reveals that the total number of heavy nodes is $O(n/k)$. The total runtime corresponding to the heavy nodes is $O(n/k) \cdot O(k^2) = O(nk)$, where $O(k^2)$ comes from combining two sparsity vectors of size $k + 1$.
- Consider an average node u. The runtime corresponding to it is upper bounded by $O(n_u k)$ where n_u is the size of the subtree of u. This is because one sparsity vector is of size $n_u + 1$ and the other is of size $k + 1$. The total runtime corresponding to the average nodes is $\sum_u O(n_u k) = O(nk)$, where the summation is over all average nodes.
- Consider a light node u whose parent is average or heavy. Let s_u be the size of the tree rooted at u. The runtime corresponding to this tree is $O(s_u^2)$ by Theorem 5.40. The total runtime corresponding to the light nodes is upper bounded by $\sum_u O(s_u^2) = O(nk)$ where the summation is over all light nodes whose parents are heavy or average. The inequality follows because $\sum_u s_u = O(n)$ and $s_u = O(k)$ for all such nodes u.

□

Chapter 6

Earth Mover Distance Sparsity

6.1 Introduction

The notion of *sparsity* has emerged as a powerful tool in signal processing theory and applications. A signal (or image) is said to be k-sparse if only k of its coefficients in a given basis expansion are nonzero. In other words, the intrinsic information content in the signal is miniscule relative to its apparent size. This simple notion enables a wide variety of conceptual and algorithmic techniques to compress, reconstruct, denoise, and process practical high-dimensional signals and images.

In several practical applications, the nonzero coefficients of signal ensembles exhibit additional, richer relationships that cannot be captured by simple sparsity. Consider, for example, a 2D "image" constructed by column-wise stacking of seismic time traces (or shot records) measured by geophones positioned on a uniform linear array. Assuming the presence of only a few subsurface reflectors, the physics of wave propagation dictates that such a 2D image would essentially consist of a small number of curved lines, possibly contaminated with noise (see Figure 6-1). A convenient model for such an image is to simply assume that each column is sparse. However, this simple model ignores the fact that the indices of the nonzeros change smoothly across adjacent columns. Such settings are commonplace: similar "line" singularities are often encountered in other applications such as biological imaging and radio-astronomy.

In this chapter, we propose a model for sparse signal ensembles where the locations of the nonzeros, or the support, of a signal transforms continuously as a function of spatial (or temporal) location. A key ingredient in our model is the classical Earth Mover Distance (EMD) [193]. Hence we call our model the *Constrained EMD* model. Informally, our proposed model assumes that:

- each signal in our ensemble is k-sparse, and
- the cumulative EMD between pairs of adjacent supports is constrained to be no greater than a nonnegative parameter B.

The parameter B controls how dramatically the support can vary across different signals. A value of $B = 0$ indicates that the support remains constant across all signals in our ensemble, while a large value of B admits potential large jumps across adjacent signal supports.

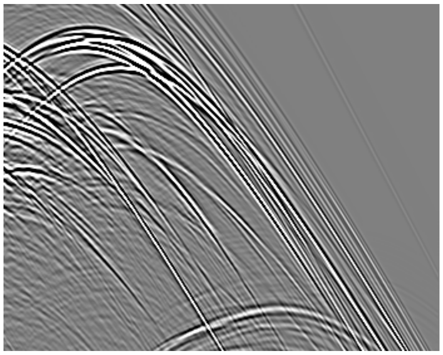


Figure (6-1): Part of a typical seismic shot record (Sigsbee2A data set). The x-axis corresponds to space (receiver) and the y-axis to time. Note that the large coefficients of neighboring columns are at similar locations.

Compressive sensing with the CEMD Model We design both head- and tail-approximations for our proposed CEMD model.

- Our tail-approximation oracle returns a support set with tail-approximation error at most a constant times larger than the optimal tail error. At the same time, the EMD-budget of the solution is still $O(B)$ (Theorem 6.15). The algorithm relies on solving a small number of *min-cost max-flow* problems over a specially defined graph.
- Our head-approximation oracle returns a support set with head value at least a constant fraction of the optimal head value. Moreover, the EMD-budget of the solution is $O(B \log \frac{k}{w})$ (Theorem 6.7). We achieve this guarantee with a greedy algorithm that iteratively adds paths to the solution.

Combining these algorithms into our new framework, we obtain a compressive sensing scheme for the CEMD model using $O(k \log(\frac{B}{k} \log(\frac{k}{w})))$ measurements for robust signal recovery. For a reasonable choice of parameters, e.g., $B = O(k)$, the bound specializes to $m = O(k \log\log(\frac{k}{w}))$, which is very close to the information-theoretic optimum of $m = O(k)$.

Additionally, we demonstrate the advantages of the Constrained EMD model, and the associated approximation algorithms, in the context of compressive sensing. Geometrically, the model is equivalent to a particular *union of subspaces* of the ambient signal space. Therefore, we can leverage the framework of *model-based compressive sensing* [36] to build a new CS reconstruction algorithm that is specially tailored to signal ensembles well-described by the Constrained EMD model. We illustrate the numerical benefits of the new algorithm in comparison with existing state-of-the-art approaches.

6.1.1 Prior work

There has been a substantial amount of work devoted to reconstructing time sequences of spatially sparse signals, e.g., [91, 227]. These works assume that the support of the signal (or even the signal itself) does not change much between two consecutive steps. However, the change of supports between two columns a and b was defined according to the ℓ_0 norm, i.e., as $\|s(a) - s(b)\|_0$. In contrast, we measure this difference using the EMD distance. As a result, our model easily handles signals such as those in Figure 6-5, where the supports of

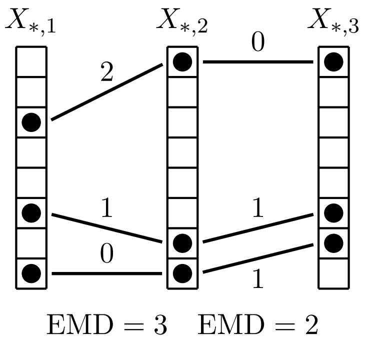


Figure (6-2): The support-EMD for a matrix with three columns and eight rows. The circles stand for supported elements in the columns. The lines indicate the matching between the supported elements and the corresponding EMD cost. The total support-EMD is $\text{EMD}(\text{supp}(X)) = 2 + 3 = 5$.

any two consecutive columns can potentially be *even disjoint*, yet differ very little according to the EMD.

Another related work is that of [125], who proposed the use of EMD in a compressive sensing context in order to measure the *approximation error* of the recovered signal. In contrast, in this chapter we are using EMD to constrain the *support set* of the signals.

6.2 The CEMD model

As an instantiation of our main results, we discuss a special structured sparsity model known as the *Constrained EMD* model [199]. A key ingredient in the model is the Earth Mover's Distance (EMD), also known as the Wasserstein metric or Mallows distance [150]:

Definition 6.1 (EMD)**.** *The EMD of two finite sets $A, B \subset \mathbb{N}$ with $|A| = |B|$ is defined as*

$$\text{EMD}(A, B) = \min_{\pi : A \to B} \sum_{a \in A} |a - \pi(a)| \;, \tag{6.1}$$

where π ranges over all one-to-one mappings from A to B.

Observe that $\text{EMD}(A, B)$ is equal to the cost of a min-cost matching between A and B. Now, consider the case where the sets A and B are the *supports* of two exactly k-sparse signals, so that $|A| = |B| = k$. In this case, the EMD not only measures how many indices change, but also how far the supported indices move. This notion can be generalized from pairs of signals to an *ensemble* of sparse signals. Figure 6-2 illustrates the following definition.

Definition 6.2 (Support-EMD)**.** *Let $\Omega \subseteq [h] \times [w]$ be the support of a matrix X with exactly s-sparse columns, i.e., $|\text{col-supp}(\Omega, c)| = s$ for $c \in [w]$. Then the EMD of Ω is defined as*

$$\text{EMD}(\Omega) = \sum_{c=1}^{w-1} \text{EMD}(\text{col-supp}(\Omega, c), \text{col-supp}(\Omega, c+1)) \;.$$

If the columns of X are not exactly s-sparse, we define the EMD of Ω as the mini-

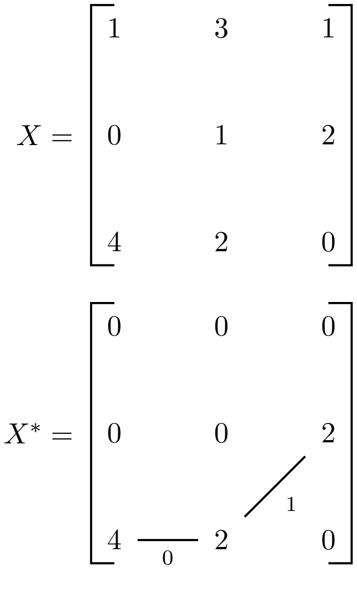


Figure (6-3): A signal X and its best approximation X^* in the EMD model $\mathcal{M}_{3,1}$. A sparsity constraint of 3 with 3 columns implies that each column has to be 1-sparse. Moreover, the total support-EMD between neighboring columns in X^* is 1. The lines in X^* indicate the support-EMD.

mum EMD of any support that contains Ω *and has exactly* s*-sparse columns. Let* $s = \max_{c \in [w]} |\text{col-supp}(\Omega, c)|$. *Then* $\text{EMD}(\Omega) = \min_\Gamma \text{EMD}(\Gamma)$*, where* $\Gamma \subseteq [h] \times [w]$, $\Omega \subseteq \Gamma$*, and* Γ *is a support with exactly* s*-sparse columns, i.e.,* $|\text{col-supp}(\Gamma, c)| = s$ *for* $c \in [w]$.

The above definitions motivate a natural structured sparsity model that essentially characterizes ensembles of sparse signals with correlated supports. Suppose we interpret the signal $x \in \mathbb{R}^n$ as a matrix $X \in \mathbb{R}^{h \times w}$ with $n = h\,w$. For given dimensions of the signal X, our model has two parameters: (i) k, the total sparsity of the signal. For simplicity, we assume here and in the rest of this chapter that k is divisible by w. Then the sparsity of each column $X_{*,i}$ is $s = k/w$. (ii) B, the support-EMD of X. We call this parameter the *EMD budget*. Formally, we have:

Definition 6.3 (Constrained EMD model). *The Constrained EMD (CEMD) model is the structured sparsity model* $\mathcal{M}_{k,B}$ *defined by the set of supports* $\mathbb{M}_{k,B} = \{\Omega \subseteq [h] \times [w] \,|\, \text{EMD}(\Omega) \leq B \text{ and } |\text{col-supp}(\Omega, c)| = \frac{k}{w} \text{ for } c \in [w]\}$.

The parameter B controls how much the support can vary from one column to the next. Setting $B = 0$ forces the support to remain constant across all columns, which corresponds to block sparsity (the blocks are the rows of X). A value of $B \geq kh$ effectively removes the EMD constraint because each supported element is allowed to move across the full height of the signal. In this case, the model demands only s-sparsity in each column. It is important to note that we only constrain the EMD of the column *supports* in the signal, not the actual amplitudes. Figure 6-3 illustrates the CEMD model with an example.

6.3 Sampling bound

Our objective is to develop a sparse recovery scheme for the Constrained EMD model. As the first ingredient, we establish the model-RIP for $\mathcal{M}_{k,B}$, i.e., we characterize the number of permissible supports (or equivalently, the number of subspaces) $l_{k,B}$ in the model and invoke Fact 2.15. For simplicity, we will assume that $w = \Omega(\log h)$, i.e., the following bounds apply for all signals X except very thin and tall matrices X. The following result is novel:

Theorem 6.4. *The number of allowed supports in the CEMD model satisfies* $\log|\mathbb{M}_{k,B}| = O(k \log \frac{B}{k})$.

Proof. For given h, w, B, and k, the support is fixed by the following three decisions: (i) The choice of the supported elements in the first column of X. (ii) The distribution of the EMD budget B over the k supported elements. This corresponds to distributing B balls into $k+1$ bins (using one bin for the part of the EMD budget not allocated to supported elements). (iii) For each supported element, the direction (up or down) to the matching element in the next column to the right. Multiplying the choices above gives $\binom{h}{s}\binom{B+k}{k}2^k$, an upper bound on the number of supports. Using the inequality $\binom{a}{b} \leq \left(\frac{a\,e}{b}\right)^b$, we get

$$\begin{aligned}
\log|\mathbb{M}_{k,B}| &\leq \log\left(\binom{h}{s}\binom{B+k}{k}2^k\right) \\
&\leq s \log \frac{h}{s} + k \log \frac{B+k}{k} + O(s+k) \\
&= O\left(k \log \frac{B}{k}\right). \qquad \square
\end{aligned}$$

If we allow each supported element to move a constant amount from one column to the next, we get $B = O(k)$ and hence, from Fact 2.15, $m = O(k + \log|\mathbb{M}_{k,B}|) = O(k)$ rows for sub-Gaussian measurement matrices. This bound is information-theoretically optimal. Furthermore, for $B = kh$ (i.e., allowing every supported element to move anywhere in the next column) we get $m = O(k \log n)$, which almost matches the standard compressive sensing bound of $m = O(k \log \frac{n}{k})$ for sub-Gaussian measurement matrices. Therefore, the CEMD model gives a smooth trade-off between the support variability and the number of measurements necessary for recovery.

We can also establish a sampling bound in the RIP-1 setting with Fact 2.24. For the case of $B = \Theta(k)$, we get $m = O(k \frac{\log n}{\log \log \frac{n}{k}})$. In order to match the block-sparsity lower bound of $m = O(k \log_w n)$, we need to assume that $B = O(k/w)$, i.e., each path (and not each element) in the support has a constant EMD-budget on average. We omit the details of this calculation here.

The following theorem is useful when establishing sampling bounds for recovery schemes using the CEMD model.

Theorem 6.5. *The CEMD model is closed under addition:* $\mathbb{M}_{k_1,B_1} + \mathbb{M}_{k_2,B_2} \subseteq \mathbb{M}^{+}_{k_1+k_2,B_1+B_2}$.

Proof. Let $\Omega_1 \in \mathbb{M}_{k_1,B_1}$ and $\Omega_2 \in \mathbb{M}_{k_2,B_2}$. Moreover, let $\Gamma = \Omega_1 \cup \Omega_2$. We have to show that $\Gamma \in \mathbb{M}_{k_1+k_2,B_1+B_2}$.

The column-sparsity of Ω_1 and Ω_2 is k_1/w and k_2/w, respectively. Hence the column-sparsity of Γ is at most $\frac{k_1+k_2}{w}$. Moreover, we can construct a matching for Γ with cost at most $B_1 + B_2$ from the matchings for Ω_1 and Ω_2. To see this, consider without loss of generality the matchings π_1 and π_2 corresponding to the first two columns in Ω_1 and Ω_2, respectively. We start constructing the new matching π' by starting with π_1. Then, we iterate over the pairs (a, b) in π_2 one by one and augment π' to include both a and b. There are four cases:

1. Both a and b are still unassigned in π'. Then we can simply add (a, b) to π'.

2. Both a and b are already assigned in π'. In this case, we do not need to modify π' to include a and b.

3. a is not included in π', but b is already assigned in π'. This is the interesting case because we must now find a new neighbor assignment for a. Let b' be the entry in the second column that is in the same row as a. If b' is not assigned yet, we can simply add (a, b') to π'. Otherwise, let a' be the value such that $\pi'(a') = b'$. Then we remove the pair (a', b') from π', add (a, b') to π', and repeat this procedure to find a new neighbor for a'. It is easy to see that this procedure terminates after a finite number of steps, and that no node currently assigned under π' loses a neighbor. Moreover, note that this operation does not increase the cost of the matching π'.

4. b is not included in π', but a is already assigned in π'. This case is symmetric to case 3 above.

Each of the four cases increases the cost of π' by at most the cost of (a, b) in π_2. Iterating over all pairs in π_2, we observe that the final matching π' has cost no more than the cumulative costs of π_1 and π_2, i.e., at most $B_1 + B_2$. Therefore, $\Gamma \in \mathbb{M}_{k_1+k_2,B_1+B_2}$. □

6.4 Head Approximation Algorithm

First, we develop a head approximation algorithm for the CEMD model. Ideally, we would have an *exact* projection algorithm H mapping arbitrary signals to signals in $\mathcal{M}_{k,B}$ with the guarantee $\|H(x)\|_p = \max_{\Omega\in\mathbb{M}_{k,B}} \|x_\Omega\|_p$. However, this appears to be a hard problem. Instead, we propose an efficient greedy algorithm satisfying the somewhat looser requirements of a head approximation oracle (Definition 2.16). Specifically, we develop an algorithm that performs the following task: given an arbitrary signal x, find a support $\Omega \in \mathbb{M}_{O(k),O(B\log k)}$ such that $\|x_\Omega\|_p^p \geq c \max_{\Gamma\in\mathbb{M}_{k,B}} \|x_\Gamma\|_p^p$, where $c > 0$ is a fixed constant.

As before, we interpret our signal x as a matrix $X \in \mathbb{R}^{h\times w}$. Let OPT denote the largest sum of coefficients achievable with a support in $\mathbb{M}_{k,B}$, i.e., $OPT = \max_{\Omega\in\mathbb{M}_{k,B}} \|x_\Omega\|_p^p$. For a signal $x \in \mathcal{M}_{k,B}$, we interpret the support of x as a set of $s = k/w$ paths from the leftmost to the rightmost column in X. Our method proceeds by greedily finding a set of paths that cover a large sum of signal coefficients. We can then show that the coefficients covered by these paths are a constant fraction of the optimal coefficient sum OPT.

Definition 6.6 (Path in a matrix). *Given a matrix $X \in \mathbb{R}^{h\times w}$, a path $r \subseteq [h] \times [w]$ is a set of w locations in X with one location per column, i.e., $|r| = w$ and $\bigcup_{(i,j)\in r} j = [w]$. The weight of r is the sum of amplitudes on r, i.e., $w_{X,p}(r) = \sum_{(i,j)\in r} |X_{i,j}|^p$. The EMD of*

Algorithm 25 Head approximation algorithm

```
function HeadApprox($X, k, B$)
    $X^{(1)} \leftarrow X$
    for $i \leftarrow 1, \ldots, s$ do
        Find the path $r_i$ from column 1 to column $w$ in $X^{(i)}$ that maximizes $w^{(i)}(r_i)$ and
            uses at most EMD-budget $\lfloor \frac{B}{i} \rfloor$.
        $X^{(i+1)} \leftarrow X^{(i)}$
        for $(u, v) \in r_i$ do
            $X^{(i+1)}_{u,v} \leftarrow 0$
        end for
    end for
    return $\bigcup_{i=1}^{s} r_i$
end function
```

r is the sum of the EMDs between locations in neighboring columns. Let $j_1, \ldots, j_w$ be the locations of r in columns 1 to w. Then, $\mathrm{EMD}(r) = \sum_{i=1}^{w-1} |j_i - j_{i+1}|$.

Trivially, we have that a path r in X is a support with $w_{X,p}(r) = \|X_r\|_p^p$ and $\mathrm{EMD}(r) = \mathrm{EMD}(\mathrm{supp}(X_r))$. Therefore, we can iteratively build a support Ω by finding s paths in X. Algorithm 25 contains the description of HeadApprox. We show that HeadApprox finds a constant fraction of the amplitude sum of the best support while only moderately increasing the size of the model. For simplicity, denote $w(r) := w_{X,p}(r)$, and $w^{(i)}(r) := w_{X^{(i)},p}(r)$. We obtain the following result:

Theorem 6.7. *Let $p \geq 1$ and $B' = \lceil H_s \rceil B$, where $H_s = \sum_{i=1}^{s} 1/i$ is the s-th harmonic number. Then* HeadApprox *is a $((\frac{1}{4})^{1/p}, \mathbb{M}_{k,B}, \mathbb{M}_{k,B'}, p)$-head-approximation oracle.*

Proof. Let Ω be the support returned by HeadApprox(X, k, B) and let $\Omega_{OPT} \in \mathbb{M}_{k,B}$ be an optimal support. We can always decompose Ω_{OPT} into s disjoint paths in X. Let $t_1, \ldots, t_s$ be such a decomposition with $\mathrm{EMD}(t_1) \geq \mathrm{EMD}(t_2) \geq \ldots \geq \mathrm{EMD}(t_s)$. Note that $\mathrm{EMD}(t_i) \leq \lfloor \frac{B}{i} \rfloor$: otherwise $\sum_{j=1}^{i} \mathrm{EMD}(t_i) > B$ and since $\mathrm{EMD}(\Omega_{OPT}) \leq B$ this would be a contradiction. Since Ω is the union of s disjoint paths in X, Ω has column-sparsity s. Moreover, we have $\mathrm{EMD}(\Omega) = \sum_{i=1}^{s} \mathrm{EMD}(r_i) \leq \sum_{i=1}^{s} \lfloor \frac{B}{i} \rfloor \leq \lceil H_s \rceil B$. Therefore, $\Omega \in \mathbb{M}^{+}_{k,B'}$.

When finding path r_i in $X^{(i)}$, there are two cases:

Case 1: $w^{(i)}(t_i) \leq \frac{1}{2} w(t_i)$, i.e., the paths $r_1, \ldots, r_{i-1}$ have already covered more than half of the coefficient sum of t_i in X.

Case 2: $w^{(i)}(t_i) > \frac{1}{2} w(t_i)$, i.e., there is still more than half of the coefficient sum of t_i remaining in $X^{(i)}$. Since $\mathrm{EMD}(t_i) \leq \lfloor \frac{B}{i} \rfloor$, the path t_i is a candidate when searching for the optimal path r_i and hence we find a path r_i with $w^{(i)}(r_i) > \frac{1}{2} w(t_i)$.

Let $C = \{i \in [s] \mid \text{case 1 holds for } r_i\}$ and $D = \{i \in [s] \mid \text{case 2 holds for } r_i\}$ (note that

$C = [s] \setminus D$). Then we have

$$\begin{aligned} \|X_\Omega\|_p^p = \sum_{i=1}^{s} w^{(i)}(r_i) &= \sum_{i \in C} w^{(i)}(r_i) + \sum_{i \in D} w^{(i)}(r_i) \\ &\geq \sum_{i \in D} w^{(i)}(r_i) \geq \frac{1}{2} \sum_{i \in D} w(t_i)\,. \end{aligned} \tag{6.2}$$

For each t_i with $i \in C$, let $E_i = t_i \cap \bigcup_{j<i} r_j$, i.e., the locations of t_i already covered by some r_j when searching for r_i. Then we have

$$\sum_{(u,v) \in E_i} |X_{u,v}|^p = w(t_i) - w^{(i)}(t_i) \geq \frac{1}{2} w(t_i)\,,$$

and

$$\sum_{i \in C} \sum_{(u,v) \in E_i} |X_{u,v}|^p \geq \frac{1}{2} \sum_{i \in C} w(t_i)\,.$$

The t_i are pairwise disjoint, and so are the E_i. For every $i \in C$ we have $E_i \subseteq \bigcup_{j=1}^{s} r_j$. Hence

$$\|X_\Omega\|_p^p = \sum_{i=1}^{s} w^{(i)}(r_i) \geq \sum_{i \in C} \sum_{(u,v) \in E_i} |X_{u,v}|^p \geq \frac{1}{2} \sum_{i \in C} w(t_i)\,. \tag{6.3}$$

Combining Equations 6.2 and 6.3 gives:

$$\begin{aligned} 2\|X_\Omega\|_p^p &\geq \frac{1}{2} \sum_{i \in C} w(t_i) + \frac{1}{2} \sum_{i \in D} w(t_i) = \frac{1}{2} OPT\,, \\ \|X_\Omega\|_p &\geq (\frac{1}{4})^{1/p} \max_{\Omega' \in \mathbb{M}_{k,B}} \|X_{\Omega'}\|_p\,. \end{aligned}$$

□

Theorem 6.8. HeadApprox *runs in* $O(snBh)$ *time.*

Proof. Observe that the running time of HeadApprox depends on the running time of finding a path with maximum weight for a given EMD budget. The search for such a path can be performed by *dynamic programming* over a graph with $whB = nB$ nodes, or equivalently "states" of the dynamic program.[1] Each state in the graph corresponds to a state in the dynamic program, i.e., a location $(i, j) \in [w] \times [h]$ and the current amount of EMD already used $b \in \{0, 1, \ldots, B\}$. At each state, we store the largest weight achieved by a path ending at the corresponding location (i, j) and using the corresponding amount of EMD budget b. Each state has h outgoing edges to the states in the next column (given the current location, the decision on the next location also fixes the new EMD amount). Hence the time complexity of finding one largest-weight path is $O(nBh)$ (the state space has size $O(nB)$ and each update requires $O(h)$ time). Since we repeat this procedure s times, the overall time complexity of HeadApprox is $O(snBh)$. □

We can achieve an arbitrary constant head-approximation ratio by combining HeadApprox

[1] We use the terminology "states" here to distinguish the dynamic program from the graph we will introduce in Section 6.5.

with BoostHead (see Section 2.6.5). The resulting algorithm has the same time complexity as HeadApprox. Moreover, the sparsity and EMD budget of the resulting support is only a constant factor larger than k and B'.

6.5 Tail-Approximation Algorithm

Next, we develop a tail-approximation algorithm for the CEMD model. Given an arbitrary signal x, our objective is to find a support $\Gamma \in \mathbb{M}_{k,O(B)}$ such that

$$\|x - x_\Gamma\|_p \leq c \min_{\Omega \in \mathbb{M}_{k,B}} \|x - x_\Omega\|_p \,, \tag{6.4}$$

where c is a constant. Note that we allow a constant factor increase in the EMD budget of the result. The algorithm we develop is precisely the graph-based approach initially proposed in [199]; however, our analysis here is rigorous and novel. Two core elements of the algorithm are the notions of a *flow network* and the *min-cost max-flow problem*, which we now briefly review. We refer the reader to [10] for an introduction to the graph-theoretic definitions and algorithms we employ.

The min-cost max-flow problem is a generalization of the classical maximum flow problem [10, 77]. In this problem, the input is a graph $G = (V, E)$ with designated source and sink nodes in which every edge has a certain capacity. The goal is to find an assignment of flow to edges such that the total flow from source to sink is maximized. The flow must also be valid, i.e., the amount of flow entering any intermediate node must be equal to the amount of flow leaving that intermediate node, and the amount of flow on any edge can be at most the capacity of that edge.

In the min-cost max-flow problem, every edge e also has a cost c_e (in addition to the capacity as before). The goal now is to find a flow $f : E \to \mathbb{R}_0^+$ with maximum capacity such that the cost of the flow, i.e., $\sum_{e \in E} c_e \cdot f(e)$, is minimized. One important property of the min-cost max-flow problem is that it still admits *integral* solutions if the edge capacities are integer.

Fact 6.9 (Theorem 9.10 in [10])**.** *If all edge capacities, the source supply, and the sink demand are integers, then there is always an integer min-cost max-flow.*

The min-cost max-flow problem has many applications, and several efficient algorithms are known [10]. We leverage this problem for our tail-approximation task by carefully constructing a suitable flow network, which we now define.

Definition 6.10 (EMD flow network)**.** *For a given signal X, sparsity k, and a parameter $\lambda > 0$, the flow network $G_{X,k,\lambda}$ consists of the following elements:*

- *The* nodes *comprise a source, a sink and a node $v_{i,j}$ for $i \in [h]$, $j \in [w]$, i.e., one node per entry in X (besides source and sink).*
- *G has an* edge *from every $v_{i,j}$ to every $v_{k,j+1}$ for $i, k \in [h]$, $j \in [w-1]$. Moreover, there is an edge from the source to every $v_{i,1}$ and from every $v_{i,w}$ to the sink.*
- *The* capacity *on every edge and node (except source and sink) is 1.*
- *The* cost *of node $v_{i,j}$ is $-|X_{i,j}|^p$. The cost of an edge from $v_{i,j}$ to $v_{k,j+1}$ is $\lambda|i-k|$. The cost of the source, the sink, and all edges incident to the source or sink is 0.*
- *The* supply *at the source is s $(= \frac{k}{w})$ and the demand at the sink is s.*

Figure 6-4 illustrates this definition with an example. The main idea is that a set of disjoint paths through the network $G_{X,k,\lambda}$ corresponds to a support in X. For any fixed value of λ, a solution of the min-cost max-flow problem on the flow network reveals a subset S of the nodes that corresponds to a support with exactly s indices per column and minimizes $-\|X_\Omega\|_p^p + \lambda \mathrm{EMD}(\Omega)$ for different choices of support Ω. In other words, the min-cost flow solves a *Lagrangian relaxation* of the original problem (6.4). See Lemmas 6.12 and 6.13 for a more formal statement of this connection.

$$X = \begin{bmatrix} 1 & 3 \\ \\ 0 & -1 \\ \\ 2 & 1 \end{bmatrix}$$

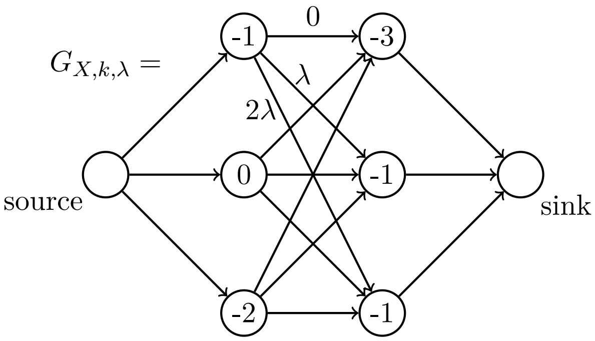


Figure (6-4): A signal X with the corresponding flow network $G_{X,k,\lambda}$ for $p = 1$. The node costs are the negative absolute values of the corresponding signal components. The numbers on edges indicate the edge costs (most edge costs are omitted for clarity). All capacities in the flow network are 1. The edge costs are the vertical distances between the start and end nodes, multiplied by λ.

A crucial issue is the choice of the Lagrange parameter λ, which defines a trade-off between the size of the tail approximation error and the support-EMD. Note that the optimal support Ω with parameters k and B does not necessarily correspond to *any* setting of λ. Nevertheless, we show that the set of supports we explore by varying λ contains a sufficiently good approximation: the tail error and the parameters k and B are only increased by constant factors compared to the optimal support Ω. Moreover, we show that we can find such a good support efficiently via a binary search over λ. Before stating our algorithm and the main result, we formalize the connection between flows and supports.

Definition 6.11 (Support of a set of paths)**.** *Let $X \in \mathbb{R}^{h \times w}$ be a signal matrix, k be a sparsity parameter, and $\lambda \geq 0$. Let $P = \{q_1, \ldots, q_s\}$ be a set of disjoint paths from source to sink in $G_{X,k,\lambda}$ such that no two paths in P intersect vertically (i.e., if the q_i are sorted vertically and $i \leq j$, then $(u,v) \in q_i$ and $(w,v) \in q_j$ implies $u < w$). Then the paths in P define a support*

$$\Omega_P = \{(u,v) \mid (u,v) \in q_i \text{ for some } i \in [s]\} . \tag{6.5}$$

Lemma 6.12. *Let $X \in \mathbb{R}^{h \times w}$ be a signal matrix, k be a sparsity parameter and $\lambda \geq 0$. Let*

$P = \{q_1, \ldots, q_s\}$ be a set of disjoint paths from source to sink in $G_{X,k,\lambda}$ such that no two paths in P intersect vertically. Finally, let f_P be the flow induced in $G_{X,k,\lambda}$ by sending a single unit of flow along each path in P and let $c(f_P)$ be the cost of f_P. Then

$$c(f_P) = -\|X_{\Omega_P}\|_p^p + \lambda \operatorname{EMD}(\Omega_P)\,. \tag{6.6}$$

Proof. The theorem follows directly from the definition of $G_{X,k,\lambda}$ and Ω_P. The node costs of P result in the term $-\|X_{\Omega_P}\|_p^p$. Since the paths in P do not intersect vertically, they are a min-cost matching for the elements in Ω_P. Hence the cost of edges between columns of X sums up to $\lambda \operatorname{EMD}(\Omega_P)$. □

For a fixed value of λ, a min-cost flow in $G_{X,k,\lambda}$ gives an optimal solution to the Lagrangian relaxation:

Lemma 6.13. *Let $G_{X,k,\lambda}$ be an EMD flow network and let f be an integral min-cost flow in $G_{X,k,\lambda}$. Then f can be decomposed into s disjoint paths $P = \{q_1, \ldots, q_s\}$ which do not intersect vertically. Moreover,*

$$\begin{aligned}\|X - X_{\Omega_P}\|_p^p &+ \lambda \mathrm{EMD}(\Omega_P)\\ &= \min_{\Omega \in \mathbb{M}_{k,B}} \|X - X_\Omega\|_p^p + \lambda \mathrm{EMD}(\Omega)\,.\end{aligned} \tag{6.7}$$

Proof. Note that $\|X - X_\Omega\|_p^p = \|X\|_p^p - \|X_\Omega\|_p^p$. Since $\|X\|_p^p$ does not depend on Ω, minimizing $\|X - X_\Omega\|_p^p + \lambda\mathrm{EMD}(\Omega)$ with respect to Ω is equivalent to minimizing $-\|X_\Omega\|_p^p + \lambda\mathrm{EMD}(\Omega)$.

Further, all edges and nodes in $G_{X,k,\lambda}$ have capacity one, so f can be composed into exactly s disjoint paths P. Moreover, the paths in P are not intersecting vertically: if q_i and q_j intersect vertically, we can relax the intersection to get a set of paths P' with smaller support EMD and hence a flow with smaller cost – a contradiction. Moreover, each support $\Omega \in \mathbb{M}_{k,B}$ gives rise to a set of disjoint, not vertically intersecting paths Q and thus also to a flow f_Q with $c(f_Q) = -\|X_{\Omega_Q}\|_p^p + \lambda\mathrm{EMD}(\Omega_Q)$. Since f is a min-cost flow, we have $c(f) \le c(f_Q)$. The statement of the theorem follows. □

We can now state our tail-approximation algorithm TailApprox (see Algorithm 26). The parameters d and δ for TailApprox quantify the acceptable tail approximation ratio (see Theorem 6.15). In the algorithm, we assume that MinCostFlow($G_{X,k,\lambda}$) returns the support corresponding to an integral min-cost flow in $G_{X,k,\lambda}$. Before we prove the main result (Theorem 6.15), we show that TailApprox always returns an optimal result for signals $X \in \mathcal{M}_{k,B}$.

Lemma 6.14. *Let $x_{\min} = \min_{|X_{i,j}|>0} |X_{i,j}|^p$ and $\lambda_0 = \frac{x_{\min}}{2wh^2}$. Moreover, let $X \in \mathcal{M}_{k,B}$ and Ω be the support returned by* MinCostFlow(G_{X,k,λ_0}). *Then $\|X - X_\Omega\|_p = 0$ and $\Omega \in \mathbb{M}^+_{k,B}$.*

Proof. Let $\Gamma = \operatorname{supp}(X)$, so $\Gamma \in \mathbb{M}^+_{k,B}$. First, we show that $\|X - X_\Omega\|_p = 0$. For contradiction, assume that $\|X - X_\Omega\|_p^p > 0$, so $\|X - X_\Omega\|_p^p \ge x_{\min} > 0$ (tail-approximation is trivial

Algorithm 26 Tail approximation algorithm

```
function TailApprox($X, k, B, d, \delta$)
    $x_{\min} \leftarrow \min_{|X_{i,j}|>0} |X_{i,j}|^p$
    $\varepsilon \leftarrow \frac{x_{\min}}{wh^2}\delta$
    $\lambda_0 \leftarrow \frac{x_{\min}}{2wh^2}$
    $\Omega \leftarrow$ MinCostFlow($G_{X,k,\lambda_0}$)
    if $\Omega \in \mathbb{M}_{k,B}$ and $\|X - X_\Omega\|_p = 0$ then
        return $\Omega$
    end if
    $\lambda_r \leftarrow 0$
    $\lambda_l \leftarrow \|X\|_p^p$
    while $\lambda_l - \lambda_r > \varepsilon$ do
        $\lambda_m \leftarrow (\lambda_l + \lambda_r)/2$
        $\Omega \leftarrow$ MinCostFlow($G_{X,k,\lambda_m}$)
        if EMD($\Omega$) $\geq B$ and EMD($\Omega$) $\leq dB$ then
            return $\Omega$
        end if
        if EMD($\Omega$) $> B$ then
            $\lambda_r \leftarrow \lambda_m$
        else
            $\lambda_l \leftarrow \lambda_m$
        end if
    end while
    $\Omega \leftarrow$ MinCostFlow($G_{X,k,\lambda_l}$)
    return $\Omega$
end function
```

for $X = 0$). Since Ω is a min-cost flow, Lemma 6.13 gives

$$\begin{aligned} x_{\min} &\leq \|X - X_\Omega\|_p^p + \lambda_0 \mathrm{EMD}(\Omega) \\ &= \min_{\Omega' \in \mathbb{M}_{k,B}} \|X - X_{\Omega'}\|_p^p + \lambda_0 \mathrm{EMD}(\Omega') \\ &\leq 0 + \frac{x_{\min}}{2wh^2} \mathrm{EMD}(\Gamma) \\ &\leq \frac{x_{\min}}{2}\,, \end{aligned}$$

which gives a contradiction. The last line follows from $\mathrm{EMD}(\Gamma) \leq kh \leq nh$.

Now, we show that $\Omega \in \mathbb{M}_{k,B}^+$. By construction of G_{X,k,λ_0}, Ω is s-sparse in each column. Moreover,

$$\|X - X_\Omega\|_p^p + \lambda_0 \mathrm{EMD}(\Omega) = \min_{\Omega' \in \mathbb{M}_{k,B}} \|X - X_\Omega\|_p^p + \lambda_0 \mathrm{EMD}(\Omega')\,,$$

$$\lambda_0 \mathrm{EMD}(\Omega) \leq 0 + \lambda_0 \mathrm{EMD}(\Gamma)\,.$$

So $\mathrm{EMD}(\Omega) \leq \mathrm{EMD}(\Gamma) \leq B$. □

Next, we prove a *bicriterion*-approximation guarantee for TailApprox that allows us to use TailApprox as a tail approximation algorithm. In particular, we show that one of the following two cases occurs:

Case 1: The tail-approximation error achieved by our solution is at least as good as the best tail-approximation error achievable with support-EMD B. The support-EMD of our solution is at most a constant times larger than B.

Case 2: Our solution has bounded tail-approximation error and support-EMD at most B.

In order to simplify the proof of the main theorem, we use the following shorthands: $\Omega_l = \textsc{MinCostFlow}(G_{X,k,\lambda_l})$, $\Omega_r = \textsc{MinCostFlow}(G_{X,k,\lambda_r})$, $b_l = \text{EMD}(\Omega_l)$, $b_r = \text{EMD}(\Omega_r)$, $t_l = \|X - X_{\Omega_l}\|_p^p$, and $t_r = \|X - X_{\Omega_r}\|_p^p$.

Theorem 6.15. *Let $d > 1$, $\delta > 0$, and let Ω be the support returned by the algorithm* TailApprox*(X, k, B, d, δ). Let OPT be the tail approximation error of the best support with support-EMD at most B, i.e., $OPT = \min_{\Gamma \in \mathbb{M}_{k,B}} \|X - X_\Gamma\|_p^p$. Then at least one of the following two guarantees holds for Ω:*

Case 1: $B \leq \text{EMD}(\Omega) \leq dB$ *and* $\|X - X_\Omega\|_p^p \leq OPT$

Case 2: $\text{EMD}(\Omega) \leq B$ *and* $\|X - X_\Omega\|_p^p \leq (1 + \frac{1}{d-1} + \delta) OPT$.

Proof. We consider the three cases in which TailApprox returns a support. If TailApprox returns in line 7, the first guarantee in the theorem is satisfied. If TailApprox reaches the binary search (line 11), we have $X \notin \mathcal{M}_{k,B}$ (the contrapositive of Lemma 6.14). Therefore, we have $OPT \geq x_{\min} > 0$ in the remaining two cases.

If TailApprox returns in line 15, we have $B \leq \text{EMD}(\Omega) \leq dB$. Moreover, Lemma 6.13 gives

$$\begin{aligned}\|X - X_\Omega\|_p^p &+ \lambda_m \text{EMD}(\Omega) \\ &\leq \min_{\Omega' \in \mathbb{M}_{k,B}} \|X - X_{\Omega'}\|_p^p + \lambda_m \text{EMD}(\Omega') \\ &\leq OPT + \lambda_m B\,.\end{aligned}$$

Since $\text{EMD}(\Omega) \geq B$, we have $\|X - X_\Omega\|_p^p \leq OPT$.

We now consider the third return statement (line 24), in which case the binary search terminated with $\lambda_l - \lambda_r \leq \varepsilon$. In the binary search, we maintain the invariant that $b_l \leq B$ and $b_r > dB$. Note that this is true before the first iteration of the binary search due to our initial choices of λ_r and λ_l.[1] Moreover, our update rule maintains the invariant.

We now prove the bound on $\|X - X_\Omega\|_p^p = t_l$. From Lemma 6.13 we have

$$\begin{aligned}t_r + \lambda_r b_r &\leq OPT + \lambda_r B \\ \lambda_r dB &\leq OPT + \lambda_r B \\ \lambda_r &\leq \frac{OPT}{B(d-1)}\,.\end{aligned}$$

[1] Intuitively, our initial choices make the support-EMD very cheap and very expensive compared to the tail approximation error.

Since the binary search terminated, we have $\lambda_l \leq \lambda_r + \varepsilon$. We now combine this inequality with our new bound on λ_r and use it in the following inequality (also from Lemma 6.13):

$$\begin{aligned} t_l + \lambda_l b_l &\leq OPT + \lambda_l B \\ t_l &\leq OPT + \lambda_l B \\ &\leq OPT + (\lambda_r + \varepsilon)B \\ &\leq OPT + \frac{OPT}{d-1} + \varepsilon B \\ &\leq (1 + \frac{1}{d-1})OPT + \frac{x_{\min}\delta B}{wh^2} \\ &\leq (1 + \frac{1}{d-1})OPT + \delta x_{\min} \\ &\leq (1 + \frac{1}{d-1} + \delta)OPT\,. \end{aligned}$$

This shows that the second guarantee of the theorem is satisfied. □

Corollary 6.16. *Let $p \geq 1$, $c > 1$, $0 < \delta < c - 1$, and $d = 1 + \frac{1}{c-\delta-1}$. Then* TailApprox *is a $(c^{1/p}, \mathbb{M}_{k,B}, \mathbb{M}_{k,dB}, p)$-tail approximation algorithm.*

Proof. The tail approximation guarantee follows directly from Theorem 6.15. Note that we cannot control which of the two guarantees the algorithm returns. However, in any case we have $\text{EMD}(\Omega) \leq dB$, so $\Omega \in \mathbb{M}_{k,dB}$. □

In order to simplify the time complexity of TailApprox, we assume that $h = \Omega(\log w)$, i.e., the matrix X is not very "wide" and "short". We arrive at the following result.

Theorem 6.17. *Let $\delta > 0$, $x_{\min} = \min_{|X_{i,j}|>0} |X_{i,j}|^p$, and $x_{\max} = \max |X_{i,j}|^p$. Then* TailApprox *runs in $O(snh(\log \frac{n}{\delta} + \log \frac{x_{\max}}{x_{\min}}))$ time.*

Proof. We can solve our instances of the min-cost flow problem by finding s augmenting paths because all edges and nodes have unit capacity. Moreover, $G_{X,k,\lambda}$ is a directed acyclic graph, so we can compute the initial node potentials in linear time. Each augmenting path can then be found with a single run of Dijkstra's algorithm, which costs $O(wh\log(wh) + wh^2) = O(nh)$ time [77]. The number of iterations of the binary search is at most

$$\begin{aligned} \log \frac{\|X\|_p^p}{\epsilon} &= \log \frac{\|X\|_p^p nh}{x_{\min}\delta} \\ &\leq \log \frac{x_{\max} n^2 h}{x_{\min}\delta} \\ &\leq \log \frac{n^3}{\delta} + \log \frac{x_{\max}}{x_{\min}}\,. \end{aligned}$$

Combining this with a per-iteration cost of $O(snh)$ gives the stated running time. □

To summarize, the algorithm proposed in [199] satisfies the criteria of a tail-approximation oracle. This, in conjunction with the head approximation oracle proposed in Section 6.4, gives a full sparse recovery scheme for the CEMD model, which we describe below.

6.6 Compressive Sensing Recovery

We now bring the results from the previous sections together. Specifically, we show that AM-IHT (Algorithm 3), equipped with HeadApprox and TailApprox, constitutes a model-based compressive sensing recovery algorithm that significantly reduces the number of measurements necessary for recovering signals in the CEMD model. The main result is the following theoretical guarantee:

Theorem 6.18. *Let $x \in \mathcal{M}_{k,B}$ be an arbitrary signal in the CEMD model with dimension $n = wh$. Let $A \in \mathbb{R}^{m \times n}$ be a measurement matrix with i.i.d. Gaussian entries and let $y \in \mathbb{R}^m$ be a noisy measurement vector, i.e., $y = Ax + e$ with arbitrary $e \in \mathbb{R}^m$. Then we can recover a signal approximation $\widehat{x} \in \mathcal{M}_{k,2B}$ satisfying $\|x - \widehat{x}\|_2 \le C\|e\|_2$ for some constant C from $m = O(k \log(\frac{B}{k} \log \frac{k}{w}))$ measurements. Moreover, the recovery algorithm runs in time $O(n \log \frac{\|x\|_2}{\|e\|_2}(k \log n + \frac{kh}{w}(B + \log n + \log \frac{x_{\max}}{x_{\min}})))$ where $x_{\min} = \min_{|x_i|>0}|x_i|$ and $x_{\max} = \max|x_i|$.*

Proof. First, we show that m rows suffice for A to have the desired model-RIP. Following the conditions in Corollary 2.28, A must satisfy the $(\delta, \mathbb{M}_{k,B} + \mathbb{M}_T + \mathbb{M}_H^{+t})$-model-RIP for small δ, where t is the number of times we boost HeadApprox (a constant depending on δ and c_T). We have $\mathbb{M}_T = \mathbb{M}_{k,2B}$ from Corollary 6.16 and $\mathbb{M}_H = \mathbb{M}_{2k,3\gamma B}$ where $\gamma = \lceil \log \frac{k}{w} \rceil + 1$ from Theorems 6.5 and 6.7 (note that HeadApprox must be a $(c_H, \mathbb{M} + \mathbb{M}_T, \mathbb{M}_H, 2)$-head-approximation oracle). Invoking Theorem 6.5 again shows that it suffices for A to have the $(\delta, \mathbb{M}_{(2+2t)k,(3+3t\gamma)B})$-model-RIP. Using Theorem 6.4 and the fact that t is a constant, Fact 2.15 then shows that

$$m \;=\; O\left(k \log \frac{\gamma B}{k}\right) \;=\; O\left(k \log \left(\frac{B}{k} \log \frac{k}{w}\right)\right)$$

suffices for A to have the desired model-RIP.

Equipped with our model-RIP, we are now able to invoke Corollary 2.28, which directly gives the desired recovery guarantee $\|x - \widehat{x}\|_2 \le C\|e\|_2$. Moreover, the corollary also shows that the number of iterations of AM-IHT is bounded by $O(\log \frac{\|x\|_2}{\|e\|_2})$. In order to prove our desired time complexity, we now only have to bound the per-iteration cost of AM-IHT.

In each iteration of AM-IHT, the following operations have a relevant time complexity: (i) Multiplication with A and A^T. The measurement matrix has at most $k \log n$ rows, so we bound this time complexity by $O(nk \log n)$. (ii) HeadApprox. From Theorem 6.8 we know that HeadApprox runs in time $O(n\frac{kh}{w}B)$. (iii) TailApprox. Theorem 6.17 shows that the tail-approximation algorithm runs in time $O(n\frac{kh}{w}(\log n + \log \frac{x_{\max}}{x_{\min}}))$. Combining these three bounds gives the running time stated in the theorem. □

Note that for $B = O(k)$, the measurement bound gives $m = O(k \log \log \frac{k}{w})$, which is a significant improvement over the standard compressive sensing measurement bound $m = O(k \log \frac{n}{k})$. In fact, the bound for m is only a $\log \log \frac{k}{w}$ factor away from the information-theoretically optimal bound $m = O(k)$. We leave it as an open problem whether this spurious factor can be eliminated via a more refined analysis or algorithm.

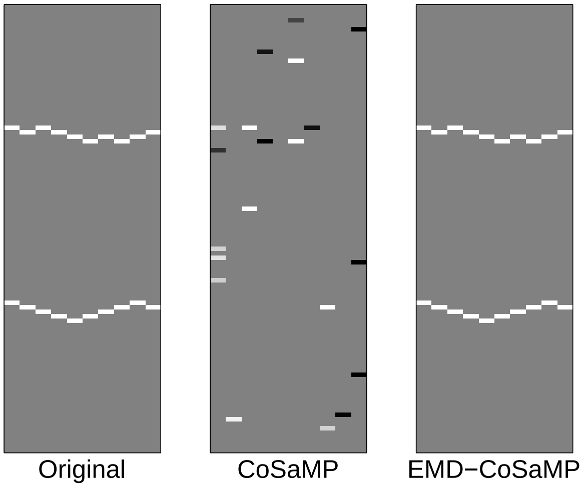


Figure (6-5): Benefits of CS reconstruction using EMD-CoSaMP. (left) Original image with parameters $h = 100$, $w = 10$, $k = 2$, $B = 20$, $m = 80$. (center) CS reconstruction using CoSaMP [170]. (right) CS reconstruction using EMD-CoSaMP. CoSaMP fails, while our proposed algorithm is able to perfectly recover the image.

6.7 Numerical Experiments

In all our experiments, we use the LEMON library [85] in order to solve the min-cost max-flow subroutine in Algorithm 26. Figure 6-5 displays a test grayscale image of size 100×10 with edge discontinuities such that the total sparsity is $2 \times 10 = 20$ and the cumulative EMD across pairs of adjacent columns is equal to $B = 20$. We measure linear samples of this image using merely $m = 80$ random Gaussian measurements, and reconstruct using CoSaMP as well our proposed approach (EMD-CoSaMP). Each iteration of EMD-CoSaMP takes less than three seconds to execute. As visually evident from Fig. 6-5, CoSaMP fails to reconstruct the image, while our proposed algorithm provides an accurate reconstruction.

Figure 6-6 displays the results of a Monte Carlo experiment to quantify the effect of the number of random measurements M required by different CS reconstruction algorithms to enable accurate reconstruction. Each data point in Fig. 6-6 was generated using 100 sample trials over randomly generated measurement matrices. Successful recovery is declared when the converged solution is within an ℓ_2 distance of 5% relative to the Euclidean norm of the original image. We observe that our proposed EMD-CoSaMP and EMD-IHT algorithms achieve successful recovery with far fewer measurements than their conventional (unmodified) counterparts.

6.8 Conclusions

We have proposed a structured sparsity model for images with line singularities that is based on the Earth Mover Distance (EMD). Moreover, we have introduced corresponding approximate projection algorithms based on ideas from greedy algorithms and min-cost maximum flow. We have leveraged these algorithm to develop a new compressive sensing recovery algorithm with significant numerical benefits. We remark that we have also employed our

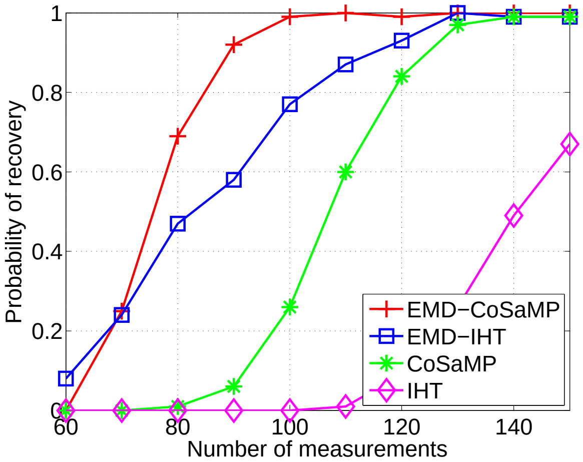


Figure (6-6): Comparison of several reconstruction algorithms. The signal is the same as in Figure 6-5. The probability of recovery is with respect to the measurment matrix and generated using 100 trial runs. The recovery algorithms using our constrained EMD model have a higher probability of recovery than standard algorithms.

new algorithms for an application in seismic data analysis, but this work is beyonod the scope of this thesis [200].

There are at least two interesting avenues for future work in the context of the CEMD model. Is our use of approximate projections necessary? A conditional hardness result for an exact projection into the CEMD model would provide strong evidence. Moreover, the time complexity of our algorithms is not optimal. An ideal approximate projection would run in time that is nearly-linear in the true input size. Note that such an approach cannot explicitly instantiate the full graph as in our algorithms since this step already requires strongly superlinear time. Hence a nearly-linear time algorithm will probably require new insights.

Chapter 7

Low-Rank Matrices

7.1 Introduction

The previous chapters focused on various forms of sparsity such as graph sparsity, group sparsity, or hierarchical sparsity. A common theme in the corresponding approximate projections is their heavy reliance on ideas from combinatorial algorithms. Combinatorial problems are a natural starting point for approximate projections because they allow us to leverage the rich toolkit developed in the approximation algorithms community [228, 238]. However, an important question is whether approximate projections are also useful in a broader context. In this chapter, we take a step in this direction and study *low-rank* constraints, which have received a large amount of attention over the past decade. We show that approximate projections lead to faster running times for low rank matrices as well. Note that low-rank matrices are an instance of *linear algebraic* structure, as opposed to the *combinatorial* structure from previous chapters.

Over the past decade, exploiting low-rank structure in high-dimensional problems has become an active area of research in machine learning, signal processing, and statistics. The general approach is to utilize a low-rank model of data in order to achieve better prediction, compression, or estimation compared to a "black box" treatment of the ambient high-dimensional space. This parallels the seminal work on compressive sensing and sparse linear regression where sparsity allows us to estimate a high-dimensional object from a small number of (linear) observations.

Important problems involving low-rank matrices include matrix completion, matrix sensing, and matrix regression. The common theme is that the rank constraint captures important struture present in real world data, which leads to better statistical performance. However, utilizing the low-rank structure also comes at a cost: incorporating the constraint into the estimation procedure often results in a more challenging algorithmic problem. If we want to incorporate a low-rank constraint into our optimization (estimation) problem via projections, the main sub-routine of interest is a *singular value decomposition* (SVD).

For a matrix $\Theta \in \mathbb{R}^{d_1 \times d_2}$, the time to compute an SVD (and hence an exact projection) is $O(d_1 \cdot d_2 \cdot \min(d_1, d_2))$. While this is a polynomial time complexity, even polynomial scaling can become prohibitively slow given the growing size of modern datasets. Note that the time complexity of an exact SVD (or the closely related low-rank projection) is unlikely to improve significantly in practice. Matrix decompositions of this form have been studied for decades and no practically faster algorithms are known. This leads to the following important question: Can we design efficient approximate projections for low-rank matrices?

In this chapter, we show that we can utilize results on approximate SVDs [165] to give approximate head and tail projections for low-rank matrices. While approximate SVDs (such as PROPACK [147]) have been used in low-rank problems before, this approach did not come with provable guarantees. We show that our variant of projected gradient descent tolerates the slack introduced by approximate SVDs. This also provides an explanation for the empirical success of approximate projections in prior work. Moreover, our approach improves over PGD with PROPACK in numerical experiments (see Section 7.4).

The remainder of this section contains related work and preliminaries for our theoretical results. We provide head and tail approximation guarantees for low-rank matrices in Section 7.2. Section 7.3 then instantiates our algorithms in the context of low-rank matrix recovery. Section 7.4 complements our theoretical results with experiments.

7.1.1 Related work

Low-rank recovery has received a tremendous amount of attention over the past few years, so we refer the reader to the recent survey [83] for an overview. When referring to prior work on low-rank recovery, it is important to note that the fastest known running time for an exact low-rank SVD (even for rank 1) of a $d_1 \times d_2$ matrix is $O(d_1 d_2 \min(d_1, d_2))$. Several papers provide rigorous proofs for low-rank recovery using exact SVDs and then refer to Lanczos methods such as PROPACK [147] while accounting a time complexity of $O(d_1 d_2 r)$ for a rank-r SVD. While Lanczos methods can be faster than exact SVDs in the presence of singular value gaps, it is important to note that all rigorous results for Lanczos SVDs either have a polynomial dependence on the approximation ratio or singular value gaps [165, 196]. No prior work on low-rank recovery establishes such singular value gaps for the inputs to the SVD subroutines (and such gaps would be necessary for *all* iterates in the recovery algorithm). In contrast, we utilize recent work on gap-independent approximate SVDs [165], which enables us to give rigorous guarantees for the entire recovery algorithm. Our results can be seen as justification for the heuristic use of Lanczos methods in prior work.

Becker, Cevher, and Kyrillidis [39] analyze an approximate SVD in combination with an iterative recovery algorithm. However, [39] only uses an approximate tail projection, and as a result the approximation ratio $c_{\mathcal{T}}$ must be very close to 1 in order to achieve a good sample complexity. Overall, this leads to a time complexity that does not provide an asymptotic improvement over using exact SVDs.

Recently, several authors have analyzed a non-convex approach to low-rank matrix recovery via factorized gradient descent [45, 74, 223, 243, 244]. While these algorithms avoid SVDs in the iterations of the gradient method, the overall recovery proofs still require an exact SVD in the initialization step. In order to match the sample complexity of our algorithm or SVP, the factorized gradient methods require multiple SVDs for this initialization [74, 223]. As a result, our algorithm offers a better provable time complexity. We remark that [74, 223] use SVP for their initialization, so combining our faster version of SVP with factorized gradient descent might give the best overall performance.

7.1.2 Preliminaries

In previous chapters, we studied structured sparsity models as special forms of conic constraint sets. While this suffices for combinatorial types of structure, the low-rank constraint in this chapter requires a more general definition. To formulate the low-rank results in a common framework with the histograms constraints in the next chapter, we now introduce *subspace models.*

Definition 7.1 (Subspace model)**.** *A subspace model $\mathbb{U}$ is a set of linear subspaces. The set of vectors associated with the subspace model $\mathbb{U}$ is $\mathcal{M}(\mathbb{U}) = \{\theta \,|\, \theta \in U \text{ for some } U \in \mathbb{U}\}$.*

It is easy to see that structured sparsity models are an instance of subspace models (each allowed support is a subspace). Moreover, low-rank matrices also fit this definition. We can encode low-rank matrices by letting $\mathbb{U}$ be the set of rank-r matrix subspaces, i.e., each subspace is given by a set of r orthogonal rank-one matrices. Note that this is an instance of a subspace model with an infinite number of subspaces. For the discrete sparsity models, we instead have a finite union of subspaces.

As in previous chapters, we use approximate projections to overcome the bottleneck posed by exact projections. Specializing our general head an tail approximations from Chapter 2 to subspace models yields the following definitions:

Definition 7.2 (Approximate tail projection)**.** *Let $\mathbb{U}$ and $\mathbb{U}_{\mathcal{T}}$ be subspace models and let $c_{\mathcal{T}} \geq 0$. Then $\mathcal{T} : \mathbb{R}^d \to \mathbb{U}_{\mathcal{T}}$ is a $(c_{\mathcal{T}}, \mathbb{U}, \mathbb{U}_{\mathcal{T}})$-approximate tail projection if the following guarantee holds for all $b \in \mathbb{R}^d$: The returned subspace $U = \mathcal{T}(b)$ satisfies $\|b - P_U b\| \leq c_{\mathcal{T}} \|b - P_{\mathbb{U}} b\|$.*

Definition 7.3 (Approximate head projection)**.** *Let $\mathbb{U}$ and $\mathbb{U}_{\mathcal{H}}$ be subspace models and let $c_{\mathcal{H}} > 0$. Then $\mathcal{H} : \mathbb{R}^d \to \mathbb{U}_{\mathcal{H}}$ is a $(c_{\mathcal{H}}, \mathbb{U}, \mathbb{U}_{\mathcal{H}})$-approximate head projection if the following guarantee holds for all $b \in \mathbb{R}^d$: The returned subspace $U = \mathcal{H}(b)$ satisfies $\|P_U b\| \geq c_{\mathcal{H}} \|P_{\mathbb{U}} b\|$.*

As before, the two definitions are distinct in the sense that a constant-factor head approximation does not imply a constant-factor tail approximation, or vice versa. Another feature of these definitions is that the approximate projections are allowed to choose subspaces from a potentially larger subspace model, i.e., we can have $\mathbb{U} \subsetneq \mathbb{U}_{\mathcal{H}}$ (or $\mathbb{U}_{\mathcal{T}}$). This is a useful property when designing approximate head and tail projection algorithms as it allows for *bicriterion* approximation guarantees.

7.2 Head and tail approximations for low-rank matrices

We now give our head an tail approximation guarantees for low-rank matrices. We use the following result from prior work on approximate SVDs.

Fact 7.4 ([165])**.** *There is an algorithm* ApproxSVD *with the following guarantee. Let $A \in \mathbb{R}^{d_1 \times d_2}$ be an arbitrary matrix, let $r \in \mathbb{N}$ be the target rank, and let $\varepsilon > 0$ be the desired accuracy. Then with probability $1 - \psi$,* ApproxSVD(A, r, ε) *returns an orthonormal set of vectors $z_1, \ldots, z_r \in \mathbb{R}^{d_1}$ such that for all $i \in [r]$, we have*

$$|z_i^T A A^T z_i - \sigma_i^2| \;\leq\; \varepsilon \sigma_{r+1}^2 \,, \tag{7.1}$$

where σ_i is the i-th largest singular value of A. Furthermore, let $Z \in \mathbb{R}^{d_1 \times r}$ be the matrix with columns z_i. Then we also have

$$\|A - ZZ^T A\|_F \;\leq\; (1+\varepsilon)\|A - A_r\|_F \;, \tag{7.2}$$

where A_r is the best rank-r Frobenius-norm approximation of A. Finally, the algorithm runs in time $O(\frac{d_1 d_2 r \log(d_2/\psi)}{\sqrt{\varepsilon}} + \frac{d_1 r^2 \log^2(d_2/\psi)}{\varepsilon} + \frac{r^3 \log^3(d_2/\psi)}{\varepsilon^{3/2}})$.

It is important to note that the above results hold for *any* input matrix and do not require singular value gaps. The guarantee (7.2) directly gives a tail approximation guarantee for the subspace corresponding to the matrix $ZZ^T A$. Moreover, we can convert the guarantee (7.1) to a head approximation guarantee. In the following, we let $\mathbb{U}_r$ be the subspace model of rank-r matrices.

As mentioned before, Equation (7.2) directly gives a tail approximation. We now show how to convert Equation (7.1) to a head approximation guarantee. In the following, we let $\mathbb{U}_r$ be the subspace model of rank-r matrices.

Theorem 7.5. *There is an algorithm* ApproxLowRank *with the following property. For an arbitrary input matrix $A \in \mathbb{R}^{d_1 \times d_2}$ and a target rank r,* ApproxLowRank *produces a subspace of rank-r matrices U and a matrix $Y = P_U A$, the projection of A onto U. With probability 99/100, the output satisfies both an $(1-\varepsilon, \mathbb{U}_r, \mathbb{U}_r)$-approximate head projection guarantee and an $(1+\varepsilon, \mathbb{U}_r, \mathbb{U}_r)$-approximate tail projection guarantee. Moreover,* ApproxLowRank *runs in time*

$$O(\frac{d_1 d_2 r \log d_2}{\sqrt{\varepsilon}} + \frac{d_1 r^2 \log^2 d_2}{\varepsilon} + \frac{r^3 \log^3 d_2}{\varepsilon^{3/2}}) \;.$$

Proof. Let $z_1, \ldots, z_r$ be the vectors returned by ApproxLowRank(A, r, ε). Then ApproxLowRank returns the matrix $Y = ZZ^T A$ and the subspace U spanned by the vectors z_i and $z_i^T A$ (it is easy to see that Y is indeed the projection of A onto U). Both operations can be performed in time $O(d_1 d_2 r)$. Hence the overall running time is dominated by the invocation of ApproxSVD, which leads to the running time stated in the theorem.

It remains to prove the desired head and tail approximation ratios. The tail approximation guarantee follows directly from Equation (7.2). For the head approximation, first note that Equation (7.1) implies

$$z_i^T A A^T z_i \;\geq\; (1-\varepsilon)\sigma_i^2 \;.$$

We now apply this inequality by rewriting the head quantity $\|ZZ^T A\|_F^2$ as follows:

$$\begin{aligned}
\|ZZ^T A\|_F^2 &= \operatorname{tr}(A^T ZZ^T ZZ^T A) \\
&= \operatorname{tr}(A^T ZZ^T A) \\
&= \operatorname{tr}(A^T (\sum_{i=1}^{r} z_i z_i^T) A) \\
&= \operatorname{tr}(\sum_{i=1}^{r} A^T z_i z_i^T A) \\
&= \sum_{i=1}^{r} \operatorname{tr}(A^T z_i z_i^T A) \\
&= \sum_{i=1}^{r} \operatorname{tr}(z_i^T A A^T z_i) \\
&= \sum_{i=1}^{r} z_i^T A A^T z_i \\
&\geq (1-\varepsilon) \sum_{i=1}^{r} \sigma_i^2 \\
&= (1-\varepsilon) \|A_r\|_F^2
\end{aligned}$$

where the matrix A_r is the best rank-r approximation of the matrix A. This proves the desired head approximation guarantee. □

Since the approximation factor ε only enters the running time in the approximate SVD, we can set the approximation ratios to small constants. This allows us to combine our approximate projections with our variant of projected gradient descent.[1] In the next section, we give a concrete examples. Empirically, we will see in Section 7.4 that a very small number of iterations in ApproxSVD already suffices for accurate recovery.

7.3 A recovery algorithm

Before we state our formal recovery result, we briefly recap the relevant context.

7.3.1 Preliminaries

As before, our goal is to recover an unknown, structured vector $\theta^* \in \mathbb{R}^d$ from linear observations of the form

$$y = X\theta^* + e \,, \tag{7.3}$$

where the vector $y \in \mathbb{R}^n$ contains the linear observations / measurements, the matrix $X \in \mathbb{R}^{n \times d}$ is the design / measurement matrix, and the vector $e \in \mathbb{R}^n$ is an arbitrary noise

[1] We remark that our definitions require head and tail projections to be *deterministic*, while the approximate SVD is *randomized*. However, the running time of ApproxSVD depends only logarithmically on the failure probability, and it is straightforward to apply a union bound over all iterations of projected gradient descent. Hence we ignore these details here to simplify the presentation.

vector.

To keep the setup simple, we study a compressive sensing version of this problem where the formal goal is to find an estimate $\hat{\theta} \in \mathbb{R}^d$ such that

$$\left\|\hat{\theta} - \theta^*\right\|_2 \;\leq\; c \cdot \|e\|_2 \;.$$

Here, the scalar c is a fixed, universal constant and $\|\cdot\|_2$ is the standard ℓ_2-norm. We assume that the vector θ^* represents the coefficients of a low-rank matrix. To be precise, we convert a matrix $\Theta \in \mathbb{R}^{d_1 \times d_2}$ into a vector by concatenating the rows (or columns) to form a vector of dimension $d = d_1 \cdot d_2$.

Restricted isometry property. Next, we introduce a variant of the well-known restricted isometry property (RIP) for subspace models. The RIP is a common regularity assumption for the design matrix X that is often used in compressive sensing and low-rank matrix recovery in order to decouple the analysis of algorithms from concrete sampling bounds. Formally, we have:

Definition 7.6 (Subspace RIP)**.** *Let $X \in \mathbb{R}^{n \times d}$, let $\mathbb{U}$ be a subspace model, and let $\delta \geq 0$. Then X satisfies the $(\mathbb{U}, \delta)$-subspace RIP if for all $\theta \in \mathcal{M}(\mathbb{U})$ we have*

$$(1-\delta)\|\theta\|^2 \leq \|X\theta\|^2 \leq (1+\delta)\|\theta\|^2 \;.$$

In the next section, we provide such a measurement matrix that combines good sample complexity with fast multiplication time (which important in iterative algorithms such as ours).

7.3.2 A fast measurement matrix for low-rank structure

Here, we establish bounds on the sample complexity of recovering a low-rank matrix with a particular focus on *fast* measurement matrices. In particular, we are interested in measurement matrices that support matrix-vector multiplications with a running time that is *nearly-linear* in the size of the vector. Our results follow from a concatenation of previous results in this area.

Consider the case where the subspace model $\mathbb{U}$ corresponds to the set of rank-r matrices of size $d_1 \times d_1$. Then, the subspace RIP corresponds to the *rank-r restricted isometry property*, first introduced by Recht, Fazel, and Parrilo [186]. A standard Gaussian matrix ensemble satisfies the rank-r RIP. But since such matrices are dense and unstructured, they have a strongly superlinear running time of $O(n \cdot d)$. To improve over the time complexity, we establish the following result:

Theorem 7.7. *Let $d = d_1^2$. Then, there exists a randomized construction of a matrix $X \in \mathbb{R}^{n \times d}$, with parameters $n = O(r \cdot d \cdot \text{polylog} d)$, such that X satisfies the rank-r RIP with high probability. Moreover, X supports matrix-vector multiplications with complexity $O(d \log d)$.*

Proof. We begin by considering matrices that satisfy the *standard* RIP for s-sparse vectors, as well as support fast matrix-vector multiplication. To the best of our knowledge, the sharpest such bounds have been recently obtained by [115]. They show that with high probability, a matrix formed by randomly subsampling

$$n = O\Big(\delta^{-2} s \log^2(s/\delta) d\Big)$$

rows of the discrete Fourier Transform (DFT) matrix satisfies the standard RIP (with isometry constant δ) over the set of s-sparse vectors.

Next, we invoke a well-known result by Krahmer and Ward [142]. Consider a diagonal matrix D_ξ, where the diagonal ξ is a Rademacher sequence uniformly distributed over $\{-1,1\}^d$. Also consider any fixed set of vectors B with $|B| = m$ where $s > O(\log \frac{m}{\eta})$. If X' is any $n \times d$ matrix that satisfies the standard RIP over the set of s-sparse vectors with constant $\delta < \varepsilon/4$, then high probability the matrix $X = X' D_\xi$ is a *Johnson-Lindenstrauss* embedding for E. Formally, the following is true with probability exceeding $1 - \eta$:

$$(1-\varepsilon)\|\beta\|_2^2 \leq \|X\beta\|_2^2 \leq (1+\varepsilon)\|\beta\|_2^2.$$

uniformly for all $\beta \in B$.

Next, we invoke Lemma 3.1 of [58], who show that the set of vectors corresponding to rank-k matrices, S_k, exhibits an ϵ-net $\bar{S}_k$ (with respect to the Euclidean norm) such that

$$|\bar{S}_r| \leq (9/\epsilon)^{(d_1+d_2+1)k}.$$

Also from [58], we have that if X is a Johnson-Lindenstrauss embedding with isometry constant ε for an $\bar{S}_k$, then X satisfies the rank-k RIP with constant $\delta = O(\varepsilon)$. Plugging in $s = O(k(d_1 + d_2))$ and $m = O(s \mathrm{polylog} d)$ and adjusting constants, we get the stated result. □

7.3.3 Recovery result

We now prove our overall result for low-rank matrix recovery.

Theorem 7.8. *Let $X \in \mathbb{R}^{n \times d}$ be a matrix with RIP for low-rank matrices, and let T_X denote the time to multiply a d-dimensional vector with X or X^T. Then there is an algorithm that recovers an estimate $\hat{\theta}$ such that*

$$\|\hat{\theta} - \theta^*\| \ \leq \ c\|e\|$$

Moreover, the algorithm runs in time $\widetilde{O}(T_X + r \cdot d_1^2)$.

Since the proof is a direct consequence of our general results in Chapter 2, we omit the details here.

Up to constant factors, the requirements on the RIP of X in Theorem 7.8 are the same as for exact projections. As a result, our sample complexity is only affected by a constant factor through the use of approximate projections. Our experiments in Section 7.4 show that the empirical loss in sample complexity is negligible. Similarly, the number of iterations $O(\log\|\theta\|/\|e\|)$ is also only affected by a constant factor compared to the use of exact projections [50, 131].

In the regime where multiplication with the matrix X is fast, the time complexity of the projection dominates the time complexity of the recovery algorithms. Here, our algorithm runs in time $\widetilde{O}(r \cdot d_1^2)$, which is the first provable running time faster than the $O(d_1^3)$ bottleneck given by a single exact SVD. While prior work has suggested the use of approximate SVDs in low-rank matrix recovery [83], our results are the first that give a provably better time complexity for this combination of projected gradient descent and approximate SVDs. Hence Theorem 7.8 can be seen as a theoretical justification for the heuristic use of approximate SVDs.

Finally, we remark that Theorem 7.8 does not directly cover the low-rank matrix completion case because the subsampling operator does not satisfy the low-rank RIP [83]. To clarify our use of approximate SVDs, we focus on the RIP setting in our proofs, similar to recent work on low-rank matrix recovery [74, 223]. We believe that similar results as for SVP [131] also hold for our algorithm. Our experiments in the next section also show that our algorithm works well for low-rank matrix completion.

7.4 Experiments

We now investigate the empirical performance of our proposed algorithms. All experiments were conducted on an iMac desktop computer with an Intel Core i5 CPU (3.2 GHz) and 16 GB RAM. With the exception of the dynamic program (DP) for 2D histograms, all code was written in Matlab. Since the Krylov SVD of [165] is only available as a Matlab routine, we also chose the Matlab version of PROPACK [147] so that the implementations are comparable. Unless reported otherwise, all reported data points were averaged over at least 10 trials.

Considering our theoretical results on approximate projections for low-rank recovery, one important empirical question is how the use of approximate SVDs such as [165] affects the sample complexity of low-rank matrix recovery. For this, we perform a standard experiment and use several algorithms to recover an image of the MIT logo from subsampled Fourier measurements (c.f. Section 7.3.2). The MIT logo has also been used in prior work [131, 186]. We use an image with dimensions 200×133 and rank 6 (see Figure 7-2). We limit our attention here to variants of SVP because the algorithm has good empirical performance and has been used as baseline in other works on low-rank recovery.

Figure 7-1 shows that SVP / IHT combined with a single iteration of a block Krylov SVD [165] achieves the same phase transition as SVP with exact SVDs. This indicates that the use of approximate projections for low-rank recovery is not only theoretically sound but can also lead to practical algorithms. In Figure 7-3, we also show corresponding running time results demonstrating that the block Krylov SVD also leads to the fastest recovery algorithm.

We also study the performance of approximate SVDs for the matrix completion problem. We generate a symmetric matrix of size 2048×2048 with rank $r = 50$ and observe a varying number of entries of the matrix. The approximation errors of the various algorithms are again comparable. Figure 7-1 shows the resulting running times for several sampling ratios. Again, SVP combined with a block Krylov SVD [165] achieves the best running time. Depending on the oversampling ratio, the block Krylov approach (now with two iterations)

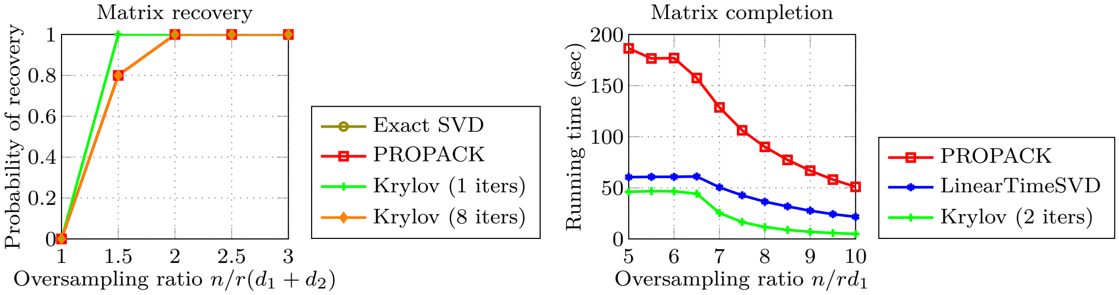


Figure (7-1): Left: Results for a low-rank matrix recovery experiment using subsampled Fourier measurements. SVP / IHT with one iteration of a block Krylov SVD achieves the same phase transition as SVP with an exact SVD. Right: Results for a low-rank matrix completion problem. SVP / IHT with a block Krylov SVD achieves the best running time and is about 4 – 8 times faster than PROPACK.

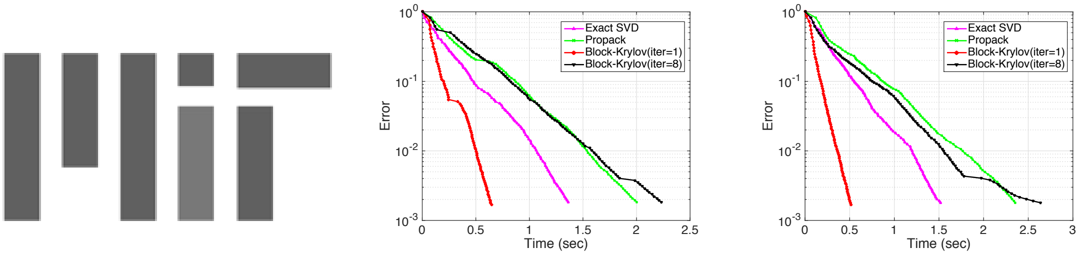


Figure (7-2): (left) Example low-rank matrix of size $d = 133 \times 200, r = 6$. (right) Recovery error of various algorithms as a function of time (2 independent trials).

is 4 to 8 times faster than SVP with PROPACK.

7.4.1 Further description of the experiments

Figure 7-2 shows an image of the MIT logo used in the low-rank matrix recovery experiments [131, 186]. For our first experiment, we record $n = 3.5(d_1 + d_2)r = 6994$ linear measurements of the image. The measurement operator is constructed by subsampling m rows of a Fourier matrix and multiplying its columns by a randomly chosen Bernoulli vector, similar to the RIP matrix given in Section 7.3.2. The goal is to recover the image from these observations.

We adapt the Singular Value Projection (SVP) algorithm of [131] by replacing the exact SVD step with approximate SVDs (some of which are very coarse), and demonstrate that we can still achieve efficient matrix recovery from few observations. As alternatives to Matlab's in-built `svd` function, we include the PROPACK [147] numerical linear algebra package, which implements a Lanczos-type method. We also include an implementation of the recent Block-Krylov SVD algorithm of [165], which offers a nice tradeoff between approximation ratio and running time. We test this method with 1 and 8 Krylov subspace iterations (8 is the default provided in the code of [165]).

Figure 7-3 shows the running times corresponding to the phase transition plot in Figure 7-1. The only stopping criteria we used were based on a small residual and a maximum number

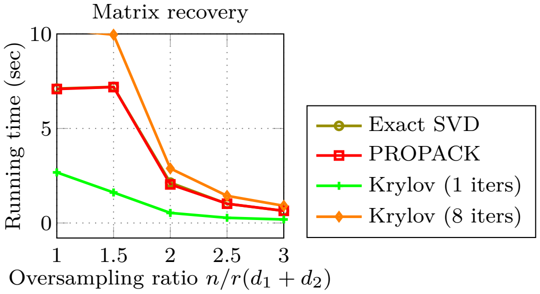


Figure (7-3): Running times corresponding to the low-rank matrix recovery experiment in Figure 7-1. The block Krylov variant of IHT with one iteration has the best running time.

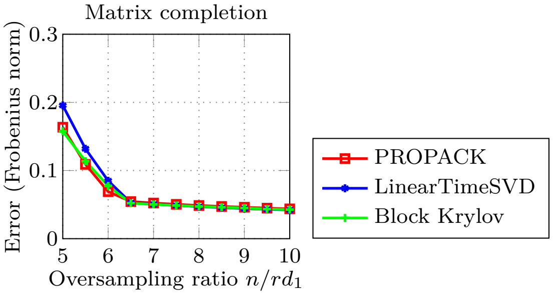


Figure (7-4): Average approximation errors for the low-rank matrix completion experiment in Figure 7-1. As for low-rank matrix recovery, the different SVDs achieve essentially the same error.

of iterations, so the running times of the algorithms are slowest in the regime where they do not recovery the signal.

The subspace IHT algorithm is iterative, i.e., it produces a sequence of matrix estimates $\{\hat{\theta}^0, \hat{\theta}^1, \ldots, \hat{\theta}^t\}$. Figure 7-2 displays the estimation error, $\frac{\|\theta^* - \hat{\theta}^t\|}{\|\theta^*\|)}$, as a function of wall-clock time, on two different trial runs. We observe from the plots that PROPACK and the Block Krylov method (with 8 iterations) perform similar to the exact SVD due to the small problem size. Interestingly, a *very coarse* approximate SVD (a single Block Krylov subspace iteration) provides the fastest convergence. Overall, using approximate SVDs within SVP / IHT does not only yield computational speed-ups, but also offers competitive statistical performance.

We also report results of using the SVP / IHT algorithm with approximate projections on a larger matrix completion problem. We generate a matrix of size $d = 2048 \times 2048$ with rank $r = 50$. We only sample n randomly chosen entries of this matrix and attempt to reconstruct the matrix from these entries using SVP with approximate low-rank projections. We vary n and obtain error curves as well as running times. Figure 7-4 shows the approximation errors for the matrix completion experiment in Figure 7-1. As for the matrix recovery experiments, all SVDs achieve essentially the same error. We note that the error floor of about 0.05 is a result of our stopping criterion.

Chapter 8

Histograms

8.1 Introduction

Next, we introduce fast approximate projections for *2D-histograms*, another natural model of data. As in the low-rank case, we think of the vector $\theta^* \in \mathbb{R}^d$ as a matrix $\Theta \in \mathbb{R}^{d_1 \times d_1}$ and assume the square case for simplicity (again, our results also apply to rectangular matrices). We say that Θ is a k-histogram if the coefficients of Θ can be described as k axis-aligned rectangles on which Θ is constant.

This definition is a generalization of 1D-histograms to the two-dimensional setting and has found applications in several areas such as databases and density estimation. Moreover, the theoretical computer science community has studied sketching and streaming algorithms for histograms, which is essentially the problem of recovering a histogram from linear observations. Similar to low-rank matrices in the previous chapter, 2D-histograms also do not fall into the structured sparsity framework of earlier Chapters.

Due to their piecewise constant structure, a natural representation for histograms is a wavelet basis. Indeed, combining Haar wavelets with tree-structured sparsity gives the correct sample complexity of $n = O(k \log d)$ for 1D-histograms. However, the wavelet approach incurs a *suboptimal* sample complexity of $O(k \log^2 d)$ for 2D-histograms. Note that it is information-theoretically possible to achieve a sample complexity of $O(k \log d)$. But unfortunately, the corresponding (exact) projection requires a complicated dynamic program (DP) with time complexity $O(d_1^5 k^2)$, which is impractical for all but very small problem dimensions [167].

In this chapter, we design significantly faster *approximate* projection algorithms for 2D histograms. Our approach is based on an approximate DP [167] that we combine with a Lagrangian relaxation of the k-rectangle constraint. Both algorithms have parameters for controlling the trade-off between the size of the output histogram, the approximation ratio, and the running time. As mentioned in Section 7.1.2, the bicriterion nature of our approximate head and tail guarantees becomes useful here. In the following two theorems, we let $\mathbb{U}_k$ be the subspace model of 2D histograms consisting of k-rectangles (see Section 7.1.2 for the definition of a subspace model).

Theorem 8.1. *Let $\zeta > 0$ and $\varepsilon > 0$ be arbitrary. Then there is an $(1+\varepsilon, \mathbb{U}_k, \mathbb{U}_{c\cdot k})$-approximate tail projection for 2D histograms where $c = O(1/\zeta^2\varepsilon)$. Moreover, the algorithm runs in time $\widetilde{O}(d^{1+\zeta})$.*

Theorem 8.2. *Let $\zeta > 0$ and $\varepsilon > 0$ be arbitrary. Then there is an $(1-\varepsilon, \mathbb{U}_k, \mathbb{U}_{c\cdot k})$-approximate head projection for 2D histograms where $c = O(1/\zeta^2\varepsilon)$. Moreover, the algorithm runs in time $\widetilde{O}(d^{1+\zeta})$.*

Note that both algorithms offer a running time that is *almost linear*. The small polynomial gap to a linear running time can be controlled as a trade-off between computational and statistical efficiency (a larger output histogram requires more samples to recover).

While we provide rigorous proofs for the approximation algorithms as stated above, we remark that we do not establish an overall recovery result similar to previous structured sparsity models or low-rank matrices. The reason is that the approximate head projection is competitive with respect to k-histograms, but not with the space $\mathbb{U}_k + \mathbb{U}_k$, i.e., the Minkowski sum of two k-histogram subspaces. The details are somewhat technical and we give a more detailed discussion in Section 8.3.1. However, under a natural structural conjecture about sums of k-histogram subspaces, we can also obtain a recovery result similar to Theorem 7.8. Moreover, we experimentally demonstrate that the sample complexity of our algorithms already improves over wavelets for k-histograms of size 32×32 (see Section 8.4).

Finally, we note that our DP approach also generalizes to γ-dimensional histograms for any constant $\gamma \geq 2$. As the dimension of the histogram structure increases, the gap in sample complexity between our algorithm and the prior wavelet-based approach becomes increasingly wide and scales as $O(k\gamma \log d)$ vs $O(k \log^\gamma d)$. For simplicity, we limit our attention to the 2D case described above.

8.1.1 Related work

1D and 2D histograms have been studied extensively in several areas such as databases [78, 128] and density estimation. Histograms are typically used to summarize "count vectors", with each coordinate of the vector θ corresponding the number of items with a given value in some data set. Computing linear sketches of such vectors, as well as efficient methods for recovering histogram approximations from those sketches, became key tools for designing space efficient dynamic streaming algorithms [105, 106, 220]. For 1D histograms it is known how to achieve the optimal sketch length bound of $n = O(k \log d)$: it can be obtained by representing k-histograms using a tree of $O(k \log d)$ wavelet coefficients as in [105] and then using the structured sparse recovery algorithm of [36]. However, applying this approach to 2D histograms leads to a sub-optimal bound of $O(k \log^2 d)$.

8.2 Approximation algorithms for 2D histograms

We now describe our approximate head and tail projections for histograms. One key ingredient in our algorithms are *hierarchical* histograms. Overall, our goal is to approximate arbitrary 2D histograms, i.e., arbitrary partitions of a $\sqrt{d} \times \sqrt{d}$ matrix with k non-overlapping rectangles (for simplicity, we limit our attention to the case of square matrices). Such histograms are also known as *tiling* histograms. However, tiling histograms are hard to work with algorithmically because they do not allow a clean decomposition for a dynamic program. Instead, work in histogram approximation has utilized hierarchical histograms, which are also partitions of a matrix into k non-overlapping rectangles. The additional restriction is that the partition can be represented as a tree in which each rectangle arises through a vertical or horizontal split of the parent rectangle. We refer the reader to [167] for a more detailed description of different histogram types.

An important result is that every tiling histogram consisting of k rectangles can be simulated with a hierarchical histogram consisting of at most $4k$ rectangles (d'Amore and Franciosa, 1992). Since Theorems 8.1 and 8.2 provide bicriterion guarantees for the output space, i.e., projections into a space of histograms consisting of $O(k)$ rectangles, we focus our attention on approximation algorithms for hierarchical histograms in the following. These results can then easily be converted into statements for tiling histograms by increasing the number of histogram tiles by 4.

Next, we introduce some histogram-specific notation. For a histogram subspace U, we denote the number of histogram pieces in U with $\gamma(U)$. We denote the set of hierarchical histograms subspaces with $\mathscr{H}_{\mathrm{h}}$. When we have an upper bound on the number of histogram pieces, we write $\mathscr{H}_{\mathrm{h},k}$ for the set of hierarchical histogram subspaces U with $\gamma(U) \leq k$.

An important subroutine in our approximate projections is the following notion of a hierachical histogram oracle.

Definition 8.3. *An (α, ζ)-hierarchical histogram oracle is an algorithm with the following guarantee: given any $b \in \mathbb{R}^{\sqrt{d}\times\sqrt{d}}$ and $\lambda \in \mathbb{R}$ as input, the algorithm returns a hierarchical histogram subspace U such that*

$$\|P_U b\|^2 - \frac{\lambda}{\alpha}\gamma(U) \;\geq\; \max_{U' \in \mathscr{H}_{\mathrm{h}}} \|P_{U'} b\|^2 - \lambda\gamma(U') \; . \tag{8.1}$$

Moreover, the algorithm runs in time $O(d^{1+\zeta})$.

An algorithm with the following guarantee directly follows from the hierarchical dynamic programming techniques introduced in [167]. In particular, Theorem 3 of [167] implies a dependence of $\alpha = O(1/\zeta^2)$.

Equation (8.1) has the flavor of a head approximation (a max-quantified guarantee). As a direct consequence of Equation (8.1), we also get the following "tail approximation" variant.

Lemma 8.4. *The solution U returned by an (α, ζ)-hierarchical histogram oracle also satisfies*

$$\|b - P_U b\|^2 + \frac{\lambda}{\alpha}\gamma(U) \;\leq\; \min_{U' \in \mathscr{H}_{\mathrm{h}}} \|b - P_{U'} b\|^2 + \lambda\gamma(U') \; . \tag{8.2}$$

Proof. Multiplying both sides of Equation (8.1) with -1 and pulling the negative sign into the max gives

$$-\|P_U b\|^2 + \frac{\lambda}{\alpha}\gamma(H) \;\leq\; \min_{U' \in \mathscr{H}_{\mathrm{h}}} -\|P_{U'} b\|^2 + \lambda\gamma(U') \; .$$

Adding $\|b\|^2$ to both sides and using that P_U and $P_{U'}$ are orthogonal projections then gives Equation (8.2) via the Pythagorean Theorem. □

Algorithm 27 Tail projection for hierarchical histograms

```
function HistogramTail($b, k, \nu, \xi$)
    $\Delta \leftarrow \min\{|b_{i,j} - b_{i',j'}| \mid b_{i,j} - b_{i',j'} \neq 0\}$
    $\varepsilon_{\min} \leftarrow \frac{\Delta^2}{d^2}$
    $\lambda_0 \leftarrow \frac{\varepsilon_{\min}}{2k}$
    $U_0 \leftarrow$ HistogramOracle($b, \lambda_0$)
    if $\|b - P_{U_0}\| = 0$ and $\gamma(U_0) \leq \alpha k$ then
        return $U_0$
    end if
    $\lambda_l \leftarrow 0$
    $\lambda_r \leftarrow 2\alpha\|b\|$
    $\varepsilon \leftarrow \frac{\varepsilon_{\min}\xi}{k}$
    while $\lambda_r - \lambda_l \geq \varepsilon$ do
        $\lambda_m \leftarrow \frac{\lambda_l + \lambda_r}{2}$
        $U_m \leftarrow$ HistogramOracle($b, \lambda_m$)
        if $\gamma(U_m) \geq \alpha k$ and $\gamma(U_m) \leq \nu\alpha k$ then
            return $U_m$
        end if
        if $\gamma(U_m) \geq \nu\alpha k$ then
            $\lambda_l \leftarrow \lambda_m$
        else
            $\lambda_r \leftarrow \lambda_m$
        end if
    end while
    return HistogramOracle($b, \lambda_r$)
end function
```

However, note that neither Equation (8.1) nor (8.2) give direct control over the number of histogram pieces k. In the following, we give algorithms that convert these guarantees into approximate projections. In a nutshell, we show that carefully choosing the trade-off parameter λ, combined with a postprocessing step of the corresponding solution, yields head an tail approximations.

8.2.1 Approximate tail projection

We now show how to construct an approximate tail projection from a hierarchical histogram oracle. In the following, we assume that HistogramOracle(b, λ) is an (α, ζ)-hierarchical histogram oracle.

First, we establish a lower bound on the approximation error $\|b - P_U\|^2$ if b is not in the histogram subspace U.

Lemma 8.5. *Let $b \in \mathbb{R}^d$ and U be a histogram subspace. If $b \notin U$, then we have $\|b - P_U\|^2 \geq \varepsilon_{\min}$ where $\varepsilon_{\min}$ is as defined in Algorithm 27.*

Proof. If $b \notin U$, there is a histogram piece in U on which b is not constant. Let R be the set of indices in this piece. We now give a lower bound on the projection error based on the histogram piece R (recall that the projection of b onto U averages b in each histogram piece):

$$\|b - P_U\|^2 \;\geq\; \sum_{(i,j)\in R} (b_{i,j} - \bar{b}_R)^2 \qquad \text{where} \quad \bar{b}_R = \frac{1}{|R|} \sum_{(i,j)\in R} b_{i,j} \;.$$

Let (i^*, j^*) be the index of the largest coefficient in the histogram piece R (ties broken arbitrarily). Then we bound the sum on the right hand side above with the term corresponding to (i^*, j^*):

$$\|b - P_U\|^2 \;\geq\; \left(b_{i^*,j^*} - \frac{1}{|R|} \sum_{(i,j)\in R} b_{i,j} \right)^2 \;.$$

Let Δ_R be the smallest non-zero difference between coefficients in R. Note that $\Delta_R > 0$ because b is not constant on R. Moreover, we have $\Delta_R \leq \max_{(i,j)\in R} b_{i^*,j^*} - b_{i,j}$. Hence we get

$$\left(b_{i^*,j^*} - \frac{1}{|R|} \sum_{(i,j)\in R} b_{i,j} \right)^2 \;\geq\; \left(b_{i^*,j^*} - \frac{|R|-1}{|R|} b_{i^*,j^*} - \frac{1}{|R|}(b_{i^*,j^*} - \Delta_R) \right)^2$$

because b_{i^*,j^*} is one of the largest coefficients in R and at least one coefficient is smaller than b_{i^*,j^*} by at least Δ_R. Combining the inequalities above and simplifying then yields

$$\|b - P_U\|^2 \;\geq\; \frac{\Delta_R^2}{|R|^2} \;\geq\; \frac{\Delta^2}{d^2} \;=\; \varepsilon_{\min} \;. \qquad \square$$

Next, we prove that the histogram oracle returns roughly a k-histogram if the input is a k-histogram and we set the parameter λ correctly.

Lemma 8.6. *Let $\varepsilon_{\min}$ and λ_0 be defined as in Algorithm 27. If b is a hierarchical k-histogram, then* HistogramOracle(b, λ_0) *returns a hierarchical histogram subspace U_0 such that $b \in U_0$ and $\gamma(U_0) \leq \alpha k$.*

Proof. First, we show that $b \in U_0$, i.e., that $\|b - P_{U_0}\| = 0$. Since $b \in \mathscr{H}_{\mathrm{h},k}$, we know that there is a hierarchical histogram subspace U' such that $\|b - P_{U'}\| = 0$ and $\gamma(U') \leq k$. Substituting this histogram subspace U' and λ_0 into Equation (8.2) gives

$$\|b - P_{U_0}\|^2 \;\leq\; \lambda_0 \gamma(U') \;\leq\; \frac{\varepsilon_{\min}^2}{2}$$

where we also used that $\frac{\lambda_0}{\alpha}\gamma(U_0) \geq 0$. Since $\varepsilon_{\min} > 0$, the contrapositive of Lemma 8.5 shows that $b \in U_0$.

Next, we prove that $\gamma(U_0) \leq \alpha k$. Substituting into Equation (8.2) again and using $\|b - P_{U_0}\| = 0$ now gives the desired bound on the number of histogram pieces:

$$\frac{\lambda_0}{\alpha}\gamma(U_0) \;\leq\; \lambda_0 k \;. \qquad \square$$

With these preliminaries in place, we now show the main result for our tail approximation algorithm.

Theorem 8.7. *Let $b \in \mathbb{R}^d$, $k \in \mathbb{N}$, $\nu > 1$, and $\xi > 0$. Then* HistogramTail(b, k, ν, ξ) *returns a histogram subspace U such that $\gamma(U) \leq \nu\alpha k$ and*

$$\|b - P_U b\|^2 \ \leq \ \left(1 + \frac{1}{\nu - 1} + \xi\right) \min_{U' \in \mathscr{H}_{\mathrm{h},k}} \|b - P_{U'} b\|^2 \ .$$

Moreover, the algorithm runs in time

$$O\left(n^{1+\zeta} \log\left(\frac{\alpha d \|b\|}{\xi \Delta}\right)\right)$$

where Δ is as defined in Algorithm 27.

Proof. We analyze the three cases in which HistogramTail returns separately. First, consider Line 7. In this case, U_0 clearly satisfies the conditions of the theorem. So in the following, we condition on the algorithm not returning in Line 7. By the contrapositive of Lemma 8.6, this implies that $b \notin \mathcal{M}(\mathscr{H}_{\mathrm{h},k})$.

Next, consider the case that HistogramTail returns in Line 16. This directly implies that $\gamma(U_m) \leq \nu\alpha k$. Moreover, substituting into Equation 8.2 and restricting the right hand side to histogram subspaces with at most k pieces gives

$$\begin{aligned}
\|b - P_{U_m}\|^2 + \frac{\lambda_m}{\alpha}\gamma(U_m) \ &\leq \ \min_{U' \in \mathscr{H}_{\mathrm{h},k}} \|b - P_{U'} b\|^2 + \lambda_m \gamma(U') \\
\|b - P_{U_m}\|^2 \ &\leq \ \min_{U' \in \mathscr{H}_{\mathrm{h},k}} \|b - P_{U'} b\|^2 + \lambda_m \gamma(U') - \lambda_m k \\
\|b - P_{U_m}\|^2 \ &\leq \ \min_{U' \in \mathscr{H}_{\mathrm{h},k}} \|b - P_{U'} b\|^2
\end{aligned}$$

where we used that $\gamma(U_m) \geq \alpha k$ and $\gamma(U') \leq k$.

For the remaining case (Line 24), we use the following shorthands in order to simplify notation: Let U_l and U_r be the histogram subspaces returned by HistogramOracle with parameters λ_l and λ_r, respectively. We denote the corresponding tail errors with $t_l = \|b - P_{U_l}\|^2$ and $t_r = \|b - P_{U_r}\|^2$. Moreover, we denote the optimal tail error with $t^* = \min_{U' \in \mathscr{H}_{\mathrm{h},k}} \|b - P_{U'} b\|^2$. Finally, let $\gamma_l = \gamma(U_l)$ and $\gamma_r = \gamma(U_r)$ be the number of histogram pieces in the respective histogram subspaces. Rewriting Equation 8.2 in terms of the new notation gives

$$t_l + \frac{\lambda_l}{\alpha}\gamma_l \ \leq \ t^* + \lambda_l k \tag{8.3}$$

$$t_r + \frac{\lambda_r}{\alpha}\gamma_r \ \leq \ t^* + \lambda_r k \tag{8.4}$$

We will use Equation 8.3 in order to bound our tail projection error t_l. For this, we establish an upper bound on λ_r. Note that $\lambda_r \leq \lambda_l + \varepsilon$ when the algorithm reaches Line 24. Moreover, the binary search over λ is initialized so that we always have $\gamma_l > \nu\alpha k$ and $\gamma_r < \alpha k$.

Combining these facts with Equation (8.4) leads to an upper bound on λ_l:

$$\begin{aligned} t_l + \frac{\lambda_l}{\alpha}\gamma_l \;&\leq\; t^* + \lambda_l k \\ \frac{\lambda_l}{\alpha}\nu\alpha k \;&\leq\; t^* + \lambda_l k \\ \lambda_l \;&\leq\; \frac{t^*}{(\nu-1)k} \; . \end{aligned}$$

We use these facts in order to establish an upper bound on t_r. Substituting into Equation (8.4) gives

$$\begin{aligned} t_r \;&\leq\; t^* + (\lambda_l + \varepsilon)k \\ t_r \;&\leq\; t^* + \frac{t^*}{\nu-1} + \frac{\varepsilon_{\min}\xi}{k}k \\ t_r \;&\leq\; \left(1 + \frac{1}{\nu-1} + \xi\right)t^* \end{aligned}$$

where we used that $t^* \geq \varepsilon_{\min}$ because b is not a hierarchical k-histogram if the algorithm reaches Line 24 (see Lemma 8.5). Combined with the fact that $\gamma_r \leq \alpha k$, this proves the statement of the theorem.

Finally, we consider the running time bound. It is straightforward to see that the overall running time is dominated by the invocations of HistogramOracle, each of which takes $O(d^\zeta)$ time. The number of iterations of the binary search is bounded by the initial gap between λ_l and λ_r and the final gap ε, which gives an iteration bound of

$$\lceil \log \frac{\lambda_r^{(0)} - \lambda_l^{(0)}}{\varepsilon} \rceil \;=\; O\left(\log\left(\frac{\alpha d^2 k \|b\|}{\xi \Delta^2}\right)\right) \; .$$

Simplifying and multiplying this iteration bound with the running time of the subroutine HistogramOracle leads to the running time bound stated in the theorem. □

Theorem 8.1 now follows directly from Theorem 8.7. We first restate Theorem 8.1:

Theorem 8.1. *Let $\zeta > 0$ and $\varepsilon > 0$ be arbitrary. Then there is an $(1+\varepsilon, \mathbb{U}_k, \mathbb{U}_{c\cdot k})$-approximate tail projection for 2D histograms where $c = O(1/\zeta^2\varepsilon)$. Moreover, the algorithm runs in time $\widetilde{O}(d^{1+\zeta})$.*

Setting $\xi = O(\varepsilon)$ and $\nu = O(1/\varepsilon)$ gives the $1+\varepsilon$ guarantee in Theorem 8.1. Moreover, we use the $\alpha = O(1/\zeta^2)$ dependence from Theorem 3 of [167].

8.2.2 Approximate head projection

Next, we show how to construct an approximate head projection from a hierarchical histogram oracle. Similar to the approximate tail projection above, we perform a binary search over the parameter λ in order achieve a good trade-off between sparsity and approximation. In contrast to the tail case, we now need an additional subroutine for extracting a "high-density" sub-histogram of a given hierarchical histogram. We reduce this task of extracting

Algorithm 28 Head projection for hierarchical histograms

```
function HistogramHead($b, k, \tau$)
    $b_{\max} \leftarrow \max_{b_{i,j}} |b_{i,j}^2|$
    $\lambda_l \leftarrow \frac{b_{\max}\tau}{k}$
    $U_l \leftarrow$ HistogramOracle($b, \lambda_l$)
    if $\gamma(U_l) \le \frac{2\alpha}{\tau}k$ then
        return $U_l$
    end if
    $\lambda_r \leftarrow 2\alpha\|b\|^2$
    $\varepsilon \leftarrow \frac{b_{\max}\tau}{2k}$
    while $\lambda_r - \lambda_l > \varepsilon$ do
        $\lambda_m \leftarrow \frac{\lambda_l + \lambda_r}{2}$
        $U_m \leftarrow$ HistogramOracle($b, \lambda_m$)
        if $\gamma(U_m) > \frac{2\alpha}{\tau}k$ then
            $\lambda_l \leftarrow \lambda_m$
        else
            $\lambda_r \leftarrow \lambda_m$
        end if
    end while
    $U_l \leftarrow$ HistogramOracle($b, \lambda_l$)
    $U_r \leftarrow$ HistogramOracle($b, \lambda_r$)
    $U_l' \leftarrow$ FindSubhistogram($b, U_l, \frac{2\alpha}{\tau}k$)
    if $\left\|P_{U_l'} b\right\|^2 \ge \|P_{U_r} b\|^2$ then
        return $U_l'$
    else
        return $U_r$
    end if
end function
```

a sub-histogram to a problem on trees. Formally, we build on the following lemma about the subroutine FindSubtree.

Lemma 8.8. *Let $T = (V, E)$ be a tree with node weights $w : V \to \mathbb{R}$. Moreover, let $s \le |V|$ be the target subtree size. Then* FindSubtree(T, w, s) *returns a node subset $V' \subseteq V$ such that V' forms a subtree in T, its size is at most $2s$, and it contains a proportional fraction of the node weights, i.e., $\sum_{i \in V'} w(i) \ge \frac{s}{|V|} \sum_{i \in V} w(i)$.*

Proof. Let w' and i be defined as in FindSubtree. An averaging argument shows that there must be a contiguous subsequence S as defined in FindSubtree with

$$\sum_{j=i}^{i+2s-1} w'(j) \;\ge\; \frac{2s}{2|V|-1} \sum_{j=1}^{2|V|-1} w'(j) \;\ge\; \frac{s}{|V|} \sum_{j \in V} w(j)$$

where the first inequality holds because S contains $2s$ nodes, and the second inequality holds by the construction of the tour W.

Let V' be the nodes in S. Note that we have defined w' such that every node weight is used

Algorithm 29 Subroutines for the head projection

```
function FindSubhistogram($(b, U, s)$)
    Let $T_U = (V_U, E_U)$ be a tree corresponding to the histogram subspace $U$.
    Let $w : V_U \to \mathbb{R}$ be the node weight function corresponding to $U$ and $b$.
    Let $T_U^*$ be the tree $T_U$ with an additional root node $r$.
    Let $w^*$ be defined as $w$ with the root node weight $w^*(r) = \|P_{R_0} b\|^2$.
    $V' \leftarrow$ FindSubtree($T_U^*, w^*, s$)
    if $r \in V'$ then
        return the sub-histogram defined by the splits in $V'$
    else
        Let $r'$ be the root node in the subtree defined by $V'$.
        Let $U''$ be a 4-piece hierarchical histogram such that one of the leaf rectangles is $R_{r'}$.
        return the composition of $U''$ and the sub-histogram defined by $V'$
    end if
end function
```

```
function FindSubtree($(T, w, s)$)
    Let $W = (v_1, \ldots, v_{2|V|-1})$ be a tour through the nodes of $T$.        ▷ $T = (V, E)$
    Let $w'(j) = \begin{cases} w(v_j) & \text{if position } j \text{ is the first appearance of } v_j \text{ in } W \\ 0 & \text{otherwise} \end{cases}$
    Let $S = (v_i, \ldots, v_{i+2s-1})$ be a contiguous subsequence of $W$ with $\sum_{j=i}^{i+2s-1} w'(j) \geq \frac{s}{|V|} \sum_{j=1}^{2|V|-1} w(j)$
    return the set of nodes in $S$.
end function
```

only once, and hence we get

$$\sum_{j \in V'} w(j) \;\geq\; \sum_{j=i}^{i+2s-1} w'(j) \;\geq\; \frac{s}{|V|} \sum_{j \in V} w(j)$$

as desired. Finally, since S is contiguous in the tour W, the nodes V' form a subtree in T of size at most $2s$. □

Utilizing Lemma 8.8, we now show how to extract a "good" sub-histogram from a given hierarchical histogram. More precisely, our goal is to find a sub-histogram U' with a bounded number of histogram pieces that still achieves a comparable "density" $\frac{\|P_{U'} b\|^2}{\gamma(U')} \approx \frac{\|P_U b\|^2}{\gamma(U)}$. In order to precisely state our algorithm and proof, we now formalize the connection between hierarchical histograms and tree graphs.

For a given histogram subspace U, let $T_U = (V_U, E_U)$ be the tree defined as folllows: First, every split in the hierarchical histogram corresponds to a node in V_U. For each split, we then add an edge from the split to the split directly above it in the histogram hierarchy. For a histogram subspace with $\gamma(U)$ pieces, this leads to a tree with $\gamma(U) - 1$ nodes. We also associate each node v in the tree with three rectangles. Specifically, let $R(v)$ be the rectangle split at v, and let $R_l(v)$ and $R_r(v)$ be the left and right child rectangles resulting from the split, respectively.

Next, we define the node weight function $w : V_U \to \mathbb{R}$. The idea is that the weight of a node corresponds to the "projection refinement", i.e., the gain in preserved energy when projected onto the finer histogram. More formally, for a rectangle R, let $P_R b$ the projection of b onto the rectangle R, i.e.,

$$(P_R b)_{i,j} \;=\; \begin{cases} 0 & \text{if } (i,j) \notin R \\ \frac{1}{|R|} \sum_{(u,v)\in\mathbb{R}} b_{u,v} & \text{otherwise} \end{cases} .$$

Then we define the weight of a node v as

$$w(v) \;=\; \|P_{R_l(v)} b\|^2 + \|P_{R_r(v)} b\|^2 - \|P_{R(v)}\|^2 .$$

Let $R_1, \ldots, R_{\gamma(U)}$ be the rectangles in the hierarchical histogram U, and let R_0 be the $\sqrt{d} \times \sqrt{d}$ "root" rectangle. Since the rectangles are non-overlapping, we have

$$\sum_{i=1}^{\gamma(U)} P_{R_i} b \;=\; P_U b .$$

Note that the rectangles $R_1, \ldots, R_{\gamma(U)}$ are exactly the child rectangles of the leaves in the tree T_U. Moreover, by the construction of the weight function w, we have

$$\|P_{R_0} b\|^2 + \sum_{v \in V_U} w(v) \;=\; \|P_U b\|^2$$

because the contributions from intermediate nodes in the tree T_U cancel out.

Lemma 8.9. *Let $b \in \mathbb{R}^{\sqrt{d}\times\sqrt{d}}$, let U be a hierarchical histogram subspace, and let $s \leq \gamma(U)$ be the target number of histogram pieces. Then* FindSubhistogram(b, U, s) *returns a hierarchical histogram subspace U' such that $\gamma(U') \leq 2s + 4$ and $\|P_{U'} b\|^2 \geq \frac{s}{\gamma(U)} \|P_U b\|^2$. Moreover, the algorithm runs in time $O(d)$.*

Proof. Note that by construction, the tree T_U^* defined in FindSubhistogram has k nodes and the node weights w^* satisfy

$$\sum_{v \in V_{T_U^*}} w(v) \;=\; \|P_U b\|^2 .$$

Lemma 8.8 then shows that the subtree defined by the set of nodes V' satisfies $|V'| \leq 2s$ and

$$\sum_{v \in V'} w(v) \;\geq\; \frac{s}{\gamma(U)} \sum_{v \in V_{T_U^*}} w(v) \;\geq\; \frac{s}{\gamma(U)} \|P_U b\|^2 .$$

Let $R'_1, \ldots, R'_{|V'|}$ be the leaf rectangles of the subtree V'. The above lower bound on the sum of the node weights implies that

$$\sum_{i=1}^{|V'|} \|P_{R'_i} b\|^2 \;\geq\; \frac{s}{\gamma(U)} \|P_U b\|^2 .$$

because the rectangles R'_i are non-overlapping and the weights of the inner tree nodes in V' cancel as before. Hence any hierarchical histogram containing the rectangles $R'_1, \ldots, R'_{|V'|}$ satisfies the desired head projection bound. It remains to show that we can convert the subtree defined by V' into a hierarchical histogram.

If the set V' contains the root node of T^*_U, the subtree V' directly gives a valid sub-histogram of U. On the other hand, if the root node of T^*_U is not in V', we can construct a simple 4-piece hierarchical histogram U'' that contains the root rectangle $R_{r'}$ of V' as one of its leaf nodes. The histogram subspace U'' is given by four splits corresponding to the boundaries of the root rectangle $R_{r'}$. We can then combine the hierarchical histogram U'' with the subtree V' by adding the splits in V' to the hierarchical histogram in U'' (by construction, all these splits are valid). The resulting hierarchical histogram then has at most $4 + |V'| \leq 4 + 2s$ pieces.

The running time bound is straightforward: all pre-processing can be accomplished in linear time by computing partial sums for the vector b (projections onto a rectangle can then be computed in constant time). The subroutine FindSubtree also runs in linear time because it requires only a single pass over the tree of size $O(\gamma(U))$. □

We can now state our approximate head projection algorithm.

Theorem 8.10. *Let $b \in \mathbb{R}^d$, $k \in \mathbb{N}$, and $0 < \tau < 1$. Then* HistogramHead(b, k, τ) *returns a histogram subspace U such that $\gamma_U \leq \frac{4\alpha}{\tau} k + 4$ and*

$$\|P_U b\|^2 \;\geq\; (1-\tau) \max_{U' \in \mathscr{H}_{\mathrm{h},k}} \|P_{U'} b\|^2 \; .$$

Moreover, the algorithm runs in time $O(d^{1+\zeta} \log \frac{\alpha d}{\tau})$.

Proof. First, we introduce a few shortands to simplify notation. Let the histogram subspace U_l be the solution returned by HistogramOracle(b, λ_l). We then write $h_l = \|P_{U_l} b\|^2$ for the head approximation of U_l and $\gamma_l = \gamma(U_l)$ for the number of histogram pieces in the histogram subspace U_l. We adopt a similar convention for h_r and γ_r (corresponding to the solution for parameter λ_r). Finally, let h^* be the optimal head approximation achievable with a k-histogram, i.e., $h^* = \max_{U' \in \mathscr{H}_{\mathrm{h},k}} \|P_{U'} b\|^2$.

Rearranging Equation (8.1), using the new notation, and substituting the optimal k-histogram solution for the max-quantifier gives

$$h_l \;\geq\; h^* - \lambda_l \left(k - \frac{\gamma_l}{\alpha} \right) \; . \tag{8.5}$$

We now consider the case that the algorithm returns in Line 6. We clearly have $\gamma(U_l) \leq \frac{4\alpha}{\tau} k + 4$ when reaching Line 6. Moreover, substituting for λ_l in Equation 8.5 gives

$$\begin{aligned} h_l \;&\geq\; h^* - \frac{b_{\max}\tau}{k} \left(k - \frac{\gamma_l}{\alpha} \right) \\ &\geq\; (1-\tau) h^* \end{aligned}$$

where the second line follows from $h^* \geq b_{\max}$. This inequality holds because any histogram with at least 4 pieces can always create a rectangle that isolates the largest element in b

(for simplicity, we assume that $k \geq 4$ and $b \neq 0$). Hence U_l satisfies the conditions of the theorem.

Next, we consider the case that the algorithm reaches the binary search. Note that the binary search is initialized and performed such that we have $\lambda_l \leq \lambda_r \leq \lambda_l + \varepsilon$ when it terminates. Moreover, we have $\gamma_r \leq \frac{2\alpha}{\tau}k$ and $\gamma_l > \frac{2\alpha}{\tau}k$. We now distinguish two sub-cases based on the "density" $\frac{h_l}{\gamma_l}$ of the solution U_l corresponding to λ_l. Let $\phi = \frac{\tau(1-\tau/2)}{2\alpha}$ be the density threshold compared to the optimal solution density $\frac{h^*}{k}$.

Sub-case 1: $\frac{h_l}{\gamma_l} \leq \phi\frac{h^*}{k}$. This inequality allows us to establish an upper bound on λ_l. Rearranging Equation (8.5) gives (note that $k - \frac{\lambda_l}{\alpha}$ is negative):

$$\begin{aligned} \lambda_l &\leq \frac{h_l - h^*}{\gamma_l/\alpha - k} \\ &\leq \frac{\alpha h_l}{\gamma_l - \alpha k} \,. \end{aligned}$$

We now use $\gamma_l \geq \frac{2\alpha}{\tau}k$:

$$\begin{aligned} \lambda_l &\leq \frac{\alpha h_l}{\gamma_l - \tau\gamma_l/2} \\ &\leq \frac{h_l}{\gamma_l} \cdot \frac{\alpha}{1-\tau/2} \\ &\leq \phi\frac{h^*}{k}\frac{\alpha}{1-\tau/2} \\ &\leq \frac{\tau}{2} \cdot \frac{h^*}{k} \,. \end{aligned}$$

where we used the density upper bound for U_l valid in this subcase and the definition of ϕ. Next, we derive a lower bound on h_r. Instantiating Equation (8.5) with U_r instead of U_l gives

$$\begin{aligned} h_r &\geq h^* - \lambda_r\left(k - \frac{\gamma_r}{\alpha}\right) \\ &\geq h^* - \lambda_r k \\ &\geq h^* - (\lambda_l + \varepsilon)k \\ &= h^* - \lambda_l k - \varepsilon k \\ &\geq h^* - \frac{\tau}{2}h^* - \frac{\tau}{2}b_{\max} \\ &\geq (1-\tau)h^* \end{aligned}$$

where we again used $b_{\max} \leq h^*$. So in this sub-case, U_r satisfies the conditions of the theorem.

Sub-case 2: $\frac{h_l}{\gamma_l} \geq \phi\frac{h^*}{k}$. In this subcase, the solution U_l has a good density, so FindSubhistogram can extract a good solution with a bounded number of histogram pieces. More formally, since $\gamma_l \geq \frac{2\alpha}{\tau}k$, we can invoke Lemma 8.9 and get

$$\begin{aligned}\|P_{U_l'}b\|^2 &\geq \frac{\frac{2\alpha}{\tau}k}{\gamma_l}h_l \\ &\geq \frac{2\alpha k}{\tau}\phi\frac{h^*}{k} \\ &\geq \left(1-\frac{\tau}{2}\right)h^* \, .\end{aligned}$$

Moreover, the output of FindSubhistogram satisfies $\gamma(U_l') \leq \frac{4\alpha}{\tau}k + 4$, and hence U_l' satisfies the conditions of the theorem.

We can now conclude the proof of the theorem: always, one of sub-case 1 and sub-case 2 holds. Since HistogramHead always returns the best of the two choices U_r and U_l', the overall result has the desired head approximation guarantee.

The overall running time is dominated by the invocations of HistogramOracle in the binary search. Each invocation takes $O(n^{1+\zeta})$ time and the number of invocations is the number of iterations of the binary search, i.e., bounded by

$$\lceil \log \frac{\lambda_r^{(0)} - \lambda_l^{(0)}}{\varepsilon} \rceil \;\leq\; \lceil \frac{\lambda_r^{(0)}}{\varepsilon} \rceil \;\leq\; \lceil \log \frac{4\alpha k \|b\|^2}{b_{\max}\tau} \rceil \, .$$

Since $k \leq d$ and $\frac{\|b\|^2}{b_{\max}} \leq d$, the running time bound in the theorem follows. □

As before, Theorem 8.2 follows as a direct consequence of Theorem 8.10. For completeness, we repeat the statement of Theorem 8.2:

Theorem 8.2. *Let $\zeta > 0$ and $\varepsilon > 0$ be arbitrary. Then there is an $(1-\varepsilon, \mathbb{U}_k, \mathbb{U}_{c\cdot k})$-approximate head projection for 2D histograms where $c = O(1/\zeta^2\varepsilon)$. Moreover, the algorithm runs in time $\widetilde{O}(d^{1+\zeta})$.*

Setting $\tau = O(\varepsilon)$ gives the $1-\varepsilon$ guarantee in Theorem 8.2. Moreover, we use the $\alpha = O(1/\zeta^2)$ dependence from Theorem 3 of [167].

8.3 Recovery of histograms

Unlike previous chapters, we cannot instantiate our head and tail approximations for 2D-histograms to get an "end-to-end" recovery guarantee. The reasons are somewhat technical and will be discussed in the next subsection. Instead, we show how to construct a fast measurement matrix for 2D-histograms (Subsection 8.3.2). Moreover, our empirical results in Section 8.4 demonstrate that our algorithms do succeed in numerical experiments and improve over the sample complexity of a wavelet approach.

8.3.1 A conjecture

While we have approximate projections for 2D histograms, they do not suffice to state an overall recovery guarantee in the current form. The issue is that our recovery framework from Chapter 2 requires an approximate head projection that is competitive with respect to the sum of subspaces $\mathbb{U} + \mathbb{U}_{\mathcal{T}}$. While this is easy to satisfy for low-rank matrices (the sum of two rank-r subspace models is contained in the rank-$2r$ subspace model), adding histogram subspace models is more subtle. For instance, consider two k-histogram subspaces corresponding to k rows and columns of a $k \times k$ matrix, respectively. The sum of the two subspaces then contains k^2 individual rectangles (a chessboard pattern). While these k^2 rectangles are not independent (the dimension of the space is only $2k$), the chessboard pattern is no directly contained in the set of $2k$-histogram subspaces. As a result, a head approximation that is competitive with respect to $2k$-histograms is not immediately competitive with respect to the sum of two k-histograms.

While head boosting (c.f. Section 2.6.5) is not directly helpful to overcome this issue, we believe that 2D histograms are "well-behaved" in the sense that boosting is still helpful. In particular, we believe that the sum of two k-histograms still allows a constant-factor head approximation with a single $O(k)$-histogram subspace. More formally, we state the following conjecture.

Conjecture 8.1. *Let $c > 0$ be fixed. Then there are universal constants $c_1 > 0$ and $c_2 > 0$ depending on c such that the following holds. For any $b \in \mathbb{R}^d$, there is a $c_1 k$-histogram subspace U such that we have*

$$\|P_U b\| \;\geq\; c_2 \|P_{c \times \mathbb{U}_k} b\| \; .$$

If the above conjecture is true, Theorem 8.10 yields an approximate head projection that is competitive to $c \times \mathbb{U}_k$. Combining this with our recovery framework from Chapter 2 then yields an overall recovery algorithm.

8.3.2 A fast measurement matrix

As in Section 7.3.2, we now establish a bound on the sample complexity of subspace recovery. This time, our focus is in 2D-histograms (instead of low-rank matrices). As before, our focus is on *fast* measurement matrices, i.e., matrices that support matrix-vector multiplications with a running time that is *nearly-linear* in the size of the vector. Our construction is again a concatenation of previous results.

We consider the case where the subspace model $\mathbb{U}$ corresponds to the set of hierarchical or tiling histograms. Since either type of histogram can be modeled as superpositions of sub-rectangles of the domain $\sqrt{d} \times \sqrt{d}$, we can simply model the histogram subspace model $\mathbb{U}$ as a subset of *dictionary-sparse* vectors

$$\{x | x = D\alpha, \; \|\alpha\|_0 \leq k\} \; .$$

Here, D is a dictionary of size $d \times \binom{d^2}{2}$ where each column of D corresponds to a single tile (normalized to unit ℓ_2-norm).

Therefore, any matrix that satisfies the RIP with respect to the dictionary D (abbreviated sometimes as the D-RIP) also suffices for reliable histogram subspace recovery. The following result can be derived similar to Theorem 2.29 in Section 2.6.6.

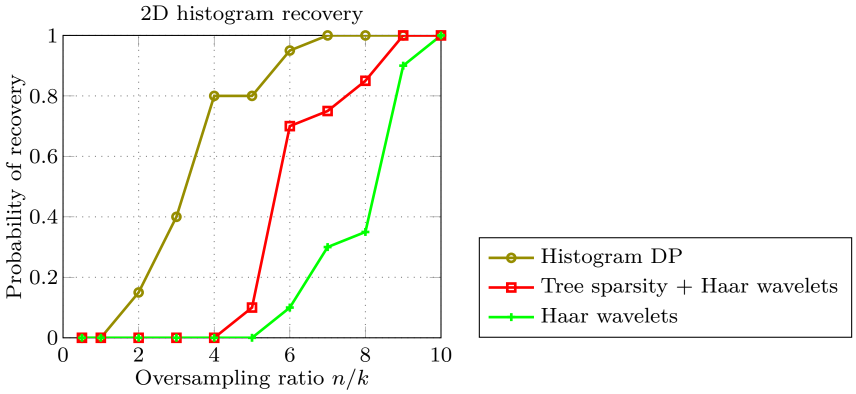


Figure (8-1): Results for recovering a hierarchical histogram from subsampled Fourier measurements. As predicted by our theoretical argument, the 2D histogram DP has the best sample complexity.

Theorem 8.11. *There exists a randomized construction of a matrix $X \in \mathbb{R}^{n \times d}$, with parameters $n = O(k \log d/k)$, such that with high probability, X satisfies the subspace RIP for the histogram subspace model. Moreover, X supports matrix-vector multiplications with complexity $O(d \log d + k^2 \text{polylog} d)$.*

8.4 Experiments

We begin with a description of the experimental setup. All experiments were conducted on an iMac desktop computer with an Intel Core i5 CPU (3.2 GHz) and 16 GB RAM. With the exception of the dynamic program (DP) for 2D histograms, all code was written in Matlab. We chose C++ for the 2D histogram DP because it heavily relies on for-loops, which tend to be slow in Matlab. Unless reported otherwise, all reported data points were averaged over at least 10 trials.

Figure 8-1 shows our results for recovering a 2D histogram from linear observations. As before, we use subsampled Fourier measurements. Our test vector is a 32×32 hierarchical histogram consisting of 4 rectangles. Hierarchical histograms are essentially 2D piecewise constant functions over a 2D domain where the constant pieces (or tiles) are generated by starting with the entire domain as a single tile and recursively partitioning tiles by making horizontal or vertical splits. We compare three approaches:

- “Standard” sparsity in the Haar wavelet domain.
- Tree sparsity in the Haar wavelet domain [36, 105].
- Our approximate projection algorithm.

The focus in our experiments is on sample complexity, so we have implemented only one “level” of the DP in [167]. As the phase transitions in Figure 8-1 show, our 2D histogram DP does indeed offer the best empirical sample complexity.

Part II

Lower Bounds for Unconstrained Optimization

Chapter 9

The Fine-Grained Complexity Of Empirical Risk Minimization

9.1 Introduction

Empirical risk minimization (ERM) has been highly influential in modern machine learning [225]. ERM underpins many core results in statistical learning theory and is one of the main computational problems in the field. Several important methods such as support vector machines (SVM), boosting, and neural networks follow the ERM paradigm [207]. As a consequence, the algorithmic aspects of ERM have received a vast amount of attention over the past decades. This naturally motivates the following basic question:

What are the computational limits for ERM algorithms?

In this thesis, we address this question both in convex and non-convex settings. Convex ERM problems have been highly successful in a wide range of applications, giving rise to popular methods such as SVMs and logistic regression. Using tools from convex optimization, the resulting problems can be solved in polynomial time. However, the exact time complexity of many important ERM problems such as kernel SVMs is not yet well understood. As the size of data sets in machine learning continues to grow, this question is becoming increasingly important. For ERM problems with millions of high-dimensional examples, even quadratic time algorithms can become painfully slow (or expensive) to run.

Non-convex ERM problems have also attracted extensive research interest, e.g., in the context of deep neural networks. First order methods that follow the gradient of the empirical loss are not guaranteed to find the global minimizer in this setting. Nevertheless, variants of gradient descent are by far the most common method for training large neural networks. Here, the computational bottleneck is to compute a number of gradients, not necessarily to minimize the empirical loss globally. Although we can compute gradients in polynomial time, the large number of parameters and examples in modern deep learning still makes this a considerable computational challenge.

Unfortunately, there are only few existing results concerning the exact time complexity of ERM or gradient computations. Since the problems have polynomial time algorithms, the classical machinery from complexity theory (such as NP hardness) is too coarse to apply.

Oracle lower bounds from optimization offer useful guidance for convex ERM problems, but the results only hold for limited classes of algorithms. Moreover, they do not account for the *cost* of executing the oracle calls, as they simply lower bound their number. Overall, we do not know if common ERM problems allow for algorithms that compute a high-accuracy solution in sub-quadratic or even nearly-linear time for all instances.[1] Furthermore, we do not know if there are more efficient techniques for computing (mini-)batch gradients than simply treating each example in the batch independently.[2]

We address both questions for multiple well-studied ERM problems.

Hardness of ERM. First, we give conditional hardness results for minimizing the empirical risk in several settings, including kernel SVMs, kernel ridge regression (KRR), and training the top layer of a neural network. Our results give evidence that no algorithms can solve these problems to high accuracy in strongly sub-quadratic time. Moreover, we provide similar conditional hardness results for kernel PCA. All of these methods are popular learning algorithms due to the expressiveness of the kernel or network embedding. Our results show that this expressiveness also leads to an expensive computational problem.

Hardness of gradient computation in neural networks. Second, we address the complexity of computing a gradient for the empirical risk of a neural network. In particular, we give evidence that computing (or even approximating, up to polynomially large factors) the norm of the gradient of the top layer in a neural network takes time that is "rectangular". The time complexity cannot be significantly better than $O(n \cdot m)$, where m is the number of examples and n is the number of units in the network. Hence, there are no algorithms that compute batch gradients faster than handling each example individually, unless common complexity-theoretic assumptions fail.

Our hardness results for gradient computation apply to common activation functions such as ReLU or sigmoid units. We remark that for *polynomial* activation functions (for instance, studied in [155]), significantly faster algorithms *do exist*. Thus, our results can be seen as mapping the "efficiency landscape" of basic machine learning sub-routines. They distinguish between what is possible and (likely) impossible, suggesting further opportunities for improvement.

Our hardness results are based on recent advances in fine-grained complexity and build on conjectures such as the Strong Exponential Time Hypothesis (SETH) [123, 124, 226]. SETH concerns the classic satisfiability problem for formulas in Conjunctive Normal Form (CNF). Informally, the conjecture states that there is no algorithm for checking satisfiability of a formula with n variables and m clauses in time less than $O(c^n \cdot \text{poly}(m))$ for some $c < 2$.[3] While our results are conditional, SETH has been employed in many recent

[1] More efficient algorithms exist if the running time is allowed to be polynomial in the accuracy parameter, e.g., [208] give such an algorithm for the kernel SVM problem that we consider as well. See also the discussion at the end of this section.

[2] Consider a network with one hidden layer containing n units and a training set with m examples, for simplicity in small dimension $d = O(\log n)$. No known results preclude an algorithm that computes a full gradient in time $O((n + m) \log n)$. This would be significantly faster than the standard $O(n \cdot m \cdot \log n)$ approach of computing the full gradient example by example.

[3] Note that SETH can be viewed as a significant strengthening of the P $\neq$ NP conjecture, which only postulates that there is no *polynomial time* algorithm for CNF satisfiability. The best known algorithms for

hardness results. Its plausibility stems from the fact that, despite 60 years of research on satisfiability algorithms, no such improvement has been discovered.

Our results hold for a significant range of the accuracy parameter. For kernel methods, our bounds hold for algorithms approximating the empirical risk up to a factor of $1 + \varepsilon$, for $\log(1/\varepsilon) = \omega(\log^2 n)$). Thus, they provide conditional quadratic lower bounds for algorithms with, say, a $\log 1/\varepsilon$ runtime dependence on the approximation error ε. A (doubly) logarithmic dependence on $1/\varepsilon$ is generally seen as the ideal rate of convergence in optimization, and algorithms with this property have been studied extensively in the machine learning community (cf. [28].). At the same time, approximate solutions to ERM problems can be sufficient for good generalization in learning tasks. Indeed, stochastic gradient descent (SGD) is often advocated as an efficient learning algorithm despite its polynomial dependence on $1/\varepsilon$ in the optimization error [52, 208]. Our results support this viewpoint since SGD sidesteps the quadratic time complexity of our lower bounds.

For other problems, our assumptions about the accuracy parameter are less stringent. In particular, for training the top layer of the neural network, we only need to assume that $\varepsilon \approx 1/n$. Finally, our lower bounds for approximating the norm of the gradient in neural networks hold even if $\varepsilon = n^{O(1)}$, i.e., for *polynomial* approximation factors (or alternatively, a constant additive factor for ReLU and sigmoid activation functions).

Finally, we note that our results do not rule out algorithms that achieve a sub-quadratic running time for well-behaved instances, e.g., instances with low-dimensional structure. Indeed, many such approaches have been investigated in the literature, for instance the Nyström method or random features for kernel problems [179, 234]. Our results offer an explanation for the wide variety of techniques. The lower bounds are evidence that there is no “silver bullet” algorithm for solving the aforementioned ERM problems in sub-quadratic time, to high accuracy, and for all instances.

9.1.1 Background

We briefly review the relevant background from fine-grained complexity. The ERM problems we consider will be formally introduced together with our results in the next section.

We obtain our conditional hardness results via reductions from two well-studied problems: *Orthogonal Vectors* and *Bichromatic Hamming Close Pair*.

Definition 9.1 (Orthogonal Vectors problem (OVP))**.** *Given two sets $A = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^d$ and $B = \{b_1, \ldots, b_n\} \subseteq \{0,1\}^d$ of n binary vectors, decide if there exists a pair $a \in A$ and $b \in B$ such that $a^\mathsf{T} b = 0$.*

For OVP, we can assume without loss of generality that all vectors in B have the same number of 1s. This can be achieved by appending d entries to every b_i and setting the necessary number of them to 1 and the rest to 0. We then append d entries to every a_i and set all of them to 0.

Definition 9.2 (Bichromatic Hamming Close Pair (BHCP) problem)**.** *Given two sets $A = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^d$ and $B = \{b_1, \ldots, b_n\} \subseteq \{0,1\}^d$ of n binary vectors and an integer $t \in \{2, \ldots, d\}$, decide if there exists a pair $a \in A$ and $b \in B$ such that the number of*

CNF satisfiability have running times of the form $O(2^{(1-o(1))n} \cdot \mathrm{poly}(m))$.

coordinates in which they differ is less than t (formally, Hamming$(a,b) := ||a-b||_1 < t$). If there is such a pair (a,b), we call it a close pair.

It is known that both OVP and BHCP require almost quadratic time (i.e., $n^{2-o(1)}$) for any $d = \omega(\log n)$ assuming SETH [15].[1] Furthermore, if we allow the sizes $|A| = n$ and $|B| = m$ to be different, both problems require $(nm)^{1-o(1)}$ time assuming SETH, as long as $m = n^{\alpha}$ for some constant $\alpha \in (0,1)$ [57]. Our proofs will proceed by embedding OVP and BHCP instances into ERM problems. Such a reduction then implies that the ERM problem requires almost quadratic time if the SETH is true. If we could solve the ERM problem faster, we would also obtain a faster algorithm for the satisfiability problem.

9.1.2 Our contributions

We now formally describe our main contributions, starting with conditional hardness results for kernel problems. We then give our results for neural network ERM problems and for computing gradients.

9.1.2.1 Kernel problems

We provide hardness results for two kernel ERM problems. In the following, let $x_1, \ldots, x_n \in \mathbb{R}^d$ be the n input vectors, where $d = \omega(\log n)$. We use $y_1, \ldots, y_n \in \mathbb{R}$ as n labels or target values. Finally, let $k(x,x')$ denote a kernel function and let $K \in \mathbb{R}^{n \times n}$ be the corresponding kernel matrix, defined as $K_{i,j} := k(x_i, x_j)$ [203]. Concretely, we focus on the Gaussian kernel $k(x,x') := \exp\left(-C\|x-x'\|_2^2\right)$ for some $C > 0$. We note that our results can be generalized to any kernel with exponential tail.

Kernel SVM. For simplicity, we present our result for hard-margin SVMs without bias terms. This gives the following optimization problem.

Definition 9.3 (Hard-margin SVM)**.** *A (primal) hard-margin SVM is an optimization problem of the following form:*

$$\begin{aligned} &\underset{\alpha_1,\ldots,\alpha_n \geq 0}{\textit{minimize}} \quad \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\, \alpha_j\, y_i\, y_j\, k(x_i, x_j) \\ &\textit{subject to} \quad y_i f(x_i) \geq 1, \quad i = 1, \ldots, n, \end{aligned} \tag{9.1}$$

where $f(x) := \sum_{i=1}^{n} \alpha_i y_i k(x_i, x)$.

The following theorem is our main result for SVMs, described in more detail in Section 9.2.1. In Sections 9.4.2.1, 9.4.2.2, and 9.4.2.3 we provide similar hardness results for other common SVM variants, including the soft-margin version.

Theorem 9.4. *Let $k(a,a')$ be the Gaussian kernel with $C = 100 \log n$ and let*

$$\varepsilon = \exp(-\omega(\log^2 n)) .$$

[1] We use $\omega(g(n))$ to denote any function f such that $\lim_{n\to\infty} f(n)/g(n) = \infty$. Similarly, we use $o(g(n))$ to denote any function f such that $\lim_{n\to\infty} f(n)/g(n) = 0$. Consequently, we will refer to functions of the form $\omega(1)$ as *super-constant* and to $n^{\omega(1)}$ as *super-polynomial*.

Then approximating the optimal value of Equation (9.1) *within a multiplicative factor* $1+\varepsilon$ *requires almost quadratic time assuming SETH.*

Kernel ridge regression. Next we consider Kernel ridge regression, which is formally defined as follows.

Definition 9.5 (Kernel ridge regression)**.** *Given a real value* $\lambda \geq 0$*, the goal of kernel ridge regression is to output*

$$\arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{2}||y - K\alpha||_2^2 + \frac{\lambda}{2}\alpha^\intercal K\alpha.$$

This problem is equivalent to computing the vector $(K+\lambda I)^{-1}y$. We focus on the special case where $\lambda = 0$ and the vector y has all equal entries $y_1 = \ldots = y_n = 1$. In this case, the entrywise sum of $K^{-1}y$ is equal to the sum of the entries in K^{-1}. Thus, we show hardness for computing the latter quantity (see Section 9.4.4for the proof).

Theorem 9.6. *Let* $k(a,a')$ *be the Gaussian kernel for any parameter* $C = \omega(\log n)$ *and let* $\varepsilon = \exp(-\omega(\log^2 n))$*. Then computing the sum of the entries in* K^{-1} *up to a multiplicative factor of* $1+\varepsilon$ *requires almost quadratic time assuming SETH.*

Kernel PCA. Finally, we turn to the Kernel PCA problem, which we define as follows [164].

Definition 9.7 (Kernel Principal Component Analysis (PCA))**.** *Let* 1_n *be an* $n \times n$ *matrix where each entry takes value* $1/n$*, and define* $K' := (I - 1_n)K(I - 1_n)$*. The goal of the kernel PCA problem is to output the* n *eigenvalues of the matrix* K'*.*

In the above definition, the output only consists of the eigenvalues, not the eigenvectors. This is because computing all n eigenvectors trivially takes at least quadratic time since the output itself has quadratic size. Our hardness proof applies to the potentially simpler problem where only the eigenvalues are desired. Specifically, we show that computing *the sum* of the eigenvalues (i.e., the *trace* of the matrix) is hard. See Section 9.4.3 for the proof.

Theorem 9.8. *Let* $k(a,a')$ *be the Gaussian kernel with* $C = 100\log n$ *and let*

$$\varepsilon = \exp(-\omega(\log^2 n)) \ .$$

Then approximating the sum of the eigenvalues of $K' = (I - 1_n)K(I - 1_n)$ *within a multiplicative factor of* $1+\varepsilon$ *requires almost quadratic time assuming SETH.*

We note that the argument in the proof shows that even approximating the sum of the entries of K is hard. This provides an evidence of hardness of the *kernel density estimation* problem for Gaussian kernels, complementing recent upper bounds of [71].

9.1.2.2 Neural network ERM problems

We now consider neural networks. We focus on the problem of optimizing the top layer while keeping lower layers unchanged. An instance of this problem is transfer learning with large networks that would take a long time and many examples to train from scratch [184]. We consider neural networks of depth 2, with the sigmoid or ReLU activation function. Our hardness result holds for a more general class of "nice" activation functions S as described later (see Definition 9.12).

Given n weight vectors $w_1, \ldots, w_n \in \mathbb{R}^d$ and n weights $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, consider the function $f : \mathbb{R}^d \to \mathbb{R}$ using a non-linearity $S : \mathbb{R} \to \mathbb{R}$:

$$f(u) \ := \ \sum_{j=1}^{n} \alpha_j \cdot S(u^{\mathsf{T}} w_j) \ .$$

This function can be implemented as a neural net that has d inputs, n nonlinear activations (units), and one linear output.

To complete the ERM problem, we also require a loss function. Our hardness results hold for a large class of "nice" loss functions, which includes the hinge loss and the logistic loss.[1] Given a nice loss function and m input vectors $a_1, \ldots, a_m \in \mathbb{R}^d$ with corresponding labels y_i, we consider the following problem:

$$\underset{\alpha_1,\ldots,\alpha_n \in \mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{m} \text{loss}(y_i, f(u_i)). \tag{9.2}$$

Our main result is captured by the following theorem (see Section 9.2.3 for the proof). For simplicity, we set $m = n$.

Theorem 9.9. *For any $d = \omega(\log n)$, approximating the optimal value in Equation* (9.2) *up to a multiplicative factor of $1 + \frac{1}{4n}$ requires almost quadratic time assuming SETH.*

9.1.2.3 Hardness of gradient computation

Finally, we consider the problem of computing the gradient of the loss function for a given set of examples. We focus on the network architecture from the previous section. Formally, we obtain the following result:

Theorem 9.10. *Consider the empirical risk in Equation* (9.2) *under the following assumptions: (i) The function f is represented by a neural network with n units, $n \cdot d$ parameters, and the ReLU activation function. (ii) We have $d = \omega(\log n)$. (iii) The loss function is the logistic loss or hinge loss. Then approximating the ℓ_p-norm (for any $p \geq 1$) of the gradient of the empirical risk for m examples within a multiplicative factor of n^C for any constant $C > 0$ takes at least $O\left((nm)^{1-o(1)}\right)$ time assuming SETH.*

See Section 9.2.4 for the proof. We also prove a similar statement for the sigmoid activation function. At the same time, we remark that for *polynomial* activation functions, significantly faster algorithms do exist, using the polynomial lifting argument. Specifically, for the polynomial activation function of the form x^r for some integer $r \geq 2$, all gradients can be computed in $O((n{+}m)d^r)$ time. Note that the running time of the standard backpropagation algorithm is $O(dnm)$ for networks with this architecture. Thus one can improve over

[1] In the binary setting we consider, the logistic loss is equivalent to the softmax loss commonly employed in deep learning.

backpropagation for a non-trivial range of parameters, especially for quadratic activation function when $r = 2$. See Section 9.4.6 for more details.

9.1.3 Related work

Recent work has demonstrated conditional quadratic hardness results for many combinatorial optimization problems over graphs and sequences. These results include computing diameter in sparse graphs [72, 189], Local Alignment [7], Fréchet distance [56], Edit Distance [30], Longest Common Subsequence, and Dynamic Time Warping [3, 57]. In the machine learning literature, [31] recently showed a tight lower bound for the problem of inferring the most likely path in a Hidden Markov Model, matching the upper bound achieved by the Viterbi algorithm [230]. As in our work, the SETH and related assumptions underlie these lower bounds. To the best of our knowledge, we give the first application of this methodology to *continuous* (as opposed to combinatorial) optimization problems.

There is a long line of work on the oracle complexity of optimization problems, going back to [173]. We refer the reader to [174] for these classical results. The oracle complexity of ERM problems is still subject of active research, e.g., see [8, 21, 22, 65, 240]. The work closest to ours is [65], which gives quadratic time lower bounds for ERM algorithms that access the kernel matrix through an evaluation oracle or a low-rank approximation.

The oracle results are fundamentally different from the lower bounds presented here. Oracle lower bounds are typically unconditional, but inherently apply only to a limited class of algorithms due to their information-theoretic nature. Moreover, they do not account for the *cost* of executing the oracle calls, as they merely lower bound their number. In contrast, our results are conditional (based on the SETH and related assumptions), but apply to *any* algorithm and account for the *total* computational cost. This significantly broadens the reach of our results. We show that the hardness is not due to the oracle abstraction but instead inherent in the computational problem.

9.2 Overview of the hardness proofs

We now give high-level overviews of our hardness proofs.

9.2.1 Kernel SVMs

Let $A = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^d$ and $B = \{b_1, \ldots, b_n\} \subseteq \{0,1\}^d$ be the two sets of binary vectors from a BHCP instance with $d = \omega(\log n)$. Our goal is to determine whether there is a close pair of vectors. We show how to solve this BHCP instance by reducing it to *three* computations of SVM, defined as follows:

1. We take the first set A of binary vectors, assign label 1 to all vectors, and solve the

corresponding SVM on the n vectors:

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0}{\text{minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j k(a_i,a_j) \\ \text{subject to} \quad & \sum_{j=1}^{n} \alpha_j k(a_i,a_j) \geq 1, \quad i = 1,\ldots,n. \end{aligned} \tag{9.3}$$

Note that we do not have y_i in the expressions because all labels are 1.

2. We take the second set B of binary vectors, assign label -1 to all vectors, and solve the corresponding SVM on the n vectors:

$$\begin{aligned} \underset{\beta_1,\ldots,\beta_n \geq 0}{\text{minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{n} \beta_i\beta_j k(b_i,b_j) \\ \text{subject to} \quad & -\sum_{j=1}^{n} \beta_j k(b_i,b_j) \leq -1, \quad i = 1,\ldots,n. \end{aligned} \tag{9.4}$$

3. We take both sets A and B of binary vectors, assign label 1 to all vectors from the first set A and label -1 to all vectors from the second set B. We then solve the corresponding SVM on the $2n$ vectors:

$$\begin{aligned} \underset{\substack{\alpha_1,\ldots,\alpha_n \geq 0\\ \beta_1,\ldots,\beta_n \geq 0}}{\text{minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j k(a_i,a_j) + \frac{1}{2}\sum_{i,j=1}^{n} \beta_i\beta_j k(b_i,b_j) - \sum_{i,j=1}^{n} \alpha_i\beta_j k(a_i,b_j) \\ \text{subject to} \quad & \sum_{j=1}^{n} \alpha_j k(a_i,a_j) - \sum_{j=1}^{n} \beta_j k(a_i,b_j) \geq 1, \quad i = 1,\ldots,n\ , \\ & -\sum_{j=1}^{n} \beta_j k(b_i,b_j) + \sum_{j=1}^{n} \alpha_j k(b_i,a_j) \leq -1, \quad i = 1,\ldots,n\ . \end{aligned} \tag{9.5}$$

Reduction running time. The reductions to SVM proceed by creating appropriately labeled data sets (A, B or $A \cup B$) and invoking the SVM subroutine on each of them. Each of those data sets contain $O(n)$ points in $\mathbb{R}^d$, so the reductions run in $O(dn)$ time. Since we only need to assume that $d = \omega(\log n)$, it follows that a sub-quadratic algorithm for SVM implies a sub-quadratic algorithm for BHCP.

Intuition behind the construction. To show a reduction from the BHCP problem to SVM computation, we have to consider two cases:

- The YES case of the BHCP problem when there are two vectors that are close in Hamming distance. That is, there exist $a_i \in A$ and $b_j \in B$ such that $\text{Hamming}(a_i, b_j) < t$.
- The NO case of the BHCP problem when there is no close pair of vectors. That is, for all $a_i \in A$ and $b_j \in B$, we have $\text{Hamming}(a_i, b_j) \geq t$.

We show that we can distinguish between these two cases by comparing the objective value of the first two SVM instances above to the objective value of the third.

Intuition for the NO case. We have $\text{Hamming}(a_i, b_j) \geq t$ for all $a_i \in A$ and $b_j \in B$. The Gaussian kernel then gives the inequality

$$k(a_i, b_j) = \exp(-100 \log n \cdot \|a_i - b_j\|_2^2) \leq \exp(-100 \log n \cdot t)$$

for all $a_i \in A$ and $b_j \in B$. This means that the value $k(a_i, b_j)$ is very small. For simplicity, assume that it is equal to 0, i.e., $k(a_i, b_j) = 0$ for all $a_i \in A$ and $b_j \in B$.

Consider the third SVM (9.5). It contains three terms involving $k(a_i, b_j)$: the third term in the objective function, the second term in the inequalities of the first type, and the second term in the inequalities of the second type. We assumed that these terms are equal to 0 and we observe that the rest of the third SVM is equal to the sum of the first SVM (9.3) and the second SVM (9.4). Thus we expect that the optimal value of the third SVM is approximately equal to the sum of the optimal values of the first and the second SVMs. If we denote the optimal value of the first SVM (9.3) by $\text{value}(A)$, the optimal value of the second SVM (9.4) by $\text{value}(B)$, and the optimal value of the third SVM (9.5) by $\text{value}(A, B)$, then we can express our intuition in terms of the approximate equality

$$\text{value}(A, B) \approx \text{value}(A) + \text{value}(B) \; .$$

Intuition for the YES case. In this case, there is a close pair of vectors $a_i \in A$ and $b_j \in B$ such that $\text{Hamming}(a_i, b_j) \leq t - 1$. Since we are using the Gaussian kernel we have the following inequality for this pair of vectors:

$$k(a_i, b_j) = \exp(-100 \log n \cdot \|a_i - b_j\|_2^2) \geq \exp(-100 \log n \cdot (t-1)) \; .$$

We therefore have a large summand in each of the three terms from the above discussion. Thus the three terms do not (approximately) disappear and there is no reason for us to expect that the approximate equality holds. We can thus expect

$$\text{value}(A, B) \not\approx \text{value}(A) + \text{value}(B) \; .$$

Thus, by computing $\text{value}(A, B)$ and comparing it to $\text{value}(A) + \text{value}(B)$ we can distinguish between the YES and NO instances of BHCP. This completes the reduction. The full proofs are given in Section 9.4.2.1.

9.2.2 Kernel ridge regression

The hardness proof for kernel ridge regression is somewhat technical, so we only outline it here. As described in Section 9.1.2, our goal is to prove that computing the sum of the entries in the matrix K^{-1} requires quadratic time. The main idea is to leverage the hardness of computing the sum $\sum_{i,j=1}^n k(a_i, b_j)$, as shown in Lemma 9.24.

To this end, we show that the following approximate equality holds:

$$s(K_A^{-1}) + s(K_B^{-1}) - s(K_{A,B}^{-1}) \;\approx\; 2 \sum_{i,j=1}^{n} k(a_i, b_j).$$

where we use the notation $K_{A,B}, K_A, K_B$ as in the previous section. This allows us to conclude that a sufficiently accurate approximation to $s(K^{-1})$ for a kernel matrix K gives us the solution to the BHCP problem. The key observation for proving the above approximate equality is the following: if a matrix has no large off-diagonal element, i.e., it is an "almost identity", then so is its inverse.

9.2.3 Training the final layer of a neural network

We start by formally defining the class of "nice" loss functions and "nice" activation functions.

Definition 9.11. *For a label $y \in \{-1, 1\}$ and a prediction $w \in \mathbb{R}$, we call the loss function $loss(y, w) : \{-1, 1\} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$* nice *if the following three properties hold:*

- *$loss(y, w) = l(yw)$ for some convex function $l : \mathbb{R} \to \mathbb{R}_{\geq 0}$.*
- *For some sufficiently large constant $K > 0$, we have that (i) $l(x) \leq o(1)$ for all $x \geq n^K$, (ii) $l(x) \geq \omega(n)$ for all $x \leq -n^K$, and (iii) $l(x) = l(0) \pm o(1/n)$ for all $x \in \pm O(n^{-K})$.*
- *$l(0) > 0$ is some constant strictly larger than 0.*

We note that the hinge loss function $\text{loss}(y, x) = \max(0, 1 - y \cdot x)$ and the logistic loss function $\text{loss}(y, x) = \frac{1}{\ln 2} \ln\left(1 + e^{-y \cdot x}\right)$ are nice loss functions according to the above definition.

Definition 9.12. *A* non-decreasing *activation functions $S : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is "nice" if it satisfies the following property: for all sufficiently large constants $T > 0$ there exist $v_0 > v_1 > v_2$ such that $S(v_0) = \Theta(1)$, $S(v_1) = 1/n^T$, $S(v_2) = 1/n^{\omega(1)}$ and $v_1 = (v_0 + v_2)/2$.*

The ReLU activation $S(z) = \max(0, z)$ satisfies these properties since we can choose $v_0 = 1$, $v_1 = 1/n^T$, and $v_2 = -1 + 2/n^T$. For the sigmoid function $S(z) = \frac{1}{1+e^{-z}}$, we can choose $v_1 = -\log(n^T - 1)$, $v_0 = v_1 + C$, and $v_2 = v_1 - C$ for some $C = \omega(\log n)$. In the rest of the proof we set $T = 1000K$, where K is the constant from Definition 9.11.

We now describe the proof of Theorem 9.9. We use the notation $\alpha := (\alpha_1, \ldots, \alpha_n)^{\mathsf{T}}$. Invoking the first property from Definition 9.11, we observe that the optimization problem (9.2) is equivalent to the following optimization problem:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \quad \sum_{i=1}^{m} l(y_i \cdot (M\alpha)_i), \tag{9.6}$$

where $M \in \mathbb{R}^{m \times n}$ is the matrix defined as $M_{i,j} := S(u_i^{\mathsf{T}} w_j)$ for $i = 1, \ldots, m$ and $j = 1, \ldots n$. For the rest of the section we will use $m = \Theta(n)$.[1]

Let $A = \{a_1, \ldots, a_n\} \subseteq \{0, 1\}^d$ and $B = \{b_1, \ldots, b_n\} \subseteq \{0, 1\}^d$ with $d = \omega(\log n)$ be the input to the Orthogonal Vectors problem. To show hardness we define a matrix M as a vertical concatenation of 3 smaller matrices: M_1, M_2 and M_2 (repeated). $M = \begin{bmatrix} M_1 \\ M_2 \\ M_2 \end{bmatrix}$.
Both matrices $M_1, M_2 \in \mathbb{R}^{n \times n}$ are of size $n \times n$. Thus the number of rows of M (equivalently, the number of training examples) is $m = 3n$.

[1] Note that our reduction does not explicitly construct M. Instead, the values of the matrix are induced by the input examples and weights.

Reduction running time. The reduction from BHCP to the retraining problem proceeds by creating n weight vectors and $O(n)$ training examples, both of which are vectors in $\mathbb{R}^d$. The procedure for creating the examples (described in the following sections) takes $O(dn)$ time. Since we only need to assume that $d = \omega(\log n)$, it follows that a sub-quadratic algorithm for the retraining problem implies a sub-quadratic algorithm for BHCP.

Reduction overview. We select the input examples and weights so that the matrices M_1 and M_2, have the following properties:

- M_1: if two vectors a_i and b_j are orthogonal, then the corresponding entry $(M_1)_{i,j} = S(v_0) = \Theta(1)$ and otherwise $(M_1)_{i,j} \approx 0$.[1]
- M_2: $(M_2)_{i,i} = S(v_1) = 1/n^{1000K}$ and $(M_2)_{i,j} \approx 0$ for all $i \neq j$

To complete the description of the optimization problem (9.6), we assign labels to the inputs corresponding to the rows of the matrix M. We assign label 1 to all inputs corresponding to rows of the matrix M_1 and the first copy of the matrix M_2. We assign label -1 to all remaining rows of the matrix M corresponding to the second copy of matrix M_2.

The proof of the theorem is completed by the following two lemmas. See Section 9.4.5 for the proofs.

Lemma 9.13. *If there is a pair of orthogonal vectors, then the optimal value of* (9.6) *is upper bounded by* $(3n-1) \cdot l(0) + o(1)$.

Lemma 9.14. *If there is no pair of orthogonal vectors, then the optimal value of* (9.6) *is lower bounded by* $3n \cdot l(0) - o(1)$.

The intuition behind the proofs of the two lemmas is as follows. To show Lemma 9.13, it suffices to demonstrate a feasible vector α. We achieve this by setting each entry of α to a "moderately large" number. In this way we ensure that:

- At most $n-1$ entries of $M_1\alpha$ are close to 0, each contributing at most $l(0) + o(1/n)$ to the loss.
- The remaining entries of $M_1\alpha$ are positive and "large", each contributing only $o(1)$ to the loss. These entries correspond to orthogonal pairs of vectors. The total loss corresponding to M_1 is thus upper bounded by $(n-1) \cdot l(0) + o(1)$.
- The entries of the two vectors $M_2\alpha$ are close to 0, contributing approximately $2n \cdot l(0)$ to the loss.

To show Lemma 9.14, we need to argue that no vector α achieves loss smaller than $3n \cdot l(0) - o(1)$. We first observe that the two copies of M_2 contributes a loss of at least $2n \cdot l(0)$, independently of the values of α. We then show that the entries of α cannot be too large (in absolute value), as otherwise the loss incurred by the two copies of the matrix M_2 would be too large. Using this property, we show that the loss incurred by M_1 is lower bounded by $n \cdot l(0) - o(1)$.

Since $l(0) > 0$ (the third property in Definition 9.11), we can distinguish $\leq 3n \cdot l(0) + o(1)$ from $\geq (3n+1) \cdot l(0) - o(1)$ with a high-accuracy ERM solution. Hence Lemmas 9.13 and 9.14 imply that Problem (9.6) is SETH hard.

[1] We write $x \approx y$ if $x = y$ up to an inversely superpolynomial additive factor, i.e., $|x - y| \leq n^{-\omega(1)}$.

9.2.4 Gradient computation

Finally, we consider the problem of computing the gradient of the loss function for a given set of examples. We focus on the network architecture as in the previous section. Specifically, let $F_{\alpha,B}(a) := \sum_{j=1}^{n} \alpha_j S(a, b_j)$ be the output of a neural net with activation function S, where: (1) a is an input vector from the set $A := \{a_1, \dots, a_m\} \subseteq \{0,1\}^d$; (2) $B := \{b_1, \dots, b_n\} \subseteq \{0,1\}^d$ is a set of binary vectors; (3) $\alpha = \{\alpha_1, \dots, \alpha_n\}^T \in \mathbb{R}^n$ is an n-dimensional real-valued vector. We first prove the following lemma.

Lemma 9.15. *For some loss function $l : \mathbb{R} \to \mathbb{R}$, let $l(F_{\alpha,B}(a))$ be the loss for input a when the label of the input a is $+1$. Consider the gradient of the total loss $l_{\alpha,A,B} := \sum_{a \in A} l(F_{\alpha,B}(a))$ at $\alpha_1 = \ldots = \alpha_n = 0$ with respect to $\alpha_1, \dots, \alpha_n$. The sum of the entries of the gradient is equal to $l'(0) \cdot \sum_{a \in A, b \in B} S(a,b)$, where $l'(0)$ is the derivative of the loss function l at 0.*

For the hinge loss function, we have that the loss function is $l(x) = \max(0, 1-x)$ if the label is $+1$. Thus, $l'(0) = -1$. For the logistic loss function, we have that the loss function is $l(x) = \frac{1}{\ln 2} \ln\left(1 + e^{-x}\right)$ if the label is $+1$. Thus, $l'(0) = -\frac{1}{2 \ln 2}$ in this case.

Proof of Theorem 9.10. Since all ℓ_p-norms are within a polynomial factor, it suffices to show the statement for ℓ_1-norm.

We set $S(a,b) := \max(0, 1 - 2a^{\mathsf{T}} b)$. Using Lemma 9.15, we get that the ℓ_1-norm of the gradient of the total loss function is equal to $|l'(0)| \cdot \sum_{a \in A, b \in B} 1_{a^{\mathsf{T}} b = 0}$. Since $l'(0) \neq 0$, this reduces OV to the gradient computation problem. Note that if there is no orthogonal pair, then the ℓ_1-norm is 0 and otherwise it is a constant strictly greater than 0. Thus approximating the ℓ_1-norm within any finite factor allows us to distinguish the cases. □

See Section 9.4.6 for further results.

9.3 Conclusions

We have shown that a range of kernel problems require quadratic time for obtaining a high accuracy solution unless the Strong Exponential Time Hypothesis is false. These problems include variants of kernel SVM, kernel ridge regression, and kernel PCA. We also gave a similar hardness result for training the final layer of a depth-2 neural network. This result is general and applies to multiple loss and activation functions. Finally, we proved that computing the empirical loss gradient for such networks takes time that is essentially "rectangular", i.e., proportional to the product of the network size and the number of examples.

We note that our quadratic (rectangular) hardness results hold for *general* inputs. There is a long line of research on algorithms for kernel problems with running times depending on various input parameters, such as its statistical dimension [241], degrees of freedom [25] or effective dimensionality [166]. It would be interesting to establish lower bounds on the complexity of kernel problems as a function of the aforementioned input parameters.

Our quadratic hardness results for kernel problems apply to kernels with exponential tails. A natural question is whether similar results can be obtained for "heavy-tailed" kernels, e.g., the Cauchy kernel. We note that similar results for the linear kernel do not seem achievable using our techniques.[1]

Several of our results are obtained by a reduction from the (exact) Bichromatic Hamming Closest Pair problem or the Orthogonal Vectors problem. This demonstrates a strong connection between kernel methods and similarity search, and suggests that perhaps a reverse reduction is also possible. Such a reduction could potentially lead to faster approximate algorithms for kernel methods: although the exact closest pair problem has no known sub-quadratic solution, efficient and practical sub-quadratic time algorithms for the approximate version of the problem exist (see e.g., [14, 16, 17, 19, 224]).

9.4 Proof details

9.4.1 Preliminaries

In this section, we define notion used later in this chapter. We start from the soft-margin support vector machine (see [163]).

Definition 9.16 (Support Vector Machine (SVM))**.** *Let $x_1, \ldots, x_n \in \mathbb{R}^d$ be n vectors and $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels. Let $k(x, x')$ be a kernel function. An optimization problem of the following form is a (primal) SVM.*

$$\begin{aligned} \underset{\substack{\alpha_1,\ldots,\alpha_n \geq 0,\\ \xi_1,\ldots,\xi_n \geq 0}}{\mathit{minimize}}_{\ b} \quad & \frac{\lambda}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \ + \ \frac{1}{n} \sum_{i=1}^{n} \xi_i \\ \mathit{subject\ to} \quad & y_i f(x_i) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \end{aligned} \tag{9.7}$$

where $f(x) := b + \sum_{i=1}^{n} \alpha_i y_i k(x_i, x)$ and $\lambda \geq 0$ is called the regularization parameter. ξ_i are known as the slack variables.

The dual SVM is defined as

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0}{\mathit{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \mathit{subject\ to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \\ & \alpha_1, \ldots, \alpha_n \leq \frac{1}{\lambda n}. \end{aligned} \tag{9.8}$$

We refer to the quantity b as the bias term. When we require that the bias is $b = 0$, we call the optimization problem as SVM without the bias term. The primal SVM without the bias term remains the same except $f(x) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x)$. The dual SVM remains the same except we remove the equality constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$.

[1] In particular, assuming a certain strengthening of SETH, known as the "non-deterministic SETH" [63], it is provably impossible to prove SETH hardness for any of the linear variants of the studied ERM problems, at least via deterministic reductions. This is due to the fact that these problems have short certificates of optimality via duality arguments. Also, it should be noted that linear analogs of some of the problems considered in this chapter (e.g., linear ridge regression) can be solved in $O(nd^2)$ time using SVD methods.

The (primal) hard-margin SVM defined in the previous section corresponds to soft-margin SVM in the setting when $\lambda \to 0$. The dual hard-margin SVM is defined as follows.

Definition 9.17 (Dual hard-margin SVM)**.** *Let $x_1, \ldots, x_n \in \mathbb{R}^d$ be n vectors and let $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels. Let $k(x, x')$ be a kernel function. An optimization problem of the following form is a dual hard-margin SVM.*

$$\begin{aligned} &\underset{\alpha_1,\ldots,\alpha_n \geq 0}{\textit{maximize}} && \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\textit{subject to} && \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned} \tag{9.9}$$

If the primal hard-margin SVM is without the bias term ($b = 0$), then we omit the inequality constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ in the dual SVM.

We will use the following fact (see [163]).

Fact 9.18. *If $\alpha_1^*, \ldots, \alpha_n^*$ achieve the minimum in an SVM, then the same $\alpha_1^*, \ldots, \alpha_n^*$ achieve the maximum in the dual SVM. Also, the minimum value and the maximum value are equal.*

9.4.2 SVM problems

9.4.2.1 SVM without the bias term

In this section we formalize the intuition from Section 9.2.1. We start from the following two lemmas.

Lemma 9.19 (NO case)**.** *If for all $a_i \in A$ and $b_j \in B$ we have $Hamming(a_i, b_j) \geq t$, then*

$$\textit{value}(A, B) \leq \textit{value}(A) + \textit{value}(B) + 200n^6 \exp(-100 \log n \cdot t).$$

Lemma 9.20 (YES case)**.** *If there exist $a_i \in A$ and $b_j \in B$ such that $Hamming(a_i, b_j) \leq t-1$, then*

$$\textit{value}(A, B) \geq \textit{value}(A) + \textit{value}(B) + \frac{1}{4} \exp(-100 \log n \cdot (t-1)).$$

Assuming the two lemmas we can distinguish the NO case from the YES case because

$$200n^6 \exp(-100 \log n \cdot t) \ll \frac{1}{4} \exp(-100 \log n \cdot (t-1))$$

by our choice of the parameter $C = 100 \log n$ for the Gaussian kernel.

Before we proceed with the proofs of the two lemmas, we prove the following auxiliary statement.

Lemma 9.21. *Consider SVM (9.3). Let $\alpha_1^*, \ldots, \alpha_n^*$ be the setting of values for $\alpha_1, \ldots, \alpha_n$ that achieves value(A). Then for all $i = 1, \ldots, n$ we have that $n \geq \alpha_i^* \geq 1/2$.*

Analogous statement holds for SVM (9.4).

Proof. First we note that value$(A) \leq n^2/2$ because the objective value of (9.3) is at most $n^2/2$ if we set $\alpha_1 = \ldots = \alpha_n = 1$. Note that all inequalities of (9.3) are satisfied for this setting of variables. Now we lower bound value(A):

$$\text{value}(A) = \frac{1}{2}\sum_{i,j}^{n} \alpha_i^* \alpha_j^* k(a_i, a_j) \geq \frac{1}{2}\sum_{i=1}^{n} (\alpha_i^*)^2.$$

From value$(A) \geq \frac{1}{2}\sum_{i=1}^{n}(\alpha_i^*)^2$ and value$(A) \leq n^2/2$ we conclude that $\alpha_i^* \leq n$ for all i.

Now we will show that $\alpha_i^* \geq 1/2$ for all $i = 1, \ldots, n$. Consider the inequality

$$\sum_{j=1}^{n} \alpha_j^* k(a_i, a_j) = \alpha_i^* + \sum_{j \,:\, j \neq i} \alpha_j^* k(a_i, a_j) \geq 1$$

which is satisfied by $\alpha_1^*, \ldots, \alpha_n^*$ because this is an inequality constraint in (9.3). Note that $k(a_i, a_j) \leq \frac{1}{10n^2}$ for all $i \neq j$ because $C = 100 \log n$ and $\|a_i - a_j\|_2^2 = \text{Hamming}(a_i, a_j) \geq 1$ for all $i \neq j$. Also, we already obtained that $\alpha_j^* \leq n$ for all j. This gives us the required lower bound for α_i^*:

$$\alpha_i^* \geq 1 - \sum_{j \,:\, j \neq i} \alpha_j^* k(a_i, a_j) \geq 1 - n \cdot n \cdot \frac{1}{10n^2} \geq 1/2.$$

□

Additive precision For particular value of t, the sufficient additive precision for solving the three SVMs is $\frac{1}{100}\exp(-100 \log n \cdot (t-1))$ to be able to distinguish the NO case from the YES case. Since we want to be able to distinguish the two cases for any $t \in \{2, \ldots, d\}$, it suffices to have an additive precision $\exp(-100 \log n \cdot d) \leq \frac{1}{100}\exp(-100 \log n \cdot (t-1))$. From [15] we know that any $d = \omega(\log n)$ is sufficient to show hardness. Therefore, any additive approximation $\exp(-\omega(\log^2 n))$ is sufficient to show the hardness for SVM.

Multiplicative precision Consider any $\varepsilon = \exp(-\omega(\log^2 n))$ and suppose we can approximate within multiplicative factor $(1+\varepsilon)$ quantities value(A), value(B) and value(A, B). From the proof of Lemma 9.21 we know that value(A), value$(B) \leq n^2/2$. If value$(A, B) \leq 10n^2$, then $(1+\varepsilon)$-approximation of the three quantities allows us to compute the three quantities within additive $\exp(-\omega(\log^2 n))$ factor and the hardness follows from the previous paragraph. On the other hand, if value$(A, B) > 10n^2$, then $(1+\varepsilon)$-approximation of value(A, B) allows us to determine that we are in the YES case.

In the rest of the section we complete the proof of the theorem by proving Lemma 9.19 and Lemma 9.20.

Proof of Lemma 9.19. Let $\alpha_1^*, \ldots, \alpha_n^*$ and $\beta_1^*, \ldots, \beta_n^*$ be the optimal assignments to SVMs (9.3) and (9.4), respectively. We use the notation $\delta := \exp(-100 \log n \cdot t)$. Note that $k(a_i, b_j) = \exp(-100 \log n \cdot \|a_i - b_j\|_2^2) \leq \delta$ for all i, j because $||a_i - b_j||_2^2 = \text{Hamming}(a_i, b_j) \geq t$ for all i, j.

We define $\alpha_i' := \alpha_i^* + 10n^2\delta$ and $\beta_i' := \beta_i^* + 10n^2\delta$ for all $i = 1, \ldots, n$. We observe that $\alpha_i', \beta_i' \leq 2n$ for all i because $\alpha_i^*, \beta_i^* \leq n$ for all i (Lemma 9.21) and $\delta = \exp(-100 \log n \cdot t) \leq$

$\frac{1}{10n^2}$. Let V be the value of the objective function in (9.5) when evaluated on α_i' and β_i'.

We make two claims. We claim that α_i' and β_i' satisfy the inequality constraints in (9.5). This implies that $\text{value}(A, B) \le V$ since (9.5) is a minimization problem. Our second claim is that $V \le \text{value}(A) + \text{value}(B) + 200n^6\delta$. The two claims combined complete the proof of the lemma.

We start with the proof of the second claim. We want to show that $V \le \text{value}(A) + \text{value}(B) + 200n^6\delta$. We get the following inequality:

$$\begin{aligned} V &= \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i'\alpha_j' k(a_i, a_j) \;+\; \frac{1}{2}\sum_{i,j=1}^{n} \beta_i'\beta_j' k(b_i, b_j) \;-\; \sum_{i,j=1}^{n} \alpha_i'\beta_j' k(a_i, b_j) \\ &\le \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i'\alpha_j' k(a_i, a_j) \;+\; \frac{1}{2}\sum_{i,j=1}^{n} \beta_i'\beta_j' k(b_i, b_j) \end{aligned}$$

since the third sum is non-negative. It is sufficient to show two inequalities

$$\frac{1}{2}\sum_{i,j=1}^{n} \alpha_i'\alpha_j' k(a_i, a_j) \;\le\; \text{value}(A) + 100n^6\delta$$

and

$$\frac{1}{2}\sum_{i,j=1}^{n} \beta_i'\beta_j' k(b_i, b_j) \;\le\; \text{value}(B) + 100n^6\delta$$

to establish the inequality $V \le \text{value}(A) + \text{value}(B) + 200n^6\delta$. We prove the first inequality. The proof for the second inequality is analogous. We use the definition of $\alpha_i' = \alpha_i^* + 10n^2\delta$:

$$\begin{aligned} &\frac{1}{2}\sum_{i,j=1}^{n} \alpha_i'\alpha_j' k(a_i, a_j) \\ &= \frac{1}{2}\sum_{i,j=1}^{n} (\alpha_i^* + 10n^2\delta)(\alpha_j^* + 10n^2\delta) k(a_i, a_j) \\ &\le \frac{1}{2}\sum_{i,j=1}^{n} \Big(\alpha_i^*\alpha_j^* k(a_i, a_j) + 20n^3\delta + 100n^4\delta^2\Big) \\ &\le \text{value}(A) + 100n^6\delta, \end{aligned}$$

where in the first inequality we use that $\alpha_i^* \le n$ and $k(a_i, a_j) \le 1$.

Now we prove the first claim. We show that the inequality constraints are satisfied by α_i' and β_i'. We prove that the inequality

$$\sum_{j=1}^{n} \alpha_j' k(a_i, a_j) - \sum_{j=1}^{n} \beta_j' k(a_i, b_j) \ge 1 \tag{9.10}$$

is satisfied for all $i = 1, \ldots, n$. The proof that the inequalities

$$-\sum_{j=1}^{n} \beta_j' k(b_i, b_j) + \sum_{j=1}^{n} \alpha_j' k(b_i, a_j) \;\le\; -1$$

are satisfied is analogous.

We lower bound the first sum of the left hand side of (9.10) by repeatedly using the definition of $\alpha'_i = \alpha^*_i + 10n^2\delta$:

$$\begin{aligned}
&\sum_{j=1}^{n} \alpha'_j k(a_i, a_j) \\
&= (\alpha^*_i + 10n^2\delta) + \sum_{j\,:\,j\neq i} \alpha'_j k(a_i, a_j) \\
&\geq 10n^2\delta + \alpha^*_i + \sum_{j\,:\,j\neq i} \alpha^*_j k(a_i, a_j) \\
&= 10n^2\delta + \sum_{j=1}^{n} \alpha^*_j k(a_i, a_j) \\
&\geq 1 + 10n^2\delta.
\end{aligned}$$

In the last inequality we used the fact that α^*_i satisfy the inequality constraints of SVM (9.3).

We upper bound the second sum of the left hand side of (9.10) by using the inequality $\beta'_j \leq 2n$ and $k(a_i, b_j) \leq \delta$ for all i, j:

$$\sum_{j=1}^{n} \beta'_j k(a_i, b_j) \leq 2n^2\delta.$$

Finally, we can show that the inequality constraint is satisfied:

$$\sum_{j=1}^{n} \alpha'_j k(a_i, a_j) - \sum_{j=1}^{n} \beta'_j k(a_i, b_j) \geq 1 + 10n^2\delta - 2n^2\delta \geq 1.$$

□

Proof of Lemma 9.20. To analyze the YES case, we consider the dual SVMs (see Definition 9.17) of the three SVMs (9.3), (9.4) and (9.5):

1. The dual SVM of SVM (9.3):

$$\underset{\alpha_1,\ldots,\alpha_n \geq 0}{\text{maximize}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j k(a_i, a_j). \tag{9.11}$$

2. The dual SVM of SVM (9.4):

$$\underset{\beta_1,\ldots,\beta_n \geq 0}{\text{maximize}} \quad \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j k(a_i, a_j). \tag{9.12}$$

3. The dual SVM of SVM (9.5):

$$\begin{aligned}\underset{\substack{\alpha_1,\ldots,\alpha_n\geq 0\\ \beta_1,\ldots,\beta_n\geq 0}}{\text{maximize}}\quad &\sum_{i=1}^n \alpha_i \;+\; \sum_{i=1}^n \beta_i \;-\; \frac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j k(a_i,a_j)\\ &-\;\frac{1}{2}\sum_{i,j=1}^n \beta_i\beta_j k(b_i,b_j) \;+\; \sum_{i,j=1}^n \alpha_i\beta_j k(a_i,b_j).\end{aligned} \tag{9.13}$$

Since the optimal values of the primal and dual SVMs are equal, we have that value(A), value(B) and value(A, B) are equal to optimal values of dual SVMs (9.11), (9.12) and (9.13), respectively (see Fact 9.18).

Let $\alpha_1^*, \ldots, \alpha_n^*$ and $\beta_1^*, \ldots, \beta_n^*$ be the optimal assignments to dual SVMs (9.11) and (9.12), respectively.

Our goal is to lower bound value(A, B). Since (9.13) is a maximization problem, it is sufficient to show an assignment to α_i and β_j that gives a large value to the objective function. For this we set $\alpha_i = \alpha_i^*$ and $\beta_j = \beta_j^*$ for all $i, j = 1, \ldots, n$. This gives the following inequality:

$$\begin{aligned}\text{value}(A,B) \geq &\sum_{i=1}^n \alpha_i^* \;+\; \sum_{i=1}^n \beta_i^* \;-\; \frac{1}{2}\sum_{i,j=1}^n \alpha_i^*\alpha_j^* k(a_i,a_j)\\ &-\;\frac{1}{2}\sum_{i,j=1}^n \beta_i^*\beta_j^* k(b_i,b_j) \;+\; \sum_{i,j=1}^n \alpha_i^*\beta_j^* k(a_i,b_j)\\ \geq&\; \text{value}(A) + \text{value}(B) + \sum_{i,j=1}^n \alpha_i^*\beta_j^* k(a_i,b_j),\end{aligned}$$

where we use the fact that value(A) and value(B) are the optimal values of dual SVMs (9.11) and (9.12), respectively.

To complete the proof of the lemma, it suffices to show the following inequality:

$$\sum_{i,j=1}^n \alpha_i^*\beta_j^* k(a_i,b_j) \geq \frac{1}{4}\exp(-100\log n \cdot (t-1)). \tag{9.14}$$

Notice that so far we did not use the fact that there is a close pair of vectors $a_i \in A$ and $b_j \in B$ such that Hamming$(a_i, b_j) \leq t-1$. We use this fact now. We lower bound the left hand side of (9.14) by the summand corresponding to the close pair:

$$\sum_{i,j=1}^n \alpha_i^*\beta_j^* k(a_i,b_j) \geq \alpha_i^*\beta_j^* k(a_i,b_j) \geq \alpha_i^*\beta_j^* \exp(-100\log n \cdot (t-1)),$$

where in the last inequality we use Hamming$(a_i, b_j) \leq t-1$ and the definition of the Gaussian kernel.

The proof is completed by observing that $\alpha_i^* \geq \frac{1}{2}$ and $\beta_i^* \geq \frac{1}{2}$ which follows from Fact 9.18 and Lemma 9.21. □

9.4.2.2 SVM with the bias term

In the previous section we showed hardness for SVM without the bias term. In this section we show hardness for SVM with the bias term.

Theorem 9.22. *Let $x_1, \ldots, x_n \in \{-1,0,1\}^d$ be n vectors and let $y_1, \ldots, y_n \in \{-1,1\}$ be n labels.*

Let $k(a,a') = \exp\left(-C\|a-a'\|_2^2\right)$ be the Gaussian kernel with $C = 100\log n$.

Consider the corresponding hard-margin SVM with the bias term:

$$\begin{aligned} &\underset{\alpha_1,\ldots,\alpha_n \geq 0,\quad b}{\text{minimize}} && \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j k(x_i,x_j) \\ &\text{subject to} && y_i f(x_i) \geq 1, \quad i = 1,\ldots,n, \end{aligned} \tag{9.15}$$

where $f(x) := b + \sum_{i=1}^n \alpha_i y_i k(x_i, x)$.

Consider any $\varepsilon = \exp(-\omega(\log^2 n))$. Approximating the optimal value of (9.15) within the multiplicative factor $(1+\varepsilon)$ requires almost quadratic time assuming SETH. This holds for the dimensionality $d = O(\log^3 n)$ of the input vectors.

The same hardness result holds for any additive $\exp(-\omega(\log^2 n))$ approximation factor.

Proof. Consider a hard instance from Theorem 9.4 for SVM without the bias term. Let $x_1, \ldots, x_n \in \{0,1\}^d$ be the n binary vectors of dimensionality $d = \omega(\log n)$ and $y_1, \ldots, y_n \in \{-1,1\}$ be the n corresponding labels. For this input consider the dual SVM without the bias term (see Definition 9.17):

$$\underset{\gamma_1,\ldots,\gamma_n \geq 0}{\text{maximize}} \quad \sum_{i=1}^{n} \gamma_i \; - \; \frac{1}{2}\sum_{i,j=1}^{n} \gamma_i\gamma_j y_i y_j k(x_i,x_j). \tag{9.16}$$

We will show how to reduce SVM without the bias term (9.16) to SVM with the bias term. By Theorem 9.4 this will give hardness result for SVM with the bias term. We start with a simpler reduction that will achieve almost what we need except the entries of the vectors will not be from the set $\{-1,0,1\}$. Then we will show how to change the reduction to fix this.

Consider $2n$ vectors

$$x_1, \ldots, x_n, -x_1, \ldots, -x_n \in \{-1,0,1\}^d$$

with $2n$ labels $y_1, \ldots, y_n, -y_1, \ldots, -y_n \in \{-1,1\}$. Consider an SVM with the bias term for the $2n$ vectors, that is, an SVM of the form (9.15). From Definition 9.17 we know that its dual SVM is

$$\begin{aligned}\underset{\substack{\alpha_1,\ldots,\alpha_n\geq 0\\ \beta_1,\ldots,\beta_n\geq 0}}{\text{maximize}} \quad & \sum_{i=1}^{n}\alpha_i \;+\; \sum_{j=1}^{n}\beta_j \;-\; \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j k(x_i,x_j)\\ & -\;\frac{1}{2}\sum_{i,j=1}^{n}\beta_i\beta_j y_i y_j k(x_i,x_j) \;+\; \sum_{i,j=1}^{n}\alpha_i\beta_j y_i y_j k(x_i,-x_j) \qquad (9.17)\\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i y_i = \sum_{j=1}^{n}\beta_j y_j.\end{aligned}$$

Consider any setting of values for α_i and β_j. Notice that if we swap the value of α_i and β_i for every i, the value of the objective function of (9.17) does not change. This is implies that we can define $\gamma_i := \frac{\alpha_i+\beta_i}{2}$ and set $\alpha_i = \beta_i = \gamma_i$ for every i. Because of the convexity of the optimization problem, the value of the objective function can only increase after this change. Clearly, the equality constraint will be satisfied. Therefore, w.l.o.g. we can assume that $\alpha_i = \beta_i = \gamma_i$ for some γ_i and we can omit the equality constraint.

We rewrite (9.17) in terms of γ_i and divide the objective function by 2:

$$\underset{\gamma_1,\ldots,\gamma_n\geq 0}{\text{maximize}} \quad \sum_{i=1}^{n}\gamma_i \;-\; \frac{1}{2}\sum_{i,j=1}^{n}\gamma_i\gamma_j y_i y_j k(x_i,x_j) \;+\; \frac{1}{2}\sum_{i,j=1}^{n}\gamma_i\gamma_j y_i y_j k(x_i,-x_j). \qquad (9.18)$$

Notice that (9.18) and (9.16) are almost the same. The only difference is the third term

$$\frac{1}{2}\sum_{i,j=1}^{n}\gamma_i\gamma_j y_i y_j k(x_i,-x_j)$$

in (9.18). We can make this term to be equal to 0 and not change the first two terms as follows. We append an extra coordinate to every vector x_i and set this coordinate to be large enough value M. If we set $M = +\infty$, the third term becomes 0. The first term does not depend on the vectors. The second term depends only on the distances between the vectors (which are not affected by adding the same entry to all vectors). Thus, the first two terms do not change after this modification.

We showed that we can reduce SVM without the bias term (9.16) to the SVM with the bias term (9.17). By combining this reduction with Theorem 9.4 we obtain hardness for SVM with the bias term. This is almost what we need except that the reduction presented above produces vectors with entries that are not from the set $\{-1,0,1\}$. In every vector x_i or $-x_i$ there is an entry that has value M or $-M$, respectively. In the rest of the proof we show how to fix this, by bounding M by $O(\log^3 n)$ and distributing its contribution over $O(\log^3 n)$ coordinates.

Final reduction The final reduction is as follows:

- Take a hard instance for the SVM without the bias term from Theorem 9.4. Let $x_1,\ldots,x_n \in \{0,1\}^d$ be the n binary vectors of dimensionality $d = \omega(\log n)$ and $y_1,\ldots,y_n \in \{-1,1\}$ be the n corresponding labels.

- Append $\log^3 n$ entries to each of the vectors x_i, $i = 1, \ldots, n$ and set the entries to be 1.
- Solve SVM (9.15) on the $2n$ vectors $x_1, \ldots, x_n, -x_1, \ldots, -x_n \in \{-1, 0, 1\}^d$ with $2n$ labels $y_1, \ldots, y_n, -y_1, \ldots, -y_n \in \{-1, 1\}$. Let V be the optimal value of the SVM divided by 2.
- Output V.

Correctness of the reduction From the above discussion we know that we output the optimal value V of the optimization problem (9.18). Let V' be the optimal value of SVM (9.16).

By Theorem 9.4, it is sufficient to show that $|V - V'| \leq \exp(-\omega(\log^2 n))$ to establish hardness for SVM with the bias term. We will show that $|V - V'| \leq n^{O(1)} \exp(-\log^3 n)$. This gives hardness for additive approximation of SVM with the bias term. However, $|V - V'| \leq \exp(-\omega(\log^2 n))$ is also sufficient to show hardness for multiplicative approximation (see the discussion on the approximation in the proof of Theorem 9.4).

In the rest of the section we show that $|V - V'| \leq n^{O(1)} \exp(-\log^3 n)$. Let γ'_i be the assignment to γ_i that achieves V' in SVM (9.16). Let γ^*_i be the assignment to γ_i that achieves V in (9.18). We will show that $\gamma'_i \leq O(n)$ for all $i = 1, \ldots, n$. It is also true that $\gamma^*_i \leq O(n)$ for all $i = 1, \ldots, n$ and the proof is analogous. Since $x_1, \ldots, x_n$ are different binary vectors and $k(x_i, x_j)$ is the Gaussian kernel with the parameter $C = 100 \log n$, we have that $k(x_i, x_j) \leq 1/n^{10}$ for all $i \neq j$. This gives the following upper bound:

$$V' = \sum_{i=1}^{n} \gamma'_i \; - \; \frac{1}{2} \sum_{i,j=1}^{n} \gamma'_i \gamma'_j y_i y_j k(x_i, x_j) \leq \sum_{i=1}^{n} \left(\gamma'_i - \left(\frac{1}{2} - o(1) \right) (\gamma'_i)^2 \right).$$

Observe that every non-negative summand on the right hand side is at most $O(1)$. Therefore, if there exists i such that $\gamma'_i \geq \omega(n)$, then the right hand side is negative. This contradicts the lower bound $V' \geq 0$ (which follows by setting all γ_i to be 0 in (9.16)).

By plugging γ'_i into (9.18) and using the fact that $\gamma'_i \leq O(n)$, we obtain the following inequality:

$$V \geq V' + \frac{1}{2} \sum_{i,j=1}^{n} \gamma'_i \gamma'_j y_i y_j k(x_i, -x_j) \geq V' - n^{O(1)} \exp(-\log^3 n). \tag{9.19}$$

In the last inequality we use $k(x_i, -x_j) \leq \exp(-\log^3 n)$ which holds for all $i, j = 1, ..., n$ (observe that each x_i and x_j ends with $\log^3 n$ entries 1 and use the definition of the Gaussian kernel).

Similarly, by plugging γ^*_i into (9.16) and using the fact that $\gamma^*_i \leq O(n)$, we obtain the following inequality:

$$V' \geq V - \frac{1}{2} \sum_{i,j=1}^{n} \gamma^*_i \gamma^*_j y_i y_j k(x_i, -x_j) \geq V - n^{O(1)} \exp(-\log^3 n). \tag{9.20}$$

Inequalities (9.19) and (9.20) combined give the desired inequality

$$|V - V'| \leq n^{O(1)} \exp(-\log^3 n) \; .$$

□

9.4.2.3 Soft-margin SVM

Theorem 9.23. *Let $x_1, \ldots, x_n \in \{-1, 0, 1\}^d$ be n vectors and let $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels.*

Let $k(a, a') = \exp\left(-C\|a - a'\|_2^2\right)$ be the Gaussian kernel with $C = 100 \log n$.

Consider the corresponding soft-margin SVM with the bias term:

$$\begin{aligned} &\underset{\substack{\alpha_1,\ldots,\alpha_n \geq 0, \\ \xi_1,\ldots,\xi_n \geq 0}}{\text{minimize}}\; b \quad && \frac{\lambda}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \; + \; \frac{1}{n} \sum_{i=1}^{n} \xi_i \\ &\text{subject to} && y_i f(x_i) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \end{aligned} \tag{9.21}$$

where $f(x) := b + \sum_{i=1}^{n} \alpha_i y_i k(x_i, x)$.

Consider any $\varepsilon = \exp(-\omega(\log^2 n))$ and any $0 < \lambda \leq \frac{1}{Kn^2}$ for a large enough constant $K > 0$. Approximating the optimal value of (9.21) within the multiplicative factor $(1 + \varepsilon)$ requires almost quadratic time assuming SETH. This holds for the dimensionality $d = O(\log^3 n)$ of the input vectors.

The same hardness result holds for any additive $\exp(-\omega(\log^2 n))$ approximation factor.

Proof. Consider the hard instance from Theorem 9.22 for the hard-margin SVM. The dual of the hard-margin SVM is (9.17). From the proof we know that the optimal α_i and β_i satisfy $\alpha_i = \beta_i = \gamma_i^* \leq 2Kn$ for some large enough constant $K > 0$ for all $i = 1, \ldots, n$. Thus, w.l.o.g. we can add these inequalities to the set of constraints. We compare the resulting dual SVM to Definition 9.16 and conclude that the resulting dual SVM is a dual of a *soft-margin* SVM with the regularization parameter $\lambda = \frac{1}{Kn^2}$. Therefore, the hardness follows from Theorem 9.22. □

9.4.3 Kernel PCA

In this section, we present the full proof of quadratic hardness for Kernel PCA. It will also be helpful for Kernel Ridge Regression in the next section.

Given a matrix X, we denote its trace (the sum of the diagonal entries) by $\mathrm{tr}(X)$ and the total sum of its entries by $s(X)$. In the context of the matrix K' defining our problem, we have the following equality:

$$\begin{aligned} \mathrm{tr}(K') &= \mathrm{tr}((I - 1_n)K(I - 1_n)) \\ &= \mathrm{tr}(K(I - 1_n)^2) = \mathrm{tr}(K(I - 1_n)) \\ &= \mathrm{tr}(K) - \mathrm{tr}(K1_n) = n - s(K)/n \; . \end{aligned}$$

Since the sum of the eigenvalues is equal to the trace of the matrix and $\mathrm{tr}(K') = n - s(K)/n$, it is sufficient to show hardness for computing $s(K)$. The following lemma completes the proof of the theorem.

Lemma 9.24. *Computing $s(K)$ within multiplicative error $1+\varepsilon$ for $\varepsilon = \exp(-\omega(\log^2 n))$ requires almost quadratic time assuming SETH.*

Proof. As for SVMs, we will reduce the BHCP problem to the computation of $s(K)$. Let A and B be the two sets of n binary vectors coming from an instance of the BHCP problem. Let $K_A, K_B \in \mathbb{R}^{n\times n}$ be the kernel matrices corresponding to the sets A and B, respectively. Let $K_{A,B} \in \mathbb{R}^{2n\times 2n}$ be the kernel matrix corresponding to the set $A \cup B$. We observe that

$$\begin{aligned} s :=& (s(K_{A,B}) - s(K_A) - s(K_B))/2 \\ =& \sum_{i,j=1}^{n} k(a_i, b_j) \\ =& \sum_{i,j=1}^{n} \exp(-C||a_i - b_j||_2^2). \end{aligned}$$

Now we consider two cases.

Case 1. There are no close pairs, that is, for all $i, j = 1, \ldots, n$ we have $||a_i - b_j||_2^2 \geq t$ and $\exp(-C||a_i - b_j||_2^2) \leq \exp(-Ct) =: \delta$. Then $s \leq n^2\delta$.

Case 2. There is a close pair. That is, $||a_{i'} - b_{j'}||_2^2 \leq t - 1$ for some i', j'. This implies that $\exp(-C||a_{i'} - b_{j'}||_2^2) \geq \exp(-C(t-1)) =: \Delta$. Thus, $s \geq \Delta$.

Since $C = 100 \log n$, we have that $\Delta \geq n^{10}\delta$ and we can distinguish the two cases.

Precision. To distinguish $s \geq \Delta$ from $s \leq n^2\delta$, it is sufficient that $\Delta \geq 2n^2\delta$. This holds for $C = 100 \log n$. The sufficient additive precision is $\exp(-Cd) = \exp(-\omega(\log^2 n))$. Since $s(K) \leq O(n^2)$ for any Gaussian kernel matrix K, we also get that $(1+\varepsilon)$ multiplicative approximation is sufficient to distinguish the cases for any $\varepsilon = \exp(-\omega(\log^2 n))$. □

9.4.4 Kernel ridge regression

We start with stating helpful definitions and lemmas.

We will use the following lemma which is a consequence of the binomial inverse theorem.

Lemma 9.25. *Let X and Y be two square matrices of equal size. Then the following equality holds:*

$$(X + Y)^{-1} = X^{-1} - X^{-1}(I + YX^{-1})^{-1}YX^{-1}.$$

Definition 9.26 (Almost identity matrix)**.** *Let $X \in \mathbb{R}^{n\times n}$ be a matrix. We call it* almost identity matrix *if $X = I + Y$ and $|Y_{i,j}| \leq n^{-\omega(1)}$ for all $i, j = 1, \ldots, n$.*

We will need the following two lemmas.

Lemma 9.27. *The product of two almost identity matrices is an almost identity matrix.*

Proof. Follows easily from the definition. □

Lemma 9.28. *The inverse of an almost identity matrix is an almost identity matrix.*

Proof. Let X be an almost identity matrix. We want to show that X^{-1} is an almost identity matrix. We write $X = I - Y$ such that $|Y_{i,j}| \leq n^{-\omega(1)}$ for all $i, j = 1, \ldots, n$. We have the following matrix equality

$$X^{-1} = (I - Y)^{-1} = I + Y + Y^2 + Y^3 + \ldots$$

To show that X^{-1} is an almost identity, we will show that the absolute value of every entry of $Z := Y + Y^2 + Y^3 + \ldots$ is at most $n^{-\omega(1)}$. Let $\varepsilon \leq n^{-\omega(1)}$ is an upper bound on $|Y_{i,j}|$ for all $i, j = 1, \ldots, n$. Then $|Z_{i,j}| \leq Z'_{i,j}$, where $Z' := Y' + (Y')^2 + (Y')^3 + \ldots$ and Y' is a matrix consisting of entries that are all equal to ε. The proof follows since $Z'_{i,j} = \sum_{k=1}^{\infty} \varepsilon^k n^{k-1} \leq 10\varepsilon \leq n^{-\omega(1)}$. □

In the rest of the section we prove Theorem 9.6.

Proof of Theorem 9.6. We reduce the BHCP problem to the problem of computing the sum of the entries of K^{-1}.

Let A and B be the two sets of binary vectors from the BHCP instance. Let $K \in \mathbb{R}^{2n \times 2n}$ be the corresponding kernel matrix. We can write the kernel matrix K as combination of four smaller matrices $K^{1,1}, K^{1,2}, K^{2,1}, K^{2,2} \in \mathbb{R}^{n \times n}$:

$$K = \left[\begin{array}{c|c} K^{1,1} & K^{1,2} \\ \hline K^{2,1} & K^{2,2} \end{array}\right].$$

$K^{1,1}$ is the kernel matrix for the set of vectors A and $K^{2,2}$ is the kernel matrix for the set of vectors B. We define two new matrices $X, Y \in \mathbb{R}^{2n \times 2n}$: $X = \left[\begin{array}{c|c} K^{1,1} & 0 \\ \hline 0 & K^{2,2} \end{array}\right]$ and $Y = \left[\begin{array}{c|c} 0 & K^{1,2} \\ \hline K^{2,1} & 0 \end{array}\right]$.

For any matrix Z, let $s(Z)$ denote the sum of all entries of Z. Using Lemma 9.25, we can write K^{-1} as follows:

$$K^{-1} = (X + Y)^{-1} = X^{-1} - X^{-1}(I + YX^{-1})^{-1}YX^{-1}.$$

We note that the matrix X is an almost identity and that $|Y_{i,j}| \leq n^{-\omega(1)}$ for all $i, j = 1, \ldots, 2n$. This follows from the fact that we use the Gaussian kernel function with the parameter $C = \omega(\log n)$ and the input vectors are binary. Combining this with lemmas 9.27 and 9.28 allows us to conclude that matrices $X^{-1}(I + YX^{-1})^{-1}$ and X^{-1} are almost identity. Since all entries of the matrix Y are non-negative, we conclude that

$$s(X^{-1}(I + YX^{-1})^{-1}YX^{-1}) = s(Y)(1 \pm n^{-\omega(1)}).$$

We obtain that

$$\begin{aligned}
s(K^{-1}) &= s(X^{-1}) - s(X^{-1}(I + YX^{-1})^{-1}YX^{-1}) \\
&= s(X^{-1}) - s(Y)(1 \pm n^{-\omega(1)}) \\
&= s\left((K^{1,1})^{-1}\right) + s\left((K^{2,2})^{-1}\right) - s(Y)(1 \pm n^{-\omega(1)}).
\end{aligned}$$

Fix any $\alpha = \exp(-\omega(\log^2 n))$. Suppose that we can estimate each $s(K^{-1})$, $s\left((K^{1,1})^{-1}\right)$ and $s\left((K^{2,2})^{-1}\right)$ within the additive factor of α. This allows us to estimate $s(Y)$ within the additive factor of 10α. This is enough to solve the BHCP problem. We consider two cases.

Case 1 There are no close pairs, that is, for all $i, j = 1, \ldots, n$ we have $||a_i - b_j||_2^2 \geq t$ and $\exp(-C||a_i - b_j||_2^2) \leq \exp(-Ct) =: \delta$. Then $s(Y) \leq 2n^2\delta$.

Case 2 There is a close pair. That is, $||a_{i'} - b_{j'}||_2^2 \leq t - 1$ for some i', j'. This implies that $\exp(-C||a_{i'} - b_{j'}||_2^2) \geq \exp(-C(t-1)) =: \Delta$. Thus, $s(Y) \geq \Delta$.

Since $C = \omega(\log n)$, we have that $\Delta \geq 100n^2\delta$ and we can distinguish the two cases assuming that the additive precision $\alpha = \exp(-\omega(\log^2 n))$ is small enough.

Precision To distinguish $s(Y) \leq 2n^2\delta$ from $s(Y) \geq \Delta$, it is sufficient that $\Delta \geq 100n^2\delta$ and $\alpha \leq \Delta/1000$. We know that $\Delta \geq 100n^2\delta$ holds because $C = \omega(\log n)$. Since $\Delta \leq \exp(-Cd)$, we want to choose C and d such that the $\alpha \leq \Delta/1000$ is satisfied. We can do that because we can pick C to be any $C = \omega(\log n)$ and the BHCP problem requires almost quadratic time assuming SETH for any $d = \omega(\log n)$.

We get that additive ε approximation is sufficient to distinguish the cases for any $\varepsilon = \exp(-\omega(\log^2 n))$. We observe that $s(K^{-1}) \leq O(n)$ for any almost identity matrix K. This means that $(1+\varepsilon)$ multiplicative approximation is sufficient for the same ε. This completes the proof of the theorem. □

9.4.5 Training of the final layer of a neural network

Recall that the trainable parameters are $\alpha := (\alpha_1, \ldots, \alpha_n)^\mathsf{T}$, and that the optimization problem (9.2) is equivalent to the following optimization problem:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \quad \sum_{i=1}^{m} l(y_i \cdot (M\alpha)_i), \tag{9.22}$$

where $M \in \mathbb{R}^{m \times n}$ is the matrix defined as $M_{i,j} := S(u_i^\mathsf{T} w_j)$ for $i = 1, \ldots, m$ and $j = 1, \ldots n$. For the rest of the section we will use $m = \Theta(n)$.

Let $A = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^d$ and $B = \{b_1, \ldots, b_n\} \subseteq \{0,1\}^d$ with $d = \omega(\log n)$ be the input to the Orthogonal Vectors problem. We construct a matrix M as a combination of three smaller matrices:

$$M = \begin{bmatrix} M_1 \\ M_2 \\ M_2 \end{bmatrix}.$$

Both matrices $M_1, M_2 \in \mathbb{R}^{n \times n}$ are of size $n \times n$. Thus we have that the number of rows of M is $m = 3n$.

We describe the two matrices M_1, M_2 below. Recall that v_0, v_1, and v_2 are given in Definition 9.12.

- $(M_1)_{i,j} = S\left(v_0 - (v_2 - v_0) \cdot a_i^\intercal b_j\right)$. For any two real values $x, y \in \mathbb{R}$ we write $x \approx y$ if $x = y$ up to an inversely superpolynomial additive factor. In other words, $|x - y| \leq n^{-\omega(1)}$. We observe that if two vectors a_i and b_j are orthogonal, then the corresponding entry $(M_1)_{i,j} = S(v_0) = \Theta(1)$ and otherwise $(M_1)_{i,j} \approx 0$. We will show that an $\left(1 + \frac{1}{4n}\right)$-approximation of the optimal value of the optimization problem (9.22) will allow us to decide whether there is an entry in M_1 that is $S(v_0) = \Theta(1)$. This will give the required hardness.

  It remains to show how to construct the matrix M_1 using a neural network. We set the weights for the j-th hidden unit to be $\begin{bmatrix} b_j \\ 1 \end{bmatrix}$. That is, d weights are specified by the vector b_j, and we add one more input with weight 1. The i-th example (corresponding to the i-th row of the matrix M_1) is the vector $\begin{bmatrix} -(v_2 - v_0)a_i \\ v_0 \end{bmatrix}$. The output of the j-th unit on this example (which corresponds to entry $(M_1)_{i,j}$) is equal to

  $$\begin{aligned} S\left(\begin{bmatrix} -(v_2 - v_0)a_i \\ v_0 \end{bmatrix}^\intercal \begin{bmatrix} b_j \\ 1 \end{bmatrix}\right) &= S\left(v_0 - (v_2 - v_0) \cdot a_i^\intercal b_j\right) \\ &= (M_1)_{i,j} \end{aligned}$$

  as required.

- $(M_2)_{i,j} = S\left(v_1 - (v_2 - v_1) \cdot \bar{b_i}^\intercal b_j\right)$, where $\bar{b_i}$ is a binary vector obtained from the binary vector b_i by complementing all bits. We observe that this forces the diagonal entries of M_2 to be equal to $(M_2)_{i,i} = S(v_1) = 1/n^{1000K}$ for all $i = 1, \ldots, n$ and the off-diagonal entries to be $(M_2)_{i,j} \approx 0$ for all $i \neq j$.[1]

To complete the description of the optimization problem (9.22), we assign labels to the inputs corresponding to the rows of the matrix M. We assign label 1 to all inputs corresponding to rows of the matrix M_1 and the first copy of the matrix M_2. We assign label -1 to all remaining rows of the matrix M corresponding to the second copy of matrix M_2.

It now suffices to prove Lemma 9.13 and Lemma 9.14.

Proof of Lemma 9.13. To obtain an upper bound on the optimal value in the presence of an orthogonal pair, we set the vector α to have all entries equal to n^{100K}. For this α we have

[1] For all $i \neq j$ we have $\bar{b_i}^\intercal b_j \geq 1$. This holds because all vectors b_i are distinct and have the same number of 1s.

- $|(M_1\alpha)_i| \geq \Omega(n^{100K})$ for all $i = 1, \dots, n$ such that there is exists $j = 1, \dots, n$ with $a_i^\intercal b_j = 0$. Let $x \geq 1$ be the number of such i.
- $|(M_1\alpha)_i| \leq n^{-\omega(1)}$ for all $i = 1, \dots, n$ such that there is no $j = 1, \dots, n$ with $a_i^\intercal b_j = 0$. The number of such i is $n - x$.

By using the second property of Definition 9.11, the total loss corresponding to M_1 is upper bounded by

$$\begin{aligned} x \cdot l(\Omega(n^{100K})) + (n-x) \cdot l(n^{-\omega(1)}) &\leq x \cdot o(1) + (n-x) \cdot (l(0) + o(1/n)) \\ &\leq (n-1) \cdot l(0) + o(1) =: l_1. \end{aligned}$$

Finally, the total loss corresponding to the two copies of the matrix M_2 is upper bounded by

$$\begin{aligned} 2n \cdot l(\pm O(n^{-800K})) &= 2n \cdot (l(0) \pm o(1/n)) \\ &\leq 2n \cdot l(0) + o(1) =: l_2. \end{aligned}$$

The total loss corresponding to the matrix M is upper bounded by $l_1 + l_2 \leq (3n-1) \cdot l(0) + o(1)$ as required. □

Proof of Lemma 9.14. We first observe that the total loss corresponding to the two copies of the matrix M_2 is lower bounded by $2n \cdot l(0)$. Consider the i-th row in both copies of matrix M_2. By using the convexity of the function l, the loss corresponding to the two rows is lower bounded by $l((M_2\alpha)_i) + l(-(M_2\alpha)_i) \geq 2 \cdot l(0)$. By summing over all n pairs of rows we obtain the required lower bound on the loss.

We claim that $\|\alpha\|_\infty \leq n^{10^6 K}$. Suppose that this is not the case and let i be the index of the largest entry of α in magnitude. Then the i-th entry of the vector $M_2\alpha$ is

$$\begin{aligned} (M_2\alpha)_i &= \alpha_i (M_2)_{i,i} \pm n \cdot \alpha_i \cdot n^{-\omega(1)} \\ &\geq \frac{\alpha_i}{n^{1000K}} - \alpha_i n^{-\omega(1)} , \end{aligned}$$

where we recall that the diagonal entries of matrix M_2 are equal to $(M_2)_{i,i} = S(v_1) = 1/n^K$. If $|\alpha_i| > n^{10^6 K}$, then $|(M_2\alpha)_i| \geq n^{1000k}$. However, by the second property in Definition 9.11, this implies that the loss is lower bounded by $\omega(n)$ for the i-row (for the first or the second copy of M_2). This contradicts a simple lower bound of $4n \cdot l(0)$ on the loss obtained by setting $\alpha = 0$ to be the all 0s vector. We use the third property of a nice loss function which says that $l(0) > 0$.

For the rest of the proof, we assume that $\|\alpha\|_\infty \leq n^{10^6 K}$. We will show that the total loss corresponding to M_1 is lower bounded by $n \cdot l(0) - o(1)$. This is sufficient since we already showed that the two copies of M_2 contribute a loss of at least $2n \cdot l(0)$.

Since all entries of the matrix M_1 are inversely superpolynomial (there is no pair of orthogonal vectors), we have that $|(M_1\alpha)_i| \leq n^{-\omega(1)}$ for all $i = 1, \dots, n$. Using the second property again, the loss corresponding to M_1 is lower bounded by

$$\begin{aligned} n \cdot l(\pm n^{-\omega(1)}) &\geq n \cdot (l(0) - o(1/n)) \\ &\geq n \cdot l(0) - o(1) \end{aligned}$$

as required. □

9.4.6 Gradient computation

We start from the proof of Lemma 9.15.

Proof.

$$\frac{\partial l_{\alpha,A,B}}{\partial \alpha_j} = \sum_{a\in A} \frac{\partial l(F_{\alpha,B}(a))}{\partial F_{\alpha,B}(a)} S(a,b_j) = l'(0)\cdot \sum_{a\in A} S(a,b_j) \qquad \text{(since } F_{\alpha,B}(a)=0).$$

□

Sigmoid activation function We can show our hardness result holds also for the sigmoid activation function.

Theorem 9.29. *Consider a neural net with of size n with the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$. Approximating the ℓ_p norm (for any $p \geq 1$) of the gradient of the empirical risk for m examples within the multiplicative factor of n^C for any constant $C > 0$ takes at least $O\left((nm)^{1-o(1)}\right)$ time assuming SETH.*

Proof. We set $S(a,b) := \sigma(-10(C+1)(\log n)\cdot a^{\mathsf{T}} b)$. Using Lemma 9.15, we get that the ℓ_1 norm of the gradient is equal to $|l'(0)| \cdot \sum_{a\in A, b\in B} \frac{1}{1+e^{10(C+1)(\log n)\cdot a^{\mathsf{T}} b}}$. It is easy to show that this quantity is at least $|l'(0)|/2$ if there is an orthogonal pair and at most $|l'(0)|/(2n^C)$ otherwise. Since $l'(0) \neq 0$, we get the required approximation hardness. □

Polynomial activation function On the other hand, by using the polynomial lifting technique, we can show that changing the activation function can lead to non-trivially faster algorithms:

Theorem 9.30. *Consider a neural net with one hidden layer of size n, with the polynomial activation function $\sigma(x) = x^r$ for some integer $r \geq 2$. Computing the gradients of the empirical loss function for m examples in $\mathbb{R}^d$ can be done in time $O((n+m)d^r)$.*

Note that the running time of the "standard" back-propagation algorithm is $O(dnm)$ for networks with this architecture. Thus our algorithm improves over back-propagation for a non-trivial range of parameters, especially for quadratic activation function when $r = 2$.

We start by defining the network architecture more formally. We consider a neural network computing a function $f : \mathbb{R}^{1\times d} \to \mathbb{R}$ defined as $f(x) := S(xA)\alpha$, where

- $x \in \mathbb{R}^{1\times d}$ is an input row vector of dimensionality d.
- $A \in \mathbb{R}^{d\times m}$ is a matrix with j-th column specifying weights of edges connecting the input units with the j-th hidden unit.
- $S : \mathbb{R} \to \mathbb{R}$ is a non-linearity that is applied entry-wise. $S(x) = x^r$ for some constant integer $r \geq 2$ for the rest of the section.

- $\alpha \in \mathbb{R}^m$ is column vector with α_j specifying the weight of the edge that connects the j-th hidden unit with the output linear unit.

Let $X \in \mathbb{R}^{n \times d}$ be the matrix specifying n inputs vectors. The i-th row of X specifies the i-th input vector. Let $z := f(X) \in \mathbb{R}^n$ be the output of function f when evaluated on the input matrix X. Let $l : \mathbb{R}^n \to \mathbb{R}$ be the total loss function defined as $l(z) := \sum_{i=1}^n l_i(z_i)$ for some functions $l_i : \mathbb{R} \to \mathbb{R}$.

Let

$$\frac{\partial l}{\partial \alpha} := \left(\frac{\partial l}{\partial \alpha_1}, \ldots, \frac{\partial l}{\partial \alpha_m} \right)^{\mathsf{T}} \in \mathbb{R}^m$$

be the vector of gradients for weights $\alpha_1, \ldots, \alpha_m$. Let $\frac{\partial l}{\partial A} \in \mathbb{R}^{d \times m}$ be the matrix that specifies gradient of l with respect to entries $A_{k,j}$. That is,

$$\left(\frac{\partial l}{\partial A} \right)_{k,j} := \frac{\partial l}{\partial A_{k,j}}$$

for $k = 1, \ldots, d$ and $j = 1, \ldots, m$.

Theorem 9.31. *We can evaluate $\frac{\partial l}{\partial \alpha}$ and $\frac{\partial l}{\partial A}$ in $O((n+m)d^r)$.*

Proof. Let $l'(z) := \left(\frac{\partial l_1}{\partial z_1}, \ldots, \frac{\partial l_n}{\partial z_n} \right) \in \mathbb{R}^n$ denote the vector that collects all $\frac{\partial l_i}{\partial z_i}$.

We note that

$$\begin{aligned} \frac{\partial l_i}{\partial \alpha_j} &= \frac{\partial l_i}{\partial z_i} \cdot (\text{output of the } j\text{-th hidden unit on the } i\text{-th input vector}) \\ &= l_i'(z_i) \cdot (X^{(r)} A^{(r)})_{i,j}. \end{aligned}$$

This gives

$$\begin{aligned} \frac{\partial l}{\partial \alpha} &= \left(X^{(r)} A^{(r)} \right)^{\mathsf{T}} l'(z) \\ &= \left(A^{(r)} \right)^{\mathsf{T}} \left(\left(X^{(r)} \right)^{\mathsf{T}} l'(z) \right). \end{aligned}$$

The last expression can be evaluated in the required $O((n+m)d^r) = (n+m)^{1+o(1)}$ time.

We note that

$$\begin{aligned} \frac{\partial l}{\partial A_{k,j}} &= \sum_{i=1}^n \frac{\partial l_i}{\partial A_{k,j}} \\ &= \sum_{i=1}^n X_{i,k} \cdot r \cdot (\text{input to the } j\text{-th hidden unit})^{r-1} \cdot \alpha_j \cdot l_i'(z_i). \end{aligned}$$

For two matrices A and B of equal size let $A \circ B$ be the entry-wise product. We define the column vector $v_k \in \mathbb{R}^n$: $(v_k)_i = X_{i,k} \cdot r \cdot l_i'(z_i)$ for $k = 1, \ldots, d$. Then the k-th row of $\frac{\partial l}{\partial A}$ is equal to $(v_k^{\mathsf{T}} X^{(r-1)} A^{(r-1)}) \circ \alpha^{\mathsf{T}}$. We observe that we can compute

$$(v_k^{\mathsf{T}} X^{(r-1)} A^{(r-1)}) \circ \alpha^{\mathsf{T}} = ((v_k^{\mathsf{T}} X^{(r-1)}) A^{(r-1)}) \circ \alpha^{\mathsf{T}}$$

in $O((n+m)d^{r-1})$ time. Since we have to do that for every $k = 1, \dots, d$, the stated runtime follows. □

Part III

Nearest Neighbor Search

Chapter 10

Unifying Theory And Experiment In Locality-Sensitive Hashing

10.1 Introduction

Nearest neighbor search is a key algorithmic problem with applications in several fields including computer vision, information retrieval, and machine learning [206]. Given a set of n points $P \subset \mathbb{R}^d$, the goal is to build a data structure that answers nearest neighbor queries efficiently: for a given query point $q \in \mathbb{R}^d$, find the point $p \in P$ that is closest to q under an appropriately chosen distance metric. The main algorithmic design goals are usually a fast query time, a small memory footprint, and – in the approximate setting – a good quality of the returned solution.

There is a wide range of algorithms for nearest neighbor search based on techniques such as space partitioning with indexing, as well as dimension reduction or sketching [198]. A popular method for point sets in high-dimensional spaces is Locality-Sensitive Hashing (LSH) [70, 112]. LSH offers a *provably* sub-linear query time and sub-quadratic space complexity. Moreover, it has been shown to achieve good empirical performance in a variety of applications [206].

LSH relies on the notion of *locality-sensitive hash functions*. Intuitively, a hash function is locality-sensitive if its probability of collision is higher for "nearby" points than for points that are "far apart". More formally, two points are nearby if their distance is at most r_1, and they are far apart if their distance is at least $r_2 = c \cdot r_1$, where $c > 1$ quantifies the gap between "near" and "far". The quality of a hash function is then characterized by two key parameters:

- p_1, the collision probability for nearby points, and
- p_2, the collision probability for points that are far apart.

The gap between p_1 and p_2 determines how "sensitive" the hash function is to changes in distance. This property is captured by the parameter

$$\rho = \frac{\log 1/p_1}{\log 1/p_2} \; .$$

As we will see later, the parameter ρ is usually expressed as a function of the distance gap c. The problem of designing good locality-sensitive hash functions and LSH-based efficient nearest neighbor search algorithms has attracted significant attention over the past twenty years.

In this thesis, we focus on LSH for the Euclidean distance *on the unit sphere*, which is an important special case for several reasons. First, the spherical case is relevant in practice. The Euclidean distance on a sphere corresponds to the *angular distance* or *cosine similarity*, which are commonly used in applications such as comparing image feature vectors [134], speaker representations [202], and tf-idf data sets [217]. On the theoretical side, Andoni and Razenshteyn [19] also show a reduction from Nearest Neighbor Search in the *entire* Euclidean space to the spherical case. These connections lead to a natural question: what are good LSH families for the Euclidean distance on the unit sphere?

The recent work of [18, 19] gives the best known theoretical guarantees for LSH-based nearest neighbor search w.r.t. this distance measure. Specifically, their algorithm has a query time of $O(n^\rho)$ and space complexity of $O(n^{1+\rho})$ for

$$\rho = \frac{1}{2c^2 - 1} \; .$$

This running time is known to be essentially optimal for a large class of algorithms [20, 93]. As a concrete example, consider the distance gap $c = 2$. Then their algorithm achieves a query time of $n^{1/7+o(1)}$.

At the heart of their algorithm is an LSH scheme called *Spherical LSH*, which works for unit vectors. Its key property is that it can distinguish between distances $r_1 = \sqrt{2}/c$ and $r_2 = \sqrt{2}$ with probabilities yielding $\rho = \frac{1}{2c^2-1}$ (the formula for the full range of distances is more complex and given in Section 10.2). Unfortunately, their approach is not applicable in practice as it is based on rather complex hash functions that are very time consuming to evaluate. For instance, simply evaluating a single hash function from [19] can take more time than a linear scan over 10^6 points. Since an LSH data structure contains many individual hash functions, using their scheme would be slower than a simple linear scan over all points in P unless the number of points n is extremely large.

On the practical side, the hyperplane LSH introduced in the influential work of Charikar [70] has worse theoretical guarantees, but works well in practice. Since the hyperplane LSH can be implemented very efficiently, it is the standard hash function in practical LSH-based nearest neighbor algorithms[1] and the resulting implementations has been shown to improve over a linear scan on real data by multiple orders of magnitude [158, 217].

The aforementioned discrepancy between the theory and practice of LSH raises an important question: is there a locality-sensitive hash function with *optimal* guarantees that also improves over the hyperplane LSH in practice?

In this thesis, we show that there is a family of locality-sensitive hash functions that achieves both objectives. Specifically, the hash functions match the theoretical guarantee of Spherical LSH from [19] and, when combined with additional techniques, give better experimental results than the hyperplane LSH. More specifically, our contributions are:

[1] Note that if the data points are *binary*, more efficient LSH schemes exist [210, 211]. However, in this thesis we consider algorithms for general (non-binary) vectors.

Theoretical guarantees for the cross-polytope LSH. We show that a hash function based on randomly rotated cross-polytopes (i.e., unit balls of the ℓ_1-norm) achieves the same parameter ρ as the Spherical LSH scheme in [19], assuming data points are unit vectors. While the cross-polytope LSH family has been proposed by researchers before [97, 219] we give the first theoretical analysis of its performance.

Fine-grained lower bound for cosine similarity LSH. To highlight the difficulty of obtaining optimal *and* practical LSH schemes, we prove the first *non-asymptotic* lower bound on the trade-off between the collision probabilities p_1 and p_2. So far, the optimal LSH upper bound $\rho = \frac{1}{2c^2-1}$ (from [18, 19] and cross-polytope from here) attain this bound only in the limit, as $p_1, p_2 \to 0$. Very small p_1 and p_2 are undesirable since the hash evaluation time is often proportional to $1/p_2$. Our lower bound proves this is unavoidable: if we require p_2 to be large, ρ has to be suboptimal.

This result has two important implications for designing practical hash functions. First, it shows that the trade-offs achieved by the cross-polytope LSH and the scheme of [18, 19] are essentially optimal. Second, the lower bound guides design of future LSH functions: if one is to significantly improve upon the cross-polytope LSH, one has to design a hash function that is computed more efficiently than by explicitly enumerating its range (see Section 10.3 for a more detailed discussion).

Multiprobe scheme for the cross-polytope LSH. The space complexity of an LSH data structure is sub-*quadratic*, but even this is often too large (i.e., strongly super-*linear* in the number of points), and several methods have been proposed to address this issue. Empirically, the most efficient scheme is multiprobe LSH [158], which leads to a significantly reduced memory footprint for the hyperplane LSH. In order to make the cross-polytope LSH competitive in practice with the multiprobe hyperplane LSH, we propose a novel multiprobe scheme for the cross-polytope LSH.

We complement these contributions with an experimental evaluation on both real and synthetic data (SIFT vectors, tf-idf data, and a random point set). In order to make the cross-polytope LSH practical, we combine it with fast pseudo-random rotations [12] via the Fast Hadamard Transform, and feature hashing [233] to exploit sparsity of data. Our results show that for data sets with around 10^5 to 10^8 points, our multiprobe variant of the cross-polytope LSH is up to $10\times$ faster than an efficient implementation of the hyperplane LSH, and up to $700\times$ faster than a linear scan. To the best of our knowledge, our combination of techniques provides the first "exponent-optimal" algorithm that empirically improves over the hyperplane LSH in terms of query time for an *exact* nearest neighbor search.

10.1.1 Related work

The cross-polytope LSH functions were originally proposed by Terasawa and Tanaka [219]. However, their analysis was mostly experimental. Specifically, the probabilities p_1 and p_2 of the proposed LSH functions were estimated empirically using the Monte Carlo method. Similar hash functions were later proposed by Eshghi and Rajaram [97]. They also use the DFT to speed-up the matrix-vector multiplication. Both of the aforementioned papers consider only a *single-probe* algorithm.

There are several works that show lower bounds on the quality of LSH hash functions [20, 93, 162, 176]. However, they provide only a lower bound on the ρ parameter for asymptotic values of p_1 and p_2, as opposed to an actual trade-off between these two quantities. In this thesis we provide such a trade-off, with implications as outlined in the introduction.

10.1.2 Preliminaries

We use $\|.\|$ to denote the Euclidean (a.k.a. ℓ_2) norm on $\mathbb{R}^d$. We also use S^{d-1} to denote the unit sphere in $\mathbb{R}^d$ centered in the origin. The Gaussian distribution with mean zero and variance of one is denoted by $N(0,1)$. Let μ be a normalized Haar measure on S^{d-1} (that is, $\mu(S^{d-1}) = 1$). Note that μ it corresponds to the uniform distribution over S^{d-1}. We also let $u \sim S^{d-1}$ be a point sampled from S^{d-1} uniformly at random. For $\eta \in \mathbb{R}$ we denote

$$\Phi_c(\eta) = \Pr_{X \sim N(0,1)}[X \geq \eta] = \frac{1}{\sqrt{2\pi}} \int_{\eta}^{\infty} e^{-t^2/2}\, dt.$$

We will be interested in the Near Neighbor Search on the sphere S^{d-1} with respect to the Euclidean distance. Note that the angular distance can be expressed via the Euclidean distance between normalized vectors, so our results apply to the angular distance as well.

Definition 10.1. *Given an n-point dataset $P \subset S^{d-1}$ on the sphere, the goal of the (c, r)-* Approximate Near Neighbor problem (ANN) *is to build a data structure that, given a query $q \in S^{d-1}$ with the promise that there exists a datapoint $p \in P$ with $\|p - q\| \leq r$, reports a datapoint $p' \in P$ within distance cr from q.*

Definition 10.2. *We say that a hash family $\mathcal{H}$ on the sphere S^{d-1} is (r_1, r_2, p_1, p_2)*-sensitive*, if for every $p, q \in S^{d-1}$ one has $\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \geq p_1$ if $\|x - y\| \leq r_1$, and $\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] \leq p_2$ if $\|x - y\| \geq r_2$,*

It is known [112] that an *efficient* (r, cr, p_1, p_2)-sensitive hash family implies a data structure for (c, r)-ANN with space $O(n^{1+\rho}/p_1 + dn)$ and query time $O(d \cdot n^{\rho}/p_1)$, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.

10.2 Cross-polytope LSH

In this section, we describe the cross-polytope LSH, analyze it, and show how to make it practical. First, we recall the definition of the cross-polytope LSH [219]: Consider the following hash family $\mathcal{H}$ for points on a unit sphere $S^{d-1} \subset \mathbb{R}^d$. Let $A \in \mathbb{R}^{d \times d}$ be a random matrix with i.i.d. Gaussian entries ("a random rotation"). To hash a point $x \in S^{d-1}$, we compute $y = Ax/\|Ax\| \in S^{d-1}$ and then find the point closest to y from $\{\pm e_i\}_{1 \leq i \leq d}$, where e_i is the i-th standard basis vector of $\mathbb{R}^d$. We use the closest neighbor as a hash of x.

The following theorem bounds the collision probability for two points under the above family $\mathcal{H}$.

Theorem 10.3. *Suppose that $p, q \in S^{d-1}$ are such that $\|p - q\| = \tau$, where $0 < \tau < 2$. Then,*

$$\ln \frac{1}{\Pr_{h \sim \mathcal{H}}[h(p) = h(q)]} = \frac{\tau^2}{4 - \tau^2} \cdot \ln d + O_{\tau}(\ln \ln d)\ .$$

Before we show how to prove this theorem, we briefly describe its implications. Theorem 10.3 shows that the cross-polytope LSH achieves essentially the same bounds on the collision probabilities as the (theoretically) optimal LSH for the sphere from [19] (see Section "Spherical LSH" there). In particular, substituting the bounds from Theorem 10.3 for the cross-polytope LSH into the standard reduction from Near Neighbor Search to LSH [112], we obtain the following data structure with sub-quadratic space and sublinear query time for Near Neighbor Search on a sphere.

Corollary 10.4. *The (c, r)-ANN on a unit sphere S^{d-1} can be solved in space $O(n^{1+\rho} + dn)$ and query time $O(d \cdot n^{\rho})$, where $\rho = \frac{1}{c^2} \cdot \frac{4-c^2r^2}{4-r^2} + o(1)$.*

We now outline the proof of Theorem 10.3. For the full proof, see Section 10.5.2.

Due to the spherical symmetry of Gaussians, we can assume that $p = e_1$ and $q = \alpha e_1 + \beta e_2$, where α, β are such that $\alpha^2 + \beta^2 = 1$ and $(\alpha - 1)^2 + \beta^2 = \tau^2$. Then, we expand the collision probability:

$$\begin{aligned}\Pr_{h\sim\mathcal{H}}[h(p) = h(q)] &= 2d \cdot \Pr_{h\sim\mathcal{H}}[h(p) = h(q) = e_1] \\ &= 2d \cdot \Pr_{u,v\sim N(0,1)^d}[\forall i \ \ |u_i| \le u_1 \text{ and } |\alpha u_i + \beta v_i| \le \alpha u_1 + \beta v_1] \\ &= 2d \cdot \mathop{\mathrm{E}}_{X_1,Y_1}\left[\Pr_{X_2,Y_2}\Big[|X_2| \le X_1 \text{ and } |\alpha X_2 + \beta Y_2| \le \alpha X_1 + \beta Y_1\Big]^{d-1}\right], \end{aligned} \tag{10.1}$$

where $X_1, Y_1, X_2, Y_2 \sim N(0, 1)$. Indeed, the first step is due to the spherical symmetry of the hash family, the second step follows from the above discussion about replacing a random orthogonal matrix with a Gaussian one and that one can assume w.l.o.g. that $p = e_1$ and $q = \alpha e_1 + \beta e_2$; the last step is due to the independence of the entries of u and v.

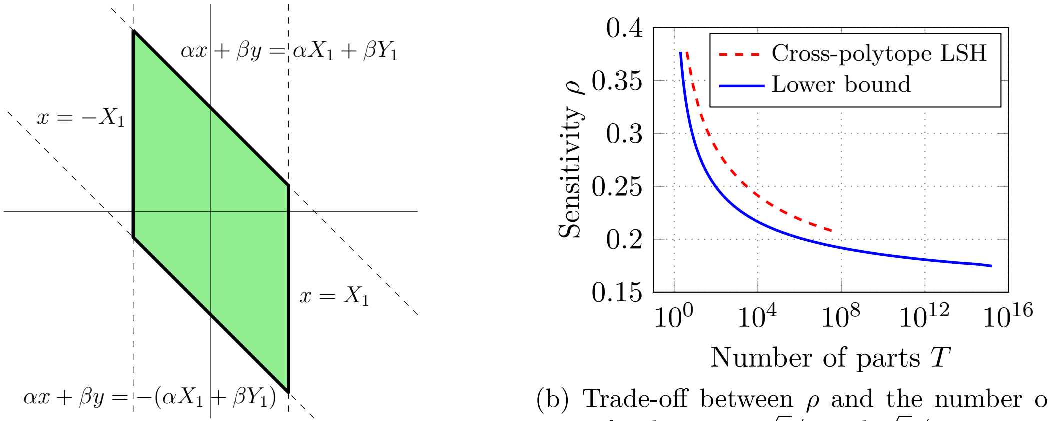
Thus, proving Theorem 10.3 reduces to estimating the right-hand side of (10.1). Note that the probability $\Pr[|X_2| \le X_1 \text{ and } |\alpha X_2 + \beta Y_2| \le \alpha X_1 + \beta Y_1]$ is equal to the Gaussian area of the planar set S_{X_1,Y_1} shown in Figure 10-1a. The latter is *heuristically* equal to $1 - e^{-\Delta^2/2}$, where Δ is the distance from the origin to the complement of S_{X_1,Y_1}, which is easy to compute (see Section 10.5.1 for the precise statement of this argument). Using this estimate, we compute (10.1) by taking the outer expectation.

10.2.1 Making the cross-polytope LSH practical

As described above, the cross-polytope LSH is not quite practical. The main bottleneck is sampling, storing, and applying a random rotation. In particular, to multiply a random Gaussian matrix with a vector, we need time proportional to d^2, which is infeasible for large d.

Pseudo-random rotations. To rectify this issue, we instead use *pseudo-random rotations*. Instead of multiplying an input vector x by a random Gaussian matrix, we apply the following linear transformation: $x \mapsto HD_3HD_2HD_1x$, where H is the Hadamard transform, and D_i for $i \in \{1, 2, 3\}$ is a random diagonal ±1-matrix. Clearly, this is an orthogonal transformation, which one can store in space $O(d)$ and evaluate in time $O(d \log d)$ using

Figure (10-1)



(a) The set appearing in the analysis of the cross-polytope LSH: $S_{X_1Y_1} = \{|x| \le X_1 \text{ and } |\alpha x + \beta y| \le \alpha X_1 + \beta Y_1\}$.

(b) Trade-off between ρ and the number of parts for distances $\sqrt{2}/2$ and $\sqrt{2}$ (approximation $c = 2$); both bounds tend to $1/7$ (see discussion in Section 10.3).

the Fast Hadamard Transform. This is similar to pseudo-random rotations used in the context of LSH [81], dimensionality reduction [12], or compressed sensing [13]. While we are currently not aware how to prove rigorously that such pseudo-random rotations perform as well as the fully random ones, empirical evaluations show that three applications of HD_i are exactly equivalent to applying a true random rotation (when d tends to infinity). We note that only *two* applications of HD_i are not sufficient.

Feature hashing. While we can apply a pseudo-random rotation in time $O(d \log d)$, even this can be too slow. E.g., consider an input vector x that is *sparse*: the number of non-zero entries of x is s much smaller than d. In this case, we can evaluate the hyperplane LSH from [70] in time $O(s)$, while computing the cross-polytope LSH (even with pseudo-random rotations) still takes time $O(d \log d)$. To speed-up the cross-polytope LSH for sparse vectors, we apply feature hashing [233]: before performing a pseudo-random rotation, we reduce the dimension from d to $d' \ll d$ by applying a linear map $x \mapsto Sx$, where S is a random sparse $d' \times d$ matrix, whose columns have *one* non-zero ± 1 entry sampled uniformly. This way, the evaluation time becomes $O(s + d' \log d')$. [1]

"Partial" cross-polytope LSH. In the above discussion, we defined the cross-polytope LSH as a hash family that returns the closest neighbor among $\{\pm e_i\}_{1 \le i \le d}$ as a hash (after a (pseudo-)random rotation). In principle, we do not have to consider all d basis vectors when computing the closest neighbor. By restricting the hash to $d' \le d$ basis vectors instead, Theorem 10.3 still holds for the new hash family (with d replaced by d') since the analysis is essentially dimension-free. This slight generalization of the cross-polytope LSH turns out to be useful for experiments (see Section 10.4). Note that the case $d' = 1$ corresponds to the hyperplane LSH.

[1] Note that one can apply Lemma 2 from the arXiv version of [233] to claim that—after such a dimension reduction—the distance between *any* two points remains sufficiently concentrated for the bounds from Theorem 10.3 to still hold (with d replaced by d').

10.2.2 Multiprobe LSH for the cross-polytope LSH

We now describe our multiprobe scheme for the cross-polytope LSH, which is a method for reducing the number of independent hash tables in an LSH data structure. Given a query point q, a "standard" LSH data structure considers only a *single* cell in each of the L hash tables (the cell is given by the hash value $h_i(q)$ for $i \in [L]$). In multiprobe LSH, we consider candidates from *multiple* cells in each table [158]. The rationale is the following: points p that are close to q but fail to collide with q under hash function h_i are still likely to hash to a value that is close to $h_i(q)$. By probing multiple hash locations close to $h_i(q)$ in the same table, multiprobe LSH achieves a given probability of success with a smaller number of hash tables than "standard" LSH. Multiprobe LSH has been shown to perform well in practice [158, 214].

The main ingredient in multiprobe LSH is a probing scheme for generating and ranking possible modifications of the hash value $h_i(q)$. The probing scheme should be computationally efficient and ensure that more likely hash locations are probed first. For a single cross-polytope hash, the order of alternative hash values is straightforward: let x be the (pseudo-)randomly rotated version of query point q. Recall that the "main" hash value is $h_i(q) = \arg\max_{j\in[d]} |x_j|$.[1] Then it is easy to see that the second highest probability of collision is achieved for the hash value corresponding to the coordinate with the second largest absolute value, etc. Therefore, we consider the indices $i \in [d]$ sorted by their absolute value as our probing sequence or "ranking" for a single cross-polytope.

The remaining question is how to combine multiple cross-polytope rankings when we have more than one hash function. As in the analysis of the cross-polytope LSH (see Section 10.2, we consider two points $q = e_1$ and $p = \alpha e_1 + \beta e_2$ at distance R. Let $A^{(i)}$ be the i.i.d. Gaussian matrix of hash function h_i, and let $x^{(i)} = A^{(i)} e_1$ be the randomly rotated version of point q. Given $x^{(i)}$, we are interested in the probability of p hashing to a certain combination of the individual cross-polytope rankings. More formally, let $r^{(i)}_{v_i}$ be the index of the v_i-th largest element of $|x^{(i)}|$, where $v \in [d]^k$ specifies the alternative probing location. Then we would like to compute

$$\begin{aligned}
&\Pr_{A^{(1)},\ldots,A^{(k)}}\left[h_i(p) = r^{(i)}_{v_i} \text{ for all } i \in [k] \mid A^{(i)} q = x^{(i)}\right] \\
&\qquad = \prod_{i=1}^{k} \Pr_{A^{(i)}}\left[\arg\max_{j\in[d]} \left|(\alpha \cdot A^{(i)} e_1 + \beta \cdot A^{(i)} e_2)_j\right| = r^{(i)}_{v_i} \;\middle|\; A^{(i)} e_1 = x^{(i)}\right] .
\end{aligned}$$

If we knew this probability for all $v \in [d]^k$, we could sort the probing locations by their probability. We now show how to approximate this probability efficiently for a single value of i (and hence drop the superscripts to simplify notation). WLOG, we permute the rows of A so that $r_v = v$ and get

$$\Pr_{A}\left[\arg\max_{j\in[d]} \left|(\alpha x + \beta \cdot A e_2)_j\right| = v \;\middle|\; A e_1 = x\right] \;=\; \Pr_{y\sim N(0,I_d)}\left[\arg\max_{j\in[d]} \left|(x + \frac{\beta}{\alpha}\cdot y)_j\right| = v\right] .$$

The RHS is the Gaussian measure of the set $S = \{y \in \mathbb{R}^d \mid \arg\max_{j\in[d]} |(x + \frac{\beta}{\alpha} y)_j| = v\}$.

[1] In order to simplify notation, we consider a slightly modified version of the cross-polytope LSH that maps both the standard basis vector $+e_j$ and its opposite $-e_j$ to the same hash value. It is easy to extend the multiprobe scheme defined here to the "full" cross-polytope LSH from Section 10.2.

Similar to the analysis of the cross-polytope LSH, we approximate the measure of S by its distance to the origin. Then the probability of probing location v is proportional to $\exp(-\|y_{x,v}\|^2)$, where $y_{x,v}$ is the shortest vector y such that $\arg\max_j |x+y|_j = v$. Note that the factor β/α becomes a proportionality constant, and hence the probing scheme does not require to know the distance R. For computational performance and simplicity, we make a further approximation and use $y_{x,v} = (\max_i |x_i| - |x_v|) \cdot e_v$, i.e., we only consider modifying a single coordinate to reach the set S.

Once we have estimated the probabilities for each $v_i \in [d]$, we incrementally construct the probing sequence using a binary heap, similar to the approach in [158]. For a probing sequence of length m, the resulting algorithm has running time $O(L \cdot d \log d + m \log m)$. In our experiments, we found that the $O(L \cdot d \log d)$ time taken to sort the probing candidates v_i dominated the running time of the hash function evaluation. In order to circumvent this issue, we use an incremental sorting approach that only sorts the relevant parts of each cross-polytope and gives a running time of $O(L \cdot d + m \log m)$.

10.3 Lower bound

Let $\mathcal{H}$ be a hash family on S^{d-1}. For $0 < r_1 < r_2 < 2$ we would like to understand the trade-off between p_1 and p_2, where p_1 is the *smallest* probability of collision under $\mathcal{H}$ for points at distance *at most* r_1 and p_2 is the *largest* probability of collision for points at distance *at least* r_2. We focus on the case $r_2 \approx \sqrt{2}$ because setting r_2 to $\sqrt{2} - o(1)$ (as d tends to infinity) allows us to replace p_2 with the following quantity that is somewhat easier to handle:

$$p_2^* = \Pr_{\substack{h \sim \mathcal{H} \\ u,v \sim S^{d-1}}} [h(u) = h(v)].$$

This quantity is at most $p_2 + o(1)$, since the distance between two random points on a unit sphere S^{d-1} is tightly concentrated around $\sqrt{2}$. So for a hash family $\mathcal{H}$ on a unit sphere S^{d-1}, we would like to understand the upper bound on p_1 in terms of p_2^* and $0 < r_1 < \sqrt{2}$.

For $0 \leq \tau \leq \sqrt{2}$ and $\eta \in \mathbb{R}$, we define

$$\Lambda(\tau, \eta) = \Pr_{X,Y \sim N(0,1)} \left[X \geq \eta \text{ and } \left(1 - \frac{\tau^2}{2}\right) \cdot X + \sqrt{\tau^2 - \frac{\tau^4}{4}} \cdot Y \geq \eta \right] \Big/ \Pr_{X \sim N(0,1)} [X \geq \eta] \,.$$

We are now ready to formulate the main result of this section.

Theorem 10.5. *Let $\mathcal{H}$ be a hash family on S^{d-1} such that every function in $\mathcal{H}$ partitions the sphere into at most T parts of measure at most $1/2$. Then we have $p_1 \leq \Lambda(r_1, \eta) + o(1)$, where $\eta \in \mathbb{R}$ is such that $\Phi_c(\eta) = p_2^*$ and $o(1)$ is a quantity that depends on T and r_1 and tends to 0 as d tends to infinity.*

The idea of the proof is first to reason about one part of the partition using the isoperimetric inequality from [98], and then to apply a certain averaging argument by proving concavity of a function related to Λ using a delicate analytic argument. For the full proof, see Section 10.5.3.

We note that the above requirement of all parts induced by $\mathcal{H}$ having measure at most $1/2$ is only a technicality. We conjecture that Theorem 10.5 holds without this restriction. In any case, as we will see below, in the interesting range of parameters this restriction is essentially irrelevant.

One can observe that if every hash function in $\mathcal{H}$ partitions the sphere into at most T parts, then $p_2^* \geq \frac{1}{T}$ (indeed, p_2^* is precisely the average sum of squares of measures of the parts). This observation, combined with Theorem 10.5, leads to the following interesting consequence. Specifically, we can numerically estimate Λ in order to give a lower bound on $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$ for any hash family $\mathcal{H}$ in which every function induces at most T parts of measure at most $1/2$. See Figure 10-1b, where we plot this lower bound for $r_1 = \sqrt{2}/2$,[1] together with an upper bound that is given by the cross-polytope LSH[2] (for which we use numerical estimates for (10.1)). We can make several conclusions from this plot. First, the cross-polytope LSH gives an almost optimal trade-off between ρ and T. Given that the evaluation time for the cross-polytope LSH is $O(T \log T)$ (if one uses pseudo-random rotations), we conclude that in order to improve upon the cross-polytope LSH substantially in practice, one should design an LSH family with ρ being close to optimal and evaluation time that is *sublinear in T*. We note that none of the known LSH families for a sphere has been shown to have this property. This direction looks especially interesting since the convergence of ρ to the optimal value (as T tends to infinity) is extremely slow (for instance, according to Figure 10-1b, for $r_1 = \sqrt{2}/2$ and $r_2 \approx \sqrt{2}$ we need more than 10^5 parts to achieve $\rho \leq 0.2$, whereas the optimal ρ is $1/7 \approx 0.143$).

10.4 Experiments

We now show that the cross-polytope LSH, combined with our multiprobe extension, leads to an algorithm that is also efficient in practice and improves over the hyperplane LSH on several data sets. The focus of our experiments is the query time for an *exact* nearest neighbor search. Since hyperplane LSH has been compared to other nearest-neighbor algorithms before [202], we limit our attention to the relative speed-up compared with hyperplane hashing.

We evaluate the two hashing schemes on three types of data sets. We use a synthetic data set of randomly generated points because this allows us to vary a single problem parameter while keeping the remaining parameters constant. We also investigate the performance of our algorithm on real data: two tf-idf data sets [151] and a set of SIFT feature vectors [134]. We have chosen these data sets in order to illustrate when the cross-polytope LSH gives large improvements over the hyperplane LSH, and when the improvements are more modest. We present our experiments at a high level here and refer the reader to subsection 10.4.1 for a more detailed description of the data sets and our experimental setup (implementation details, CPU, etc.).

In all experiments, we set the algorithm parameters so that the empirical probability of successfully finding the exact nearest neighbor is at least 0.9. Moreover, we set the number of LSH tables L so that the amount of additional memory occupied by the LSH data structure

[1] The situation is qualitatively similar for other values of r_1.

[2] More specifically, for the "partial" version from Section 10.2.1, since T should be constant, while d grows

is comparable to the amount of memory necessary for storing the data set. We believe that this is the most interesting regime because significant memory overheads are often impossible for large data sets. In order to determine the parameters that are not fixed by the above constraints, we perform a grid search over the remaining parameter space and report the best combination of parameters. For the cross-polytope hash, we consider "partial" cross-polytopes in the last of the k hash functions in order to get a smooth trade-off between the various parameters (see Section 10.2.1).

Multiprobe experiments. In order to demonstrate that the multiprobe scheme is critical for making the cross-polytope LSH competitive with hyperplane hashing, we compare the performance of a "standard" cross-polytope LSH data structure with our multiprobe variant on an instance of the random data set ($n = 2^{20}$, $d = 128$). As can be seen in Table 10.2 (Subsection 10.4.1), the multiprobe variant is about 13× faster in our memory-constrained setting ($L = 10$). Note that in all of the following experiments, the speed-up of the multiprobe cross-polytope LSH compared to the multiprobe hyperplane LSH is less than 11×. Hence without our multiprobe addition, the cross-polytope LSH would be slower than the hyperplane LSH, for which a multiprobe scheme is already known [158].

Experiments on random data. Next, we show that the better time complexity of the cross-polytope LSH already applies for moderate values of n. In particular, we compare the cross-polytope LSH, combined with fast rotations (Section 10.2.1) and our multiprobe scheme, to a multi-probe hyperplane LSH on random data. We keep the dimension $d = 128$ and the distance to the nearest neighbor $R = \sqrt{2}/2$ fixed, and vary the size of the data set from 2^{20} to 2^{28}. The number of hash tables L is set to 10. For 2^{20} points, the cross-polytope LSH is already 3.5× faster than the hyperplane LSH, and for $n = 2^{28}$ the speedup is 10.3× (see Table 10.3 in Subsection 10.4.1). Compared to a linear scan, the speed-up achieved by the cross-polytope LSH ranges from 76× for $n = 2^{20}$ to about 700× for $n = 2^{28}$.

Experiments on real data. On the SIFT data set ($n = 10^6$ and $d = 128$), the cross-polytope LSH achieves a modest speed-up of 1.2× compared to the hyperplane LSH (see Table 10.1). On the other hand, the speed-up is is $3 - 4\times$ on the two tf-idf data sets, which is a significant improvement considering the relatively small size of the NYT data set ($n \approx 300,000$). One important difference between the data sets is that the typical distance to the nearest neighbor is smaller in the SIFT data set, which can make the nearest neighbor problem easier (see Subsection 10.4.1). Since the tf-idf data sets are very high-dimensional but sparse ($d \approx 100,000$), we use the feature hashing approach described in Section 10.2.1 in order to reduce the hashing time of the cross-polytope LSH (the standard hyperplane LSH already runs in time proportional to the sparsity of a vector). We use 1024 and 2048 as feature hashing dimensions for NYT and pubmed, respectively.

10.4.1 Further description of experiments

In order to compare meaningful running time numbers, we have written fast C++ implementations of both the cross-polytope LSH and the hyperplane LSH. This enables a fair comparison since both implementations have been optimized by us to the same degree. In

Data set	Method	Query time (ms)	**Speed-up vs HP**	Best k	Number of candidates	Distances time (ms)
NYT	HP	120 ms		19	57,200	96
NYT	CP	35 ms	**3.4**×	2 (64)	17,900	30
pubmed	HP	857 ms		20	1,481,000	762
pubmed	CP	213 ms	**4.0**×	2 (512)	304,000	168
SIFT	HP	3.7 ms		30	18,600	3.0
SIFT	CP	3.1 ms	**1.2**×	6 (1)	13,400	2.2

Table (10.1): Average running times for a single nearest neighbor query with the hyperplane (HP) and cross-polytope (CP) algorithms on three real data sets. The cross-polytope LSH is faster than the hyperplane LSH on all data sets, with significant speed-ups for the two tf-idf data sets NYT and pubmed. For the cross-polytope LSH, the entries for k include both the number of individual hash functions per table and (in parenthesis) the dimension of the last of the k cross-polytopes.

particular, hyperplane hashing can be implemented efficiently using a matrix-vector multiplication sub-routine for which we use the eigen library (eigen is also used for all other linear algebra operations). For the fast pseudo-random rotation in the cross-polytope LSH, we have written a SIMD-optimized version of the Fast Hadamard Transform (FHT). We compiled our code with g++ 4.9 and the -O3 flag. All experiments except those in Table 10.3 ran on an Intel Core i5-2500 CPU (3.3 - 3.7 GHz, 6 MB cache) with 8 GB of RAM. Since 8 GB of RAM was too small for the larger values of n, we ran the experiments in Table 10.3 on a machine with an Intel Xeon E5-2690 v2 CPU (3.0 GHz, 25 MB cache) and 512 GB of RAM.

In our experiments, we evaluate the performance of the cross-polytope LSH on the following data sets. Figure 10-2 shows the distribution of distances to the nearest neighbor for the four data sets.

random For the random data sets, we generate a set of n points uniformly at random on the unit sphere. In order to generate a query, we pick a random point q' from the data set and generate a point at distance R from q' on the unit sphere. In our experiments, we vary the dimension of the point set between 128 and 1,024. Experiments with the random data set are useful because we can study the impact of various parameters (e.g., the dimension d or the number of points n) while keeping the remaining parameters constant.

pubmed / NYT The pubmed and NYT data sets contain bag-of-words representations of medical publication abstracts and newspaper articles, respectively [151]. We convert this representation into standard tf-idf feature vectors with dimensionality about 100,000. The number of points in the pubmed data set is about 8 million, for NYT it is 300,000. Before setting up the LSH data structures, we set 1000 data points aside as query vectors. When selecting query vectors, we limit our attention to points for which the inner product with the nearest neighbor is between 0.3 and 0.8. We believe that this is the most interesting range since near-duplicates (inner product close to 1) can be identified more efficiently with other methods, and points without a close nearest neighbor (inner product less than

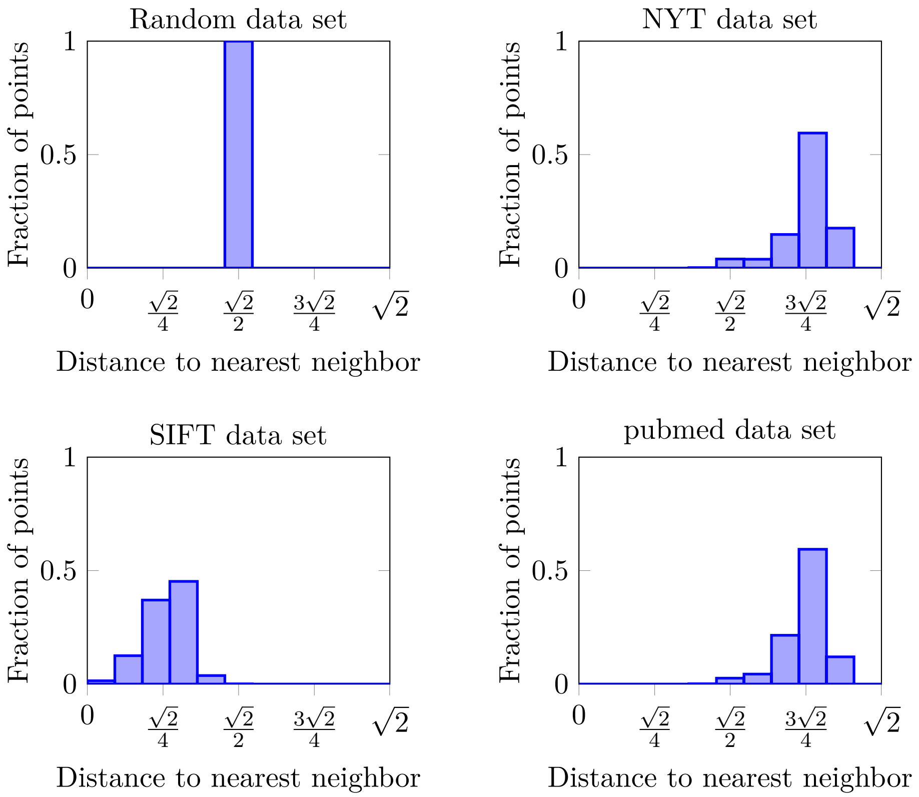


Figure (10-2): Distance to the nearest neighbor for the four data sets used in our experiments. The SIFT data set has the closest nearest neighbors.

0.3) often do not have a semantically meaningful match.

SIFT We use the standard data set of one million SIFT feature vectors from [134], which also contains a set of 10,000 query vectors. The SIFT feature vectors have dimension 128 and (approximately) live on a sphere. We normalize the feature vectors to unit length but keep the original nearest neighbor assignments—this is possible because only a very small fraction of nearest neighbors changes through normalization. We include this data set as an example where the speed-up of the cross-polytope LSH is more modest.

10.5 Proof details

10.5.1 Gaussian measure of a planar set

In this section we formalize the intuition that the standard Gaussian measure of a closed subset $A \subseteq \mathbb{R}^2$ behaves like $e^{-\Delta_A^2/2}$, where Δ_A is the distance from the origin to A, unless A is quite special.

For a closed subset $A \subseteq \mathbb{R}^2$ and $r > 0$ denote $0 \leq \mu_A(r) \leq 1$ the normalized measure of the

Method	k	Extra probes	**Query time (ms)**	Number of candidates	CP hashing time (ms)	Distances time (ms)
Single-probe	1	0	**6.7**	39800	0.01	6.3
Multiprobe	3	896	**0.51**	867	0.22	0.16

Table (10.2): Comparison of "standard" LSH using the cross-polytope (CP) hash vs. our multiprobe variant ($L = 10$ in both cases). On a random data set with $n = 2^{20}$, $d = 128$, and $R = \sqrt{2}/2$, the single-probe scheme requires 13× more time per query. Due to the larger value of k, the multiprobe variant performs fewer distance computations, which leads to a better trade-off between the hash computation time and the time spent on computing distances to candidates from the hash tables.

Data set size n	2^{20}	2^{22}	2^{24}	2^{26}	2^{28}
HP query time (ms)	2.6	7.4	25	63	185
CP query time (ms)	0.75	1.4	3.1	8.8	18
Speed-up	**3.5**×	**5.3**×	**8.1**×	**7.2**×	**10.3**×
k for CP	3 (16)	3 (64)	3 (128)	4 (2)	4 (64)

Table (10.3): Average running times for a single nearest neighbor query with the hyperplane (HP) and cross-polytope (CP) algorithms on a random data set with $d = 128$ and $R = \sqrt{2}/2$. The cross-polytope LSH is up to 10× faster than the hyperplane LSH. The last row of the table indicates the optimal choice of k for the cross-polytope LSH and (in parenthesis) the dimension of the last of the k cross-polytopes; all other cross-polytopes have full dimension 128. Note that the speed-up ratio is not monotonically increasing because the cross-polytope LSH performs better for values of n where the optimal setting of k uses a last cross-polytope with high dimension.

intersection $A \cap rS^1$ (A with the circle centered in the origin and of radius r):

$$\mu_A(r) := \frac{\mu(A \cap rS^1)}{2\pi r};$$

here μ is the standard one-dimensional Lebesgue measure (see Figure 10-3a). Denote $\Delta_A := \inf\{r > 0 : \mu_A(r) > 0\}$ the (essential) distance from the origin to A. Let $\mathcal{G}(A)$ be the standard Gaussian measure of A.

Lemma 10.6. *Suppose that $A \subseteq \mathbb{R}^2$ is a closed set such that $\mu_A(r)$ is non-decreasing. Then,*

$$\sup_{r>0}\Big(\mu_A(r) \cdot e^{-r^2/2}\Big) \leq \mathcal{G}(A) \leq e^{-\Delta_A^2/2}.$$

Proof. For the upper bound, we note that

$$\mathcal{G}(A) = \int_0^\infty \mu_A(r) \cdot re^{-r^2/2}\,dr \leq \int_{\Delta_A}^\infty re^{-r^2/2}\,dr = e^{-\Delta_A^2/2}.$$

For the lower bound, we similarly have, for every $r^* > 0$,

$$\mathcal{G}(A) = \int_0^\infty \mu_A(r) \cdot re^{-r^2/2}\, dr \ge \mu_A(r^*) \cdot \int_{r^*}^\infty re^{-r^2/2}\, dr = \mu_A(r^*)e^{-(r^*)^2/2},$$

where we use that $\mu_A(r^*)$ is non-decreasing. □

Now we derive two corollaries of Lemma 10.6.

Lemma 10.7. *Let $K \subseteq \mathbb{R}^2$ be the complement of an open convex subset of the plane that is symmetric around the origin. Then, for every $0 < \varepsilon < 1/3$,*

$$\Omega\Big(\varepsilon^{1/2} \cdot e^{-(1+\varepsilon)\cdot\Delta_K^2/2}\Big) \le \mathcal{G}(K) \le e^{-\Delta_K^2/2}.$$

Proof. This follows from Lemma 10.6: indeed, due to the convexity of the complement of K, $\mu_K(r)$ is non-decreasing. It is easy to check that

$$\mu_K\Big((1+\varepsilon)\Delta_K\Big) = \Omega\Big(\varepsilon^{1/2}\Big),$$

again, due to the convexity (see Figure 10-3b). Thus, the required bounds follow. □

Lemma 10.8. *Let $K \subseteq \mathbb{R}^2$ be an intersection of two closed half-planes such that:*

- *K does not contain a line;*
- *the "corner" of K is the closest point of K to the origin;*
- *the angle between half-planes equals to $0 < \alpha < \pi$.*

Then, for every $0 < \varepsilon < 1/2$,

$$\Omega_\alpha\Big(\varepsilon \cdot e^{-(1+\varepsilon)\cdot\Delta_K^2}\Big) \le \mathcal{G}(K) \le e^{-\Delta_K^2/2}.$$

Proof. This, again, follows from Lemma 10.6. The second condition implies that $\mu_K(r)$ is non-decreasing, and an easy computation shows that

$$\mu_K((1+\varepsilon)\Delta_K) \ge \Omega_\alpha(\varepsilon)$$

(see Figure 10-3c). □

10.5.2 Proof of Theorem 10.3

In this section, we complete the proof of Theorem 10.3, following the outline from Section 10.2. Our starting point is the collision probability bound from Eqn. (10.1).

For $u, v \in \mathbb{R}$ with $u \ge 0$ and $\alpha u + \beta v \ge 0$ define,

$$\sigma(u, v) = \Pr_{X_2, Y_2 \sim N(0,1)}[|X_2| \le u \text{ and } |\alpha X_2 + \beta Y_2| \le \alpha u + \beta v].$$

Then, the right-hand side of (10.1) is equal to

$$2d \cdot \mathop{\mathrm{E}}_{X_1, Y_1 \sim N(0,1)} [\sigma(X_1, Y_1)^{d-1}].$$

Let us define

$$\Delta(u, v) = \min\{u, \alpha u + \beta v\}.$$

Lemma 10.9. *For every* $0 < \varepsilon < 1/3$*,*

$$1 - e^{-\Delta(u,v)^2/2} \leq \sigma(u, v) \leq 1 - \Omega\left(\varepsilon^{1/2} \cdot e^{-(1+\varepsilon)\Delta(u,v)^2/2}\right).$$

Proof. This is a combination of Lemma 10.7 together with the following obvious observation: the distance from the origin to the set $\{(x, y) : |x| \geq u \text{ or } |\alpha x + \beta y| \geq \alpha u + \beta v\}$ is equal to $\Delta(u, v)$ (see Figure 10-1a). □

Lemma 10.10. *For every* $t \geq 0$ *and* $0 < \varepsilon < 1/3$*,*

$$\Omega_\tau\left(\varepsilon \cdot e^{-(1+\varepsilon)\cdot\frac{4}{4-\tau^2}\cdot\frac{t^2}{2}}\right) \leq \mathop{\mathrm{Pr}}_{X_1, Y_1 \sim N(0,1)} [\Delta(X_1, Y_1) \geq t] \leq e^{-\frac{4}{4-\tau^2}\cdot\frac{t^2}{2}}.$$

Proof. Similar to the previous lemma, this is a consequence of Lemma 10.8 together with the fact that the squared distance from the origin to the set $\{(x, y) \colon x \geq t \text{ and } \alpha x + \beta y \geq t\}$ is equal to $\frac{4}{4-\tau^2} \cdot t^2$. □

10.5.2.1 Idealized proof

Let us expand Eqn. (10.1) further, assuming that the "idealized" versions of Lemma 10.9 and Lemma 10.10 hold. Namely, we assume that

$$\sigma(u, v) = 1 - e^{-\Delta(u,v)^2/2}; \tag{10.2}$$

and

$$\mathop{\mathrm{Pr}}_{X_1, Y_1 \sim N(0,1)} [\Delta(X_1, Y_1) \geq t] = e^{-\frac{4}{4-\tau^2}\cdot\frac{t^2}{2}}. \tag{10.3}$$

In the next section we redo the computations using the precise bounds for $\sigma(u, v)$ and $\Pr[\Delta(X_1, Y_1) \geq t]$.

Expanding Eqn. (10.1), we have

$$\begin{aligned}
\mathop{\mathrm{E}}_{X_1,Y_1\sim N(0,1)}[\sigma(X_1,Y_1)^{d-1}] &= \int_0^1 \Pr_{X_1,Y_1\sim N(0,1)}[\sigma(X_1,Y_1)\geq t^{\frac{1}{d-1}}]\,dt \\
&= \int_0^1 \Pr_{X_1,Y_1\sim N(0,1)}[e^{-\Delta(X_1,Y_1)^2/2}\leq 1-t^{\frac{1}{d-1}}]\,dt \\
&= \int_0^1 (1-t^{\frac{1}{d-1}})^{\frac{4}{4-\tau^2}}\,dt \\
&= (d-1)\cdot\int_0^1 (1-u)^{\frac{4}{4-\tau^2}}u^{d-2}\,dt \\
&= (d-1)\cdot B\left(\frac{8-\tau^2}{4-\tau^2};d-1\right) \\
&= \Theta_\tau(1)\cdot d^{-\frac{4}{4-\tau^2}}, \qquad (10.4)
\end{aligned}$$

where:

- the first step is a standard expansion of an expectation;
- the second step is due to (10.2);
- the third step is due to (10.3);
- the fourth step is a change of variables;
- the fifth step is a definition of the Beta function;
- the sixth step is due to the Stirling approximation.

Overall, substituting (10.4) into (10.1), we get:

$$\ln\frac{1}{\Pr_{h\sim\mathcal{H}}[h(p)=h(q)]} = \frac{\tau^2}{4-\tau^2}\cdot\ln d \pm O_\tau(1).$$

10.5.2.2 The real proof

We now perform the exact calculations, using the bounds (involving ε) from Lemma 10.9 and Lemma 10.10. We set $\varepsilon = 1/d$ and obtain the following asymptotic statements:

$$\sigma(u,v) = 1 - d^{\pm O(1)}\cdot e^{-(1\pm d^{-\Omega(1)})\cdot\Delta(u,v)^2/2};$$

and

$$\Pr_{X,Y\sim N(0,1)}[\Delta(X,Y)\geq t] = d^{\pm O(1)}\cdot e^{-(1\pm d^{-\Omega(1)})\cdot\frac{4}{4-\tau^2}\cdot\frac{t^2}{2}}.$$

Then, we can repeat the "idealized" proof (see Eqn. (10.4)) verbatim with the new estimates and obtain the final form of Theorem 10.3:

$$\ln\frac{1}{\Pr_{h\sim\mathcal{H}}[h(p)=h(q)]} = \frac{\tau^2}{4-\tau^2}\cdot\ln d \pm O_\tau(\ln\ln d).$$

Note the difference in the low order term between idealized and the real version. As we argue in Section 10.3, the latter $O_\tau(\ln \ln d)$ is, in fact, tight.

10.5.3 Proof of Theorem 10.5

Lemma 10.11. *Let $A \subset S^{d-1}$ be a measurable subset of a sphere with $\mu(A) = \mu_0 \leq 1/2$. Then, for $0 < \tau < \sqrt{2}$, one has*

$$\Pr_{u,v \sim S^{d-1}}[v \in A \mid u \in A, \|u - v\| \leq \tau] = \frac{\Pr_{X,Y \sim N(0,1)}[X \geq \eta \text{ and } \alpha X + \beta Y \geq \eta] + o(1)}{\Pr_{X \sim N(0,1)}[X \geq \eta] + o(1)}, \tag{10.5}$$

where:

- $\alpha = 1 - \frac{\tau^2}{2}$;
- $\beta = \sqrt{\tau^2 - \frac{\tau^4}{4}}$;
- $\eta \in \mathbb{R}$ *is such that* $\Pr_{X \sim N(0,1)}[X \geq \eta] = \mu_0$.

In particular, if $\mu_0 = \Omega(1)$, then

$$\Pr_{u,v \sim S^{d-1}}[v \in A \mid u \in A, \|u - v\| \leq \tau] = \Lambda(\tau, \Phi_c^{-1}(\mu_0)) + o(1).$$

Proof. First, the left-hand side of (10.5) is maximized by a spherical cap of measure μ_0. This follows from Theorem 5 of [98]. So, from now on we assume that A is a spherical cap.

Second, one has

$$\begin{aligned}
&\Pr_{u,v \sim S^{d-1}}[v \in A \mid u \in A, \|u - v\| \leq \tau] \\
&= \Pr_{u,v \sim S^{d-1}}[v \in A \mid u \in A, \|u - v\| = \tau \pm o(1)] + o(1) \\
&= \frac{\Pr_{u \sim S^{d-1}}[u_1 \geq \widetilde{\eta} \text{ and } (\alpha \pm o(1))u_1 + (\beta \pm o(1))u_2 \geq \widetilde{\eta}]}{\Pr_{u \sim S^{d-1}}[u_1 \geq \widetilde{\eta}]} + o(1) \\
&= \frac{\Pr_{X,Y \sim N(0,1)}[X \geq \eta \text{ and } \alpha X + \beta Y \geq \eta] + o(1)}{\Pr_{X \sim N(0,1)}[X \geq \eta] + o(1)},
\end{aligned}$$

where $\widetilde{\eta}$ is such that $\Pr_{u \sim S^{d-1}}[u_1 \geq \widetilde{\eta}] = \mu_0$ and:

- the first step is due to the concentration of measure on the sphere;
- the second step is expansion of the conditional probability;
- the third step is due to the fact that a $O(1)$-dimensional projection of the uniform measure on a sphere of radius $\sqrt{d}$ in $\mathbb{R}^d$ converges in total variation to a standard Gaussian measure [87].

□

Lemma 10.12. *For every $0 < \tau < \sqrt{2}$, the function $\mu \mapsto \Lambda(\tau, \Phi_c^{-1}(\mu))$ is concave for $0 < \mu < 1/2$.*

Proof. Abusing notation, for this proof we denote $\Lambda(\eta) = \Lambda(\tau, \eta)$ and

$$I(\eta) = \Pr_{X,Y \sim N(0,1)}[X \geq \eta \text{ and } \alpha X + \beta Y \geq \eta]$$

(that is, $\Lambda(\eta) = I(\eta)/\Phi_c(\eta)$). One has $\Phi_c'(\eta) = -\frac{e^{-\eta^2/2}}{\sqrt{2\pi}}$ and

$$I'(\eta) = -\sqrt{\frac{2}{\pi}} \cdot e^{-\eta^2/2} \cdot \Phi_c\left(\frac{(1-\alpha)\eta}{\beta}\right).$$

Combining, we get

$$\Lambda'(\eta) = \frac{e^{-\eta^2/2}}{\sqrt{2\pi}} \cdot \frac{I(\eta) - 2\Phi_c(\eta)\Phi_c\left(\frac{(1-\alpha)\eta}{\beta}\right)}{\Phi_c(\eta)^2}$$

and

$$\frac{d\Lambda(\Phi_c^{-1}(\mu))}{d\mu} = \frac{2\Phi_c(\eta^*)\Phi_c\left(\frac{(1-\alpha)\eta^*}{\beta}\right) - I(\eta^*)}{\Phi_c(\eta^*)^2} =: \Pi(\eta^*),$$

where $\eta^* = \eta^*(\mu) = \Phi_c^{-1}(\mu)$. It is sufficient to show that $\Pi(\eta^*)$ is non-decreasing in η^* for $\eta^* \geq 0$.

We have

$$\Pi'(\eta) = \sqrt{\frac{2}{\pi}} \cdot \frac{e^{-\eta^2/2}}{\Phi_c(\eta)^3}\left(2 \cdot \Phi_c(\eta)\Phi_c\left(\frac{(1-\alpha)\eta}{\beta}\right) - I(\eta) - \frac{1-\alpha}{\beta} \cdot e^{\frac{\alpha(1-\alpha)}{\beta^2} \cdot \eta^2}\Phi_c(\eta)^2\right)$$
$$=: \sqrt{\frac{2}{\pi}} \cdot \frac{e^{-\eta^2/2}}{\Phi_c(\eta)^3} \cdot \Omega(\eta).$$

We need to show that $\Omega(\eta) \geq 0$ for $\eta \geq 0$. We will do this by showing that $\Omega'(\eta) \leq 0$ for $\eta \geq 0$ and that $\lim_{\eta \to \infty} \Omega(\eta) = 0$. The latter is obvious, so let us show the former.

$$\Omega'(\eta) = -\frac{2\alpha(1-\alpha)^2}{\beta^3} \cdot e^{\frac{\alpha(1-\alpha)}{\beta^2}\eta^2} \cdot \Phi_c(\eta)^2 \cdot \eta \leq 0$$

for $\eta \geq 0$. □

Now we are ready to prove Theorem 10.5. Let us first assume that all the parts have measure $\Omega(1)$. Later we will show that this assumption can be removed. W.l.o.g. we assume that functions from the family have subsets integers as a range. We have,

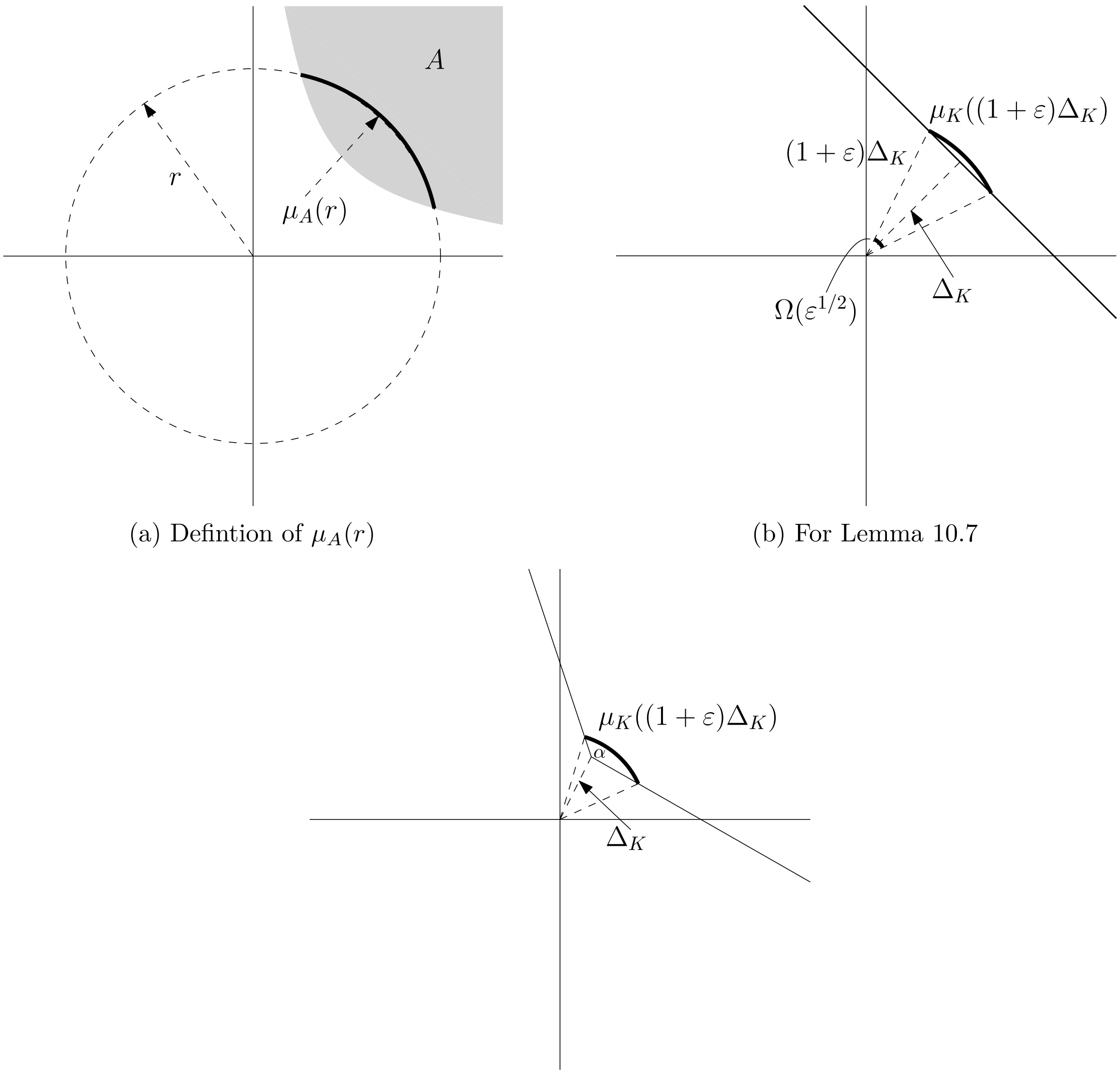
$$
\begin{aligned}
p_1 &\le \Pr_{\substack{u,v\sim S^{d-1}\\ h\sim\mathcal{H}}}[h(u)=h(v) \mid \|u-v\|\le\tau] \\
&= \mathop{\mathrm{E}}_{h\sim\mathcal{H}}\left[\sum_i \mu(h^{-1}(i))\Pr[v\in h^{-1}(i)\mid u\in h^{-1}(i), \|u-v\|\le\tau]\right] \\
&\le \mathop{\mathrm{E}}_{h\sim\mathcal{H}}\left[\sum_i \mu(h^{-1}(i))\Lambda(\tau,\Phi_c^{-1}(\mu(h^{-1}(i))))\right]+o(1) \\
&\le \Lambda\left(\tau,\Phi_c^{-1}\left(\mathop{\mathrm{E}}_{h\sim\mathcal{H}}\left[\sum_i \mu(h^{-1}(i))^2\right]\right)\right)+o(1) \\
&\le \Lambda(\tau,\Phi_c^{-1}(p^*(\mathcal{H})))+o(1),
\end{aligned}
$$

where:

- the first step is by the definition of p_1;
- the third step is due to the condition $\mu(h^{-1}(i))=\Omega(1)$ and Lemma 10.11;
- the fourth step is due to Lemma 10.12 and the assumption $\mu(h^{-1}(i))\le 1/2$;
- the final step is due to the definition of $p^*(\mathcal{H})$.

To get rid of the assumption that a measure of every part is $\Omega(1)$ observe that all parts with measure at most ε contribute to the expectation at most $\varepsilon\cdot T$, since there are at most T pieces in total. Note that if $\varepsilon=o(1)$, then $\varepsilon\cdot T=o(1)$, since we assume T being fixed.

Figure (10-3)


A
r
$\mu_A(r)$
$\mu_K((1+\varepsilon)\Delta_K)$
$(1+\varepsilon)\Delta_K$
$\Omega(\varepsilon^{1/2})$
Δ_K
$\mu_K((1+\varepsilon)\Delta_K)$
α
Δ_K

(a) Defintion of $\mu_A(r)$

(b) For Lemma 10.7

(c) For Lemma 10.8

Chapter 11

Learning Embeddings For Faster Nearest Neighbor Search

11.1 Introduction

Learning representations has become a major field of study over the past decade, with many succesful applications in computer vision, speech recognition, and natural language processing. The primary focus in these directions has been *accuracy*, usually with a focus on classification tasks. Here, the main goal is to learn a representation that enables a standard classifier (e.g., a linear model such as softmax regression) to correctly label the transformed data. However, as learned representations have achieved high accuracy in a wide range of applications, additional goals are becoming increasingly important. One such desideratum is *computational efficiency*: how quickly can we process the learned representations? This question is particularly relevant in the context of large databases, where the goal is to store many millions or even billions of images, texts, and videos. Common instantiations of such settings include web search, recommender systems, near-duplicate detection (e.g., for copyrighted content), and large-scale face recognition.

In this thesis, we study the problem of learning representations through the lenss of *similarity search*. Similarity search, also known as Nearest Neighbor Search (NNS), is a fundamental algorithmic problem with a wide range of applications in machine learning and broader data science. The most common example is similarity search in large corpora such as the aforementioned image databases, segments of speech, or document collections. More recently, NNS has also appeared as a sub-routine in other algorithms such as optimization methods [86], cryptography [146], and large-scale classification [229]. A key challenge in the design of efficient NNS methods is the interaction between the data and the algorithms: How can we exploit structure in the data to enable fast and accurate search? Research on NNS has established a large set of techniques such as kd-trees, locality-sensitive hashing, and quantization methods that utilize various forms of structure in the data.

Traditionally, NNS is used on hand-crafted feature vectors: image similarity search is often performed on SIFT vectors, speakers are identified via their i-vector, and document similarity is computed via tf-idf representations. However, recent progress in deep learning is replacing many of these hand-crafted feature vectors with learned representations. There is often a clear reason: learned representations often lead to higher accuracy and semantically more meaningful NNS results. However, this raises an important question: Do existing NNS algorithms perform well on these new classes of feature vectors? Moreover, learning the representations in NNS also offers an interesting new design space: Instead of adapting the algorithm to the dataset, can we learn a representation that is particularly well suited for fast NNS?

Our main contribution of this chapter is studying this interaction between deep representations and NNS algorithms in more detail. First, we point out that angular gaps are a crucial property of learned representations when it comes to the efficiency of NNS methods. We then explore the design space of neural networks and find that architectural changes such as normalization layers can have significant impact on the angular gaps, even when the classification accuracy is not affected. Based on our experiments, we propose changes to the network architecture and training process that make the resulting representations more amenable to NNS algorithms. Our proposals are intuitive and simple to implement, yet enable speed-ups of 5× or more for nearest neighbor search. Moreover, our changes do not negatively affect the accuracy of the network and sometimes even improve training.

Chapter outline. In Section 11.2, we first explain a property of vectorial representations that is crucial for the performance of NNS methods. This property will guide our evaluation of various neural network design consideration in Section 11.3. We present our experimental results in Section 11.4.

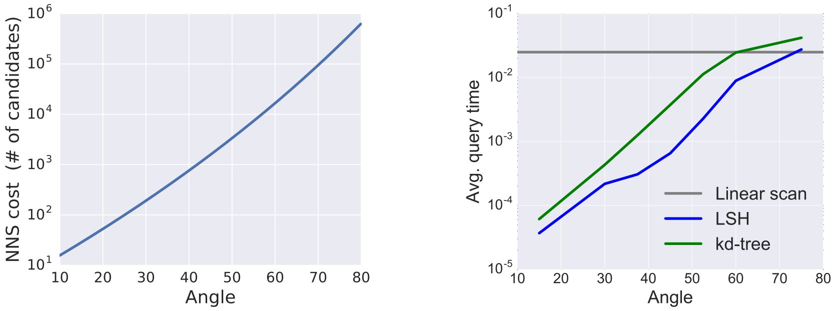
11.2 Smaller angles, faster similarity search

We first take a closer look at what properties of representations enable fast nearest neighbor search (NNS). We comprehensively address this question both from a theoretical and an empirical perspective. The resulting insights will then guide our approach to learning representations that are good for NNS.

Preliminaries. Before we begin with our investigations, we briefly define the NNS problem formally. The goal in NNS is to first preprocess a dataset D of points from a metric (X, dist) so that we can later quickly answer queris of the following form: "Given a query point $\mathbf{q} \in X$, which point $\mathbf{p} \in D$ minimizes the distance $\mathrm{dist}(\mathbf{q}, \mathbf{p})$?". A common concrete instance of the NNS problem is similarity search under cosine similarity. Here, the set of points D contains unit vectors in $\mathbb{R}^d$ and the distance measure dist is the angular distance $\angle(\mathbf{q}, \mathbf{p}) \overset{\text{def}}{=} \cos^{-1} \frac{\mathbf{q}^\top \mathbf{p}}{\|\mathbf{q}\|_2 \|\mathbf{p}\|_2}$

11.2.1 Theoretical analysis

Different NNS algorithms rely on different properties on the data in order to enable faster similarity search. In this thesis, we focus on locality-sensitive hashing (LSH), a well-establish NNS method, where it is possible to analyze the impact of *distance gaps* on the query time quantitatively [112, 127]. Concretely, we study the cosine similarity NNS problem mentioned above and analyze the running time of the popular hyperplane LSH [70].



(a) Analytically estimated NNS cost (b) Empirically estimated NNS cost

Figure (11-1): Analytically and empirically estimated costs of an approximate nearest neighbor search as a function of the angle between the query and the nearest neighbor. The analytical cost is given as the number of distance computations between the query point and the candidate points returned by the LSH table. The empirical cost is given as the total time per query on a laptop computer from 2015 (CPU: Intel Core i7-4980HQ).

Suppose that a query q makes an angle of α with the closest vector. Under reasonable assumptions on the balance of the data, we show the following statement in Section 11.7.2. If we want to ensure that the probability of finding the closest vector is at least, $1-\exp(-5) \approx 99.33\%$, the expected number of candidates considered in a nearest neighbor search, for a dataset of size n, grows with α as

$$N(\alpha) \;=\; 5n^{-\log_2(1-\frac{\alpha}{\pi})} \qquad (11.1)$$

We defer our formal theorem with its proof to the supplementary material.

Equation (11.1) allows us to estimate the number of nearest neighbor candidates (i.e., the time complexity of a query) as we vary the angle between the query vector and the nearest neighbor. A crucial property of this relation is that smaller angles between nearest neighbor and the query point enable significantly faster NNS. We illustrate by substituting concrete numbers. For $n = 10^6$ data points, improving the angle α from $\pi/3$ (60 degrees) to $\pi/4$ (45 degrees) reduces the number of nearest neighbor candidates (i.e., distance computations) from about 16K to 1.5K, whichis roughly a 10× speed-up.. In Figure 11-1(a), we plot the expression (11.1) as a function of the angle α for $n = 10^6$.

11.2.2 Empirical analysis

While the above theoretical analysis is for the relatively simple hyperplane LSH, the same trend also applies to more sophisticated NNS algorithms. We show that two state-of-the-art NNS methods show empirical behavior that is very similar to Figure 11-1(a). Concretely, we compare the following two ANN implementations that are commonly used:

- Annoy, which implements a variant of the classical kd-tree [44].
- FALCONN, which implements the fastest known LSH family for angular distance [185].

We generate a dataset of $n = 10^6$ random unit vectors and plant queries at a given angle

α from a randomly selected subset of database points. Since the dataset is drawn from the uniform distribution over the hypersphere, it is "well-spread" and matches the assumptions of our analysis above. For each angle α, we generate 10,000 queries and report the average query time for finding the correct nearest neighbor with empirical success probability 0.9.

Figure 11-1(b) shows that the query times of the two ANN implementations largely agree with our theoretical prediction.[1] An important aspect of the plot is the linear scan baseline, i.e., the cost of computing all distances between query and database points (which always finds the correct nearest neighbor). At 90% relative accuracy, the ANN algorithms we evaluated only improve over a linear scan once the angle is around 60 degrees. For larger angles, current ANN algorithms cannot maintain both high accuracy and faster inference speed. Overall, our empirical findings also underline the importance of the angle between query and nearest neighbor for NNS performance.

11.3 Learning representations for smaller angles

In the previous section, we have seen that the angle between the query and its nearest neighbor play a crucial role in the performance of fast NNS methods. We now build on this insight to learn representations where this angle is small. As deep neural networks have become widely used for learning representations, we focus specifically on how to modify neural networks and their training process to enable faster NNS.

11.3.1 Preliminaries

We study neural networks for supervised classification into one of C classes. Each example $\mathbf{x}$ comes from a domain $\mathcal{X}$, typically $\mathbb{R}^p$, and our goal is to predict its label $y \in [C]$. Given training examples $(x_i, y_i) \in \mathcal{X} \times [C]$, we learn model parameters by optimizing a certain loss function on the training data. We then use the resulting model to predict labels.

For our purposes, a neural network consists of three sets of parameters: (i) the network weights $\theta \in \Theta$ defining a mapping $\phi_\theta : \mathcal{X} \to \mathbb{R}^d$, (ii) a set of *class vectors* $\{v_y\}_{y\in[C]}$ that also lie in $\mathbb{R}^d$, and (iii) a bias vector $\mathbf{b} \in \mathbb{R}^C$. We will often write the class vectors as a matrix V. We refer to the vectors $\phi_\theta(x)$ as the *representation vectors.* We employ the commonly used softmax loss, which is defined as:

$$\ell(x, y; \theta, V, \mathbf{b}) \stackrel{\text{def}}{=} -\log \frac{\exp(\phi_\theta(x)^\top v_y + b_y)}{\sum_{j\in[C]} \exp(\phi_\theta(x)^\top v_j + b_j)} \ .$$

Noting that $\phi_\theta(x)^\top v_j + b_j = (\phi_\theta(x), 1)^\top (v_j, b_j)$, the bias term is superflous for the purposes of the softmax objective. To simplify the discussion, we will therefore ignore the biases and

[1] There are some mismatches, for instance in the regime of small angles. For FALCONN, this is because a real LSH implementation also needs to compute the hash functions, which is not included in our theoretical analysis above.

write:

$$\ell(x, y; \theta, V) \overset{\text{def}}{=} -\log \frac{\exp(\phi_\theta(x)^\top v_y)}{\sum_{j\in[C]} \exp(\phi_\theta(x)^\top v_j)} \ .$$

The Softmax Objective. To understand the softmax objective better, we write it as

$$\ell(x, y; \theta, V) = -\phi_\theta(x)^\top v_y + \log \sum_{j\in[C]} \exp(\phi_\theta(x)^\top v_j).$$

The training process aims to achieve a small loss, i.e., we want the correct dot product $\phi_\theta(x)^\top v_y$ to be large and the other dot products $\phi_\theta(x)^\top v_j$ to be small. We further re-write the inner product in order to highlight the connection to the angle between class vectors and representation vectors:

$$\phi_\theta(x)^\top v_j = \|\phi_\theta(x)\|_2 \, \|v_j\|_2 \, \cos\angle(\phi_\theta(x), v_j) \ . \tag{11.2}$$

Viewed this way, three properties of the loss function become apparent:

1. While the relative norm of the class vectors does influence the predictions of the model, the overall scale does not. Doubling the lengths of all class vectors has no influence on the prediction accuracy, yet it can change the softmax loss.
2. The norm of the representation vector $\phi_\theta(x)$ is irrelevant for the 0-1 prediction loss objective. Yet it can have significant influence on the softmax loss.
3. A small angle between the representation vector $\phi_\theta(x)$ and the class vector v_j is beneficial for both the prediction loss objective as well as the softmax loss objective. Other things being equal, we prefer a model that maps examples to representation vectors that are well-aligned with the correct class vectors.

Next section, we view these three terms through the lens of angles for NNS problems.

11.3.2 Improving the representation quality: smaller angles

When training a representation with small angles between query and nearest neighbor, we will view the representation before the softmax layer as the query point and the class vectors in the softmax as dataset points. Our main focus is then to improve the *mean correct angle* between the representation vector $\phi_\theta(x)$ and the correct class vector v_y. For an evaluation set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, we define the mean correct angle as

$$\frac{1}{m} \sum_{i=1}^{m} \angle(\phi_\theta(x_i), v_{y_i}) \ .$$

In this section, we describe how our modifications improve the mean correct angle. As we have seen in Section 11.2, this angle is an algorithm-agnostic quantity that dictates the speed-ups we can expect from various ANN algorithms.

To illustrate the impact of various network design choices, we evaluate the effect of each modification on a medium-sized dataset (in the supplementary material, we also show the effect on a smaller dataset that facilitates easier visualization and aids intuition). In Section 11.4, we then evaluate our modifications on two larger datasets.

multiMNIST. This dataset is derived from the popular MNIST dataset [148]. We construct large-multiclass variants of MNIST by concatenating multiple digits (see Section 11.4 for more details). We call these variants *multiMNIST*. The multiMNIST instances allow us to confirm our intuitions on a larger dataset with a higher-dimensional softmax and more classes. Here, we use multiMNIST instances with 1,000 to 10,000 classes.

We investigate the impact of three training modifications.

Control Representation Vector Norms As discussed in Section 11.3.1, scaling the representation vectors $\phi_\theta(x)$ can change the loss function without changing the prediction accuracy. The dynamics of (stochastic) gradient descent update parameters so as to increase the inner product $\phi_\theta(x)^\top v_j$, or equivalently, the product

$$\|\phi_\theta(x)\|_2 \cdot \|v_j\|_2 \cdot \cos \angle(\phi_\theta(x), v_j) \ .$$

With a constraint on the term $\|\phi_\theta(x)\|_2$, the training process tends to increase the remaining terms instead, which leads to smaller angles $\angle(\phi_\theta(x), v_j)$. We consider several options for controlling the norms of the representation vectors. Layer Normalization [24] ensures that the presoftmax vector has mean entry zero and a constant norm. Scaling (which is similar to Weight Normalization [197]) divides each presoftmax vector by its mean before propogating and thus ensures a constant norm (but not necessarily mean zero). A variant of scaling just uses the normalized vector but does not back-prop through the normalization. Batch normalization [129] can be used to ensure that each activation has mean 0 and variance 1 at equilibrium, which implies that the mean squared norm of the representation is d. Finally, one can simply "regularize" by adding a $\lambda \frac{1}{n} \sum_{i=1}^{n} \|\phi_\theta(x_i)\|_2^2$ term to the loss.

While batch norm is often used in networks for other reasons, it is unclear *a priori* which of these options would work best from the viewpoints of the angles, the final accuracy and the convergence speed. We compare these options in more detail in Section 11.7.4. Our experiments show that from the point of view of accuracy, convergence speed and angles, batch norm empirically gives better results than the other options.

For lack of space, we only show the comparison with batch norm in the rest of this section. Using batch norm does not hurt the accuracy on the evaluation set, nor the convergence speed. The mean angle improves and the representation vector norms are now significantly smaller than in the unconstrained case.

Use All Directions In a standard ReLU-based neural network, the representation vector $\phi(x)$ is the output of a ReLU. In particular, this means that each coordinate of $\phi(x)$ is non-negative, i.e., the representation vector always lies in the non-negative orthant. While the non-linearity in neural networks is an important source of their representation power, the ReLU at the last layer yields a more difficult dataset for NNS methods. Instead of producing a representation on the entire unit sphere, the representation vector is restricted to lie in the non-negative orthant. Implicitly, this also restricts the locations of the class vectors in the softmax: in order to achieve reasonable angle, the class vectors must be close

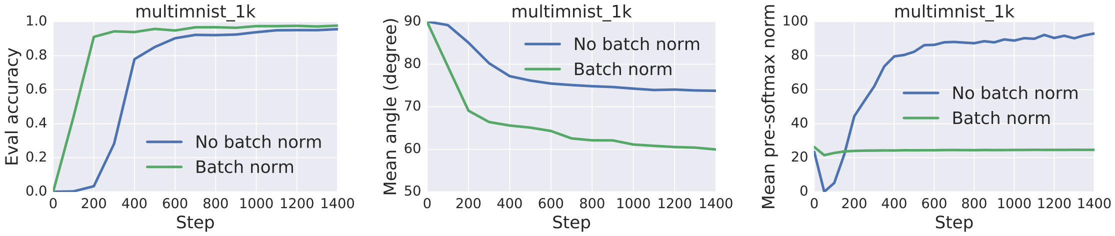


Figure (11-2): The effect of batch normalization on the evaluation accuracy, mean correct angle, and norm of the representation (pre-softmax) vectors. The plots show two training runs on our multiMNIST dataset with 1,000 classes. The norm of the representation vectors is now nearly constant, and the mean correct angle improves significantly. The normalization also improves training speed.

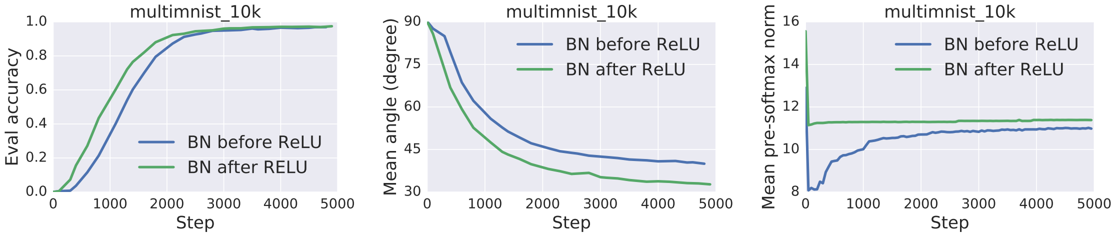


Figure (11-3): The effect of swapping ReLU and batch normalization before the softmax layer. The plots show two training runs on our multiMNIST dataset with 10,000 classes. The norm of the representation vectors show less variation during training, and the mean correct angle improves. The normalization also improves training speed.

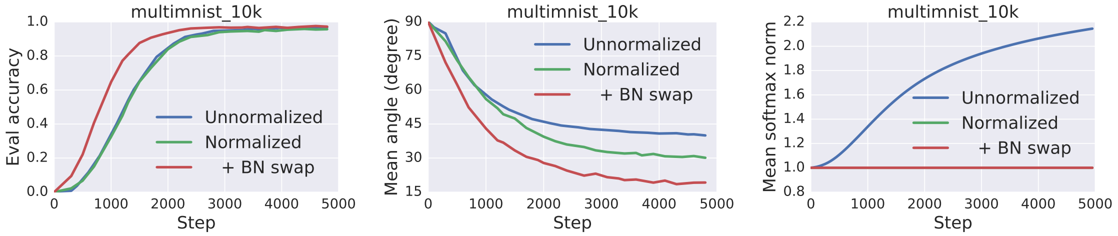


Figure (11-4): The effect of normalizing the class vectors in the softmax layer. The plots show two training runs on our multiMNIST dataset with 10,000 classes. The average norm of class vectors is now constant and significantly smaller than before. Moreover, the mean correct angle improves while the accuracy is not negatively affected. Importantly, the positive effects are cumulative with the previous modification of swapping the order of ReLU and batch normalization (we are not comparing to a baseline without batch normalization because we could not train a 10,000 class multiMNIST dataset without batch normalization).

to the non-negative orthant. To ameliorate this issue, we propose to either remove the last layer ReLU or to place it *before* the batch norm so that the final representation vector $\phi_\theta(x)$ can have both positive and negative coordinates.

As we can see in Figure 11-3, this modification improves the mean correct angle on multiMNIST. Moreover, the network achieves a good classification accuracy faster, and the representation vector norms are better controlled.

Equalize Class Vector Lengths NNS is often an easier problem when all dataset points have the same norm as various distance measure such as Euclidean distance, maximum inner product, and cosine similarity are then comparable. Hence it is desirable to ensure that all class vectors in the softmax layer have the same norm. In many trained networks, the ratio between largest and smallest class vector norm is already fairly small, e.g., about 1.5 in case of the Inception architecture [218] for Imagenet. The fact that this ratio is not too large suggests that forcing it to be exactly 1 *while training* may not hurt the performance of the model. This is our third modification: we constrain the norms of the class vectors to all be equal. This constraint can be realized via projected gradient descent: we first initialize the class vectors to have unit norm.[1] Then after every gradient step (or a small number of steps), we re-normalize the class vectors to be norm 1 again.

This modification has two advantages. First, it guarantees that the NNS problem is exactly an angular NNS problem. Moreover, the modification also helps in achieving smaller angles. Intuitively, we constrain the second term $\|v_j\|_2$ in Equation (11.2) to be 1. We see the effect of this modification on the multiMNIST dataset in Figure 11-4. The mean correct angle improves and the class vector norms are now significantly smaller than in the unconstrained case. Importantly, the improvement in mean correct angle is *cumulative* with the previous modifications. Overall, we not only obtain an angular NNS problem (instead of MIPS), we also have a better conditioned problem due to the smaller angles.

[1] The constant 1 is somewhat arbitrary and other values may be considered.

11.4 Experiments

We now evaluate our training modifications on two large datasets. We begin by describing these two datasets that we use in our experiments, together with the corresponding network architecture. In Section 11.7.3, we also report results on a third dataset (CelebA).

MultiMNIST As in Section 11.3, we use our multiclass variant of the classical MNIST dataset [148]. MNIST contains grayscale images of hand-written digits $\{0, 1, 2, \ldots, 9\}$ with fixed size 28×28. To create a multiclass dataset, we horizontally concatenate c of these images together to form composed images. We label each composed image with the concatenation of the corresponding digits. Thus, for example $c = 2$ corresponds to class space $\{00, 01, \ldots, 99\}$ and image size 28×56. We construct a training dataset by concatenating random images from the MNIST train dataset, and proceed similarly for the evaluation dataset. We refer to a multiMNIST dataset with C classes as multiMNIST_C. Figure 11-9 shows an example image from multiMNIST_100K (so $c = 5$).

Our model architecture is based on the MNIST model architecture in the TensorFlow tutorial [1].[1] The network consists of two 5×5 convolutional layers with 32 and 64 channels, respectively. Both convolutional layers are followed by batch norm, ReLu and MaxPool. We then use a fully connected layer to project down to d dimensions, where d is the softmax dimension typically taken to be between 256 and 1024. We train the network with stochastic gradient descent on a single machine with a GPU.

Sports1M We perform our second set of experiments on the Sports1M dataset [138].[2] This dataset contains about 1 million videos. As in [229], we construct a multiclass problem where the examples are the video frames, and the labels are the video ids. We use the first half of each video as training examples and the second half as test examples. After removing some videos to ensure a common format, this gives us a classification problem with about 850,000 labels.

We convert each frame to its VGG features [213], using a pretrained VGG16 network.[3] In particular, we use the output of the last convolutional layer as features, followed by a fully connected layer of size 1024 and a softmax layer of dimension $d = 256$. To accelerate training, we employ a sampled softmax loss approximation with sample size 8192. This also allows us to test whether our training modifications are compatible with the sampled softmax approximation (the multiMNIST test cases are trained with a full softmax). We train our models using a distributed TensorFlow setup employing 15 workers with one GPU each and 10 parameter servers.

Evaluation Objectives The goal of our experiments is to investigate two main questions: (i) How effective are our modifications for improving the angle between the representation vectors and the class vectors? (ii) Do our modifications hurt the accuracy of the model?

[1] Available at `https://www.tensorflow.org/get_started/mnist/pros`.

[2] Available at `http://cs.stanford.edu/people/karpathy/deepvideo/`.

[3] Downloaded from `https://www.cs.toronto.edu/~frossard/post/vgg16/`.

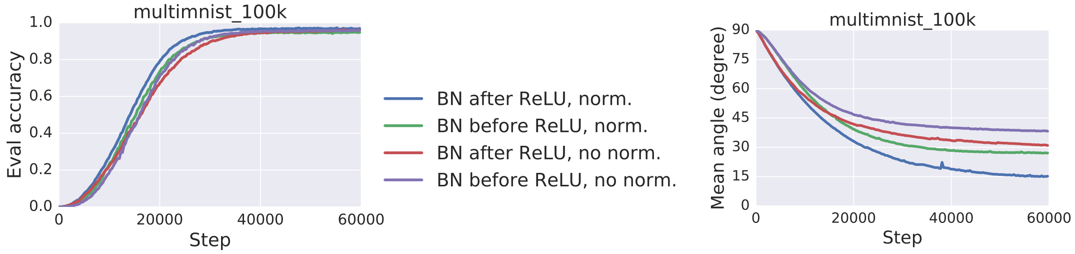


Figure (11-5): The effect of our training modifications on a multiMNIST_100K dataset. The mean correct angle improves significantly while the evaluation accuracy is not affected negatively.

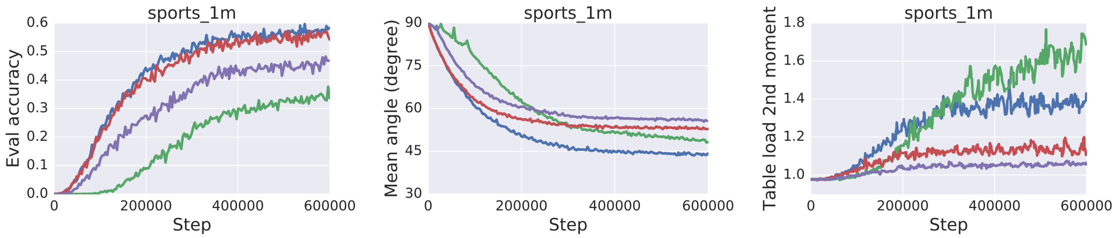


Figure (11-6): The effect of our training modifications on the Sporst1M dataset (the legend is the same as in Figure 11-5 above). The mean correct angle is significantly smaller. The evaluation accuracy also improves. Moreover, the points remain evenly spread over the unit sphere, i.e., the "table balance" quantity $M(P,h)/M^*$ remains close to 1, especially for the best-performing combination of our techniques.

Moreover, we also investigate the "well-distributedness" of the class vectors. In Theorem 11.1 in Section 11.7.2, we define a specific measure of imbalance that impacts the run time of an LSH-based nearest neighbor scheme. More precisely, we show that as long as second moment of the table loads, denoted by $M(P,h)$ remains close to an "ideal" value M^*, smaller angles improve the NNS performance. We want to determine if this measure of imbalance, $M(P,h)/M^*$, remains small.

Results The results of our experiments on multiMNIST_100K are shown in Figure 11-5. The baseline training approach uses the common configuration of applying batch normalization before the nonlinearity and does not normalize the softmax vectors. Compared to this baseline, our training modifications decrease the mean correct angle from about 40 to 15. This corresponds to a 30× faster softmax evaluation at inference time. Moreover, our modifications lead to no loss in accuracy.

Figure 11-6 shows the results on the Sports1M dataset. In this case, our modifications improve the accuracy after 600K steps. Placing the batch normalization after the nonlinearity seems particularly important. The mean correct angle decreases by slightly more than 10 degrees. As we see in Figure 11-7, this yields a 5× faster nearest neighbor query time.

The right panel in Figure 11-6 shows the second moment of the table loads relative to the ideal value, i.e., $M(P,h)/M^*$. While our modifications lead to an increase in this value, the

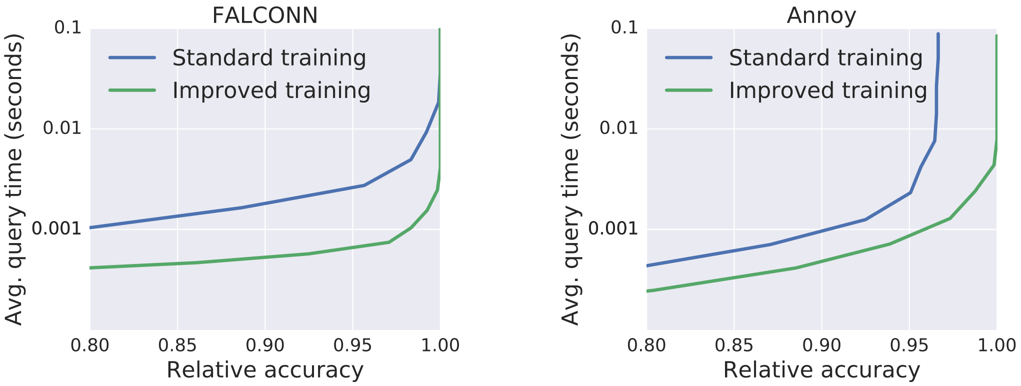


Figure (11-7): Effect of our training modifications on the query times of nearest neighbor algorithms. We report the *relative* accuracies, i.e., the probability of finding the correct nearest neighbor conditioned on the model being correct. Both diagrams are for the Sports1M dataset.

For LSH as implemented in the FALCONN library (left), our training yields a 5× speed-up in the relevant high accuracy regime. The variant of kd-trees implemented in the Annoy library (right) does not reach relative accuracy 1 when the softmax is trained using the standard approach. In contrast, the softmax resulting from our training techniques is more amenable to the kd-tree algorithm. Again, we obtain faster query times for fixed accuracy.

increase is small and stays around 1.4× for the relevant configuration that achieves the best accuracy and mean correct angle.

In summary, our experiments demonstrate that our modifications obtain significantly smaller angles between representation vectors and the correct class vectors. This translates to an order of magnitude improvement in inference speed, and leads to learning better representations. The improvement comes at no decrease in overall accuracy.

11.5 Related work

Liu et al. [153] proposes a method for training with larger angular distance gaps. In contrast to our approach, the authors modify the loss function, not the training process or network architecture. The focus of their work is also on classification accuracy and not fast similarity search. The authors do not quantify the improvements in angular distance and train on datasets with a relatively small number of classes (100 or less).

Liu et al. [152] also modify the loss function for learning representations with angular distance gaps. Again, the focus of their work is on accuracy and not fast similarity search. In particular, the authors do not investigate the effect of their changes on the angular gap on large datasets. The focus of our work is on fast similarity search and we evaluate various network modifications with end-to-end experiments using state-of-the art NNS methods.

11.6 Conclusions and future work

We have demonstrated how to learn representations specifically for faster similarity search in large datasets. To this end, we have studied multiple modifications to neural network training and architecture that lead to a smaller angle between the learned representation vectors produced by the network and the class vectors in the softmax layer. This angle is a crucial measure of performance for approximate nearest neighbor algorithms and enables a 5× speed-up in query times. An interesting direction for future research is whether these insights can also lead to faster training of large multiclass networks, which are common in language models or recommender systems.

11.7 Further results

11.7.1 Visualization of our approach

2D-GMM. To visualize the effect of our modifications, we consider a synthetic example that allows us to illustrate the geometry of the softmax layer. The data is drawn from a small number of well separated Gaussians in two dimensions. The labels correspond to the Gaussians they are sampled from. The network we train has two hidden layers with ten hidden units each, followed by a projection to two dimensions. We use ReLU non-linearities. We train the network using AdaGrad [94] to convergence, i.e., (near) zero training loss. The appealing property of this synthetic example is that we can plot the representation vectors $\phi_\theta(x)$ and class vectors v_i directly in two dimensions.[1] The results are in line with those on multimnist and are meant to help visualize the effect of our modifications.

We show in Figure 11-8 the effect of our modifications on this dataset. In Figures 11-8(a) and (b). we see that applying batch normalization leads to pre-softmax vectors that have approximately the same norm. Moreover, the angle between the pre-softmax vectors and the correct class vector improves noticeably.

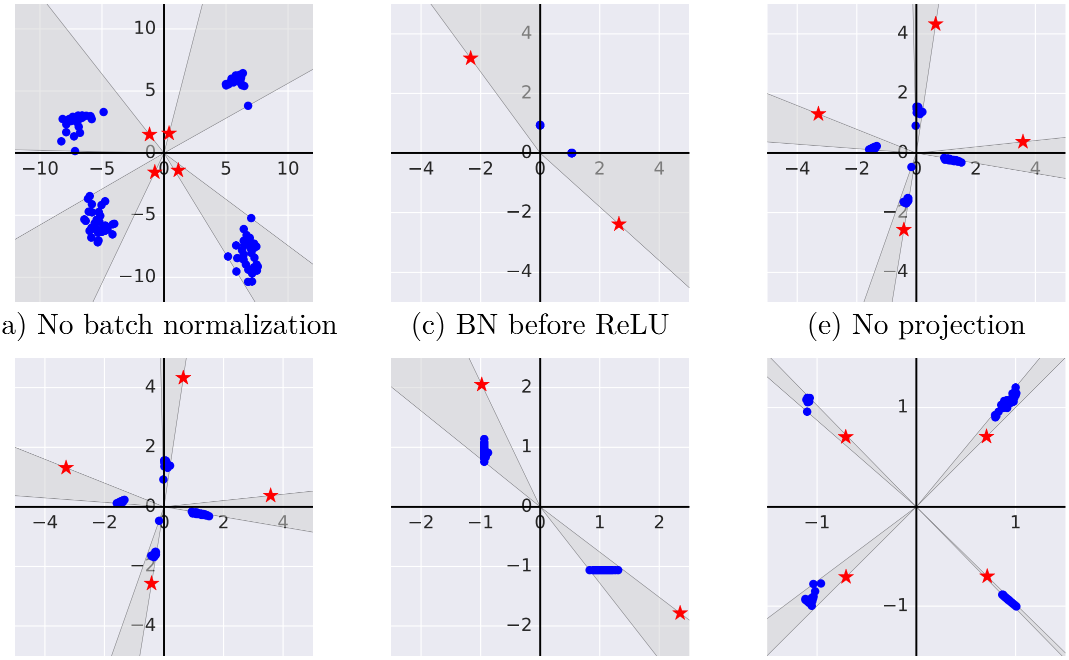
Figures 11-8(c) and (d) show the effect of using all directions on the 2D-GMM dataset for a mixture of 2 Gaussians. When the ReLU is after the batch norm, the pre-softmax vectors are in the non-negative quadrant. Moving the batch norm after the ReLU leads to smaller angles.

Finally, we visualize the effect of projecting the softmax vectors in Figures 11-8(e) and (f). This modification leads to even thinner sectors, i.e., even smaller angles between the representation vectors and the correct class vectors.

11.7.2 Theoretical analysis of NNS

For the case of locality-sensitive hashing (LSH), we can analyze the speed vs. accuracy tradeoff quantitatively [112, 127].

[1] A complementary approach would be to train a higher-dimensional softmax and project the vectors to two dimensions for visualization. However, such a projection necessarily distorts the geometry. Hence we have trained a two-dimensional softmax to ensure a faithful illustration.



(a) No batch normalization

(c) BN before ReLU

(e) No projection

(b) With batch normalization

(d) BN after ReLU

(f) With projection

Figure (11-8): Visualization of our training modifications for a two-dimensional softmax trained on a synthetic dataset. Each column shows the effect of one modification. The top row contains the baseline result without the respective training modification and the bottom row shows the result with modification.

The data is drawn from a mixture of Gaussians with equally spread means and variance 0.1 in each direction. Since a ReLU unit before the softmax limits the expressivity of the model, we could only train two classes with the two-dimensional softmax and a final ReLU (middle column). Hence we did not employ a ReLU before the softmax layer in the left and right columns. In all cases, the network trained to accuracy 1.

The blue points in the plots are the representation vectors of the examples and the red $\star$'s are the class vectors in the softmax. The gray sectors indicate the maximum angles between the representation vectors and the corresponding class vectors. The plots show that our modifications lead to significantly smaller angles.

Suppose that we use a hyperplane LSH scheme with k hyperplanes per table and m independent hash tables [70]. For the purpose of our analysis, we consider a variant where we select a candidate set of class vectors as follows: we first find the hash bucket of the query vector in each of the m tables and collect all database points in these hash buckets as a candidate set. Among these candidates, we then compute exact distances in order to find the (approximate) nearest neighbor. The goal is to choose the parameters m and k so that (i) the probability of finding the closest vector in the candidates set is close to 1, and (ii) the number of candidates is small.

In order to analyze this LSH scheme, we need to consider two questions:

- How many hash tables m do we need to query so that we find the nearest neighbor in one of them with good probability?
- How many spurious candidate vectors will we encounter in the m hash buckets we explore?

We now address the first question. For a query q, the probability that that a dataset point within an angle of α lies in the same LSH bucket as q is $p_\alpha \overset{\text{def}}{=} (1 - \frac{\alpha}{\pi})^k$. To ensure that the failure probability is small, say, $\exp(-5) \approx 0.67\%$, it suffices to take m to be $5/p_\alpha$. It can then be easily verified that we should set k to be as large as possible in order to obtain a small candidate set. However, taking k to be larger than $\log n$ leads to many hash buckets being empty, which increases the cost of generating the set of candidates itself. So we set k to be $\lfloor \log n \rfloor$; see also [112, Theorem 3.4].

Next, we bound the total number of candidates in the hash buckets. The expected number of points in a given hash bucket is $n/2^k$ (the expectation is taken over the randomness in the hash function). However, this quantity does not necessarily bound the expected number of candidates during a query. The event that q lands in a certain hash bucket may be correlated with a large number of database points occupying the same bucket (consider a heavily clustered dataset). To ensure a fast query time, we want to bound the expected number of points in a bucket conditioned on q landing in the bucket. This quantity is a function of the data and query distribution. Under the assumption that the data points are reasonably spread out (i.e., each hash bucket contains roughly the same number of database points), the bound of $n/2^k$ for a single table is approximately correct. To measure how "well-spread" the dataset P is under the hash function $h : \mathbb{R}^d \to [2^k]$, we define the following quantity $M(P, h)$:

$$M(P, h) \overset{\text{def}}{=} \sum_{i \in [2^k]} |h^{-1}(i) \cap P|^2 \ .$$

The table load $M(P, h)$ simply sums the squares of the hash bucket occupancy counts, which penalizes evenly spread out point sets less than highly clustered ones. With this definition, we can now analyze the expected number of spurious candidates under the assumption that the query q comes from the same data distribution as the database P. In the regime where the mean correct angle between pre-softmax vectors and class vectors is small, this is a well justified assumption.

Theorem 11.1. *Suppose that the query point q is chosen uniformly at random from the database P and that the hash function h was used to build the hash table. Then the expected number of points in the bucket containing q is $M(P, h)/n$. Moreover, if the hash function* *values* $\{h(p)\}_{p \in P}$ *are independent random variables, then*

$$\mathbb{E}[M(P,h)] \;=\; M^* \overset{\text{def}}{=} \frac{n^2}{2^k} + n\left(1 - \frac{1}{2^k}\right) \;. \tag{11.3}$$

Proof. For the first part, let q be a point from P chosen at random. The quantity of interest is

$$\begin{aligned}
\mathbb{E}_q[\text{Number of points in bucket containing } q] &= \sum_{q' \in P} \mathbb{E}_q[\mathbf{1}(q' \text{ and } q \text{ in same bucket})] \\
&= \sum_{q' \in P} \mathbb{E}_q[\mathbf{1}(h(q) = h(q'))] \\
&= \sum_{q' \in P} |h^{-1}(h(q')) \cap P|/n \\
&= \sum_{i \in [2^k]} \sum_{q' \in P : h(q') = i} |h^{-1}(i) \cap P|/n \\
&= \sum_{i \in [2^k]} |h^{-1}(i) \cap P|^2/n \\
&= M(P,h)/n.
\end{aligned}$$

For the second part, we follow a similar chain of equalities and write:

$$\begin{aligned}
M(P,h) &= \sum_{i \in [2^k]} |h^{-1}(i) \cap P|^2 \\
&= \sum_{i \in [2^k]} \sum_{q' \in P : h(q') = i} |h^{-1}(i) \cap P| \\
&= \sum_{q' \in P} |h^{-1}(h(q')) \cap P| \\
&= \sum_{q' \in P} n\mathbb{E}_q[\mathbf{1}(h(q) = h(q'))].
\end{aligned}$$

Now note that $\mathbb{E}_q[\mathbf{1}(h(q) = h(q'))]$ is 1 when $q = q'$. Under the independence assumption, this expectation is $\frac{1}{2^k}$ in every other case. Thus we get

$$\begin{aligned}
\mathbb{E}[M(P,h)] &= \sum_{q' \in P} n\mathbb{E}_q[\mathbf{1}(h(q) = h(q'))] \\
&= n + n(n-1)\frac{1}{2^k} \\
&= \frac{n^2}{2^k} + n\left(1 - \frac{1}{2^k}\right).
\end{aligned}$$

The claim follows. □

If for a specific database P and hash function h, $M(P,h)/M^*$ is close to 1, then the LSH table is sufficiently balanced for the inference to be efficient. Note that if we set k to be

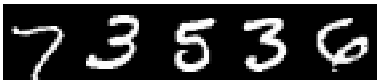


Figure (11-9): An example from multiMNIST_100k; label 73536.

$\lfloor \log_2 n \rfloor$, then M/n is at most 3, i.e., we have a small constant number of candidate points per hash bucket. As a result, the number of candidates in the nearest neighbor search roughly equals the number of hash buckets. Using the earlier formula for the collision probability p_α, we get the following formula for the number of candidate points as a function of angle α:

$$\frac{5}{p_\alpha} \;=\; 5n^{-\log_2(1-\frac{\alpha}{\pi})}$$

This expression allows us to estimate the number of candidate distance computations as we decrease the angle between the pre-softmax vector and the correct class vectors. We illustrate our estimate by substituting concrete numbers: if $n = 10^6$, improving α from $\pi/3$ (60 degrees) to $\pi/4$ (45 degrees) reduces the number of hash bucket lookups from about 16K to 1.5K, i.e., an order of magnitude improvement. In Figure 11-1(a), we plot the expression (11.7.2) as a function of the angle α for $n = 10^6$.

11.7.3 Experiments on CelebA

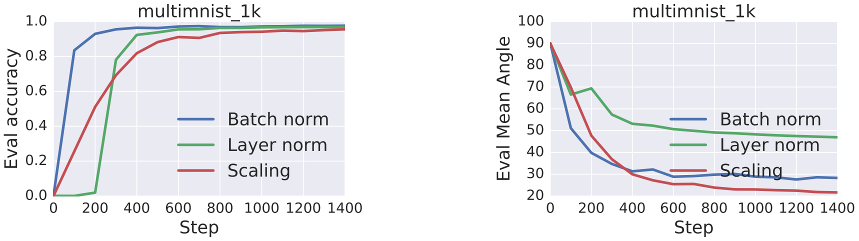
To investigate the impact of various network modifications on a third dataset, we also conduct experiments on CelebA [154]. CelebA consists of about 200K face images of roughly 10K different celebrities. The goal of our experiment is to evaluate the quality of the representation learned by a standard Resnet model with 32 layers. In particular, we want to understand how well the learned representation *generalizes* to previously unseen identities (not only unseen images of celebrities that are also present in the training dataset). To this end, we learn a representation on about 8K identities via the multiclass approach that we also employed for other datasets. At evaluation time, we then take 2K unseen identities and compute representations for all examples (roughly 20K images). We compute two quantities: (i) the accuracy of a nearest-neighbor classifier on the evaluation dataset. (ii) the angular gaps to the nearest neighbor of the same class.

Our results show that normalizing the class vectors significantly improves both accuracy and angular gaps. Combining softmax normalization with the Batch Normalization – ReLU swap yields an accuracy of 74%, which is significantly higher than baselines without softmax normalization that achieve 60% accuracy (for both possible orders of normaliation layer an non-linearity). Moreover, the angle between query and nearest neighbor improves by about 8 degree while the set of evaluation points remains as balanced as the baselines without normalization of the class vectors.

11.7.4 Comparison of normalization options

We consider various options for controlling the norms of the representation vectors.

BN: Batch Normalization.



(a) Eval Accuracy

(b) Mean Angle

Figure (11-10): Comparison of various Normalization Options on MultiMNIST 1K

LN: Layer Normalization [24].

Scaling: Using a normalized representation $\hat{\phi}_\theta(x) = \phi_\theta(x)/\|\phi_\theta(x)\|$.

To evaluate these options, we used multiMNIST data set with 1K classes. We use a network identical to that in Section 11.4 with a representation layer of 256 dimensions. We compare these options on various axes: the final accuracy, the speed of convergence and the mean angle. For each of the configurations, we did a grid search over learning rates to pick the best one. The final accuracy and angles are shown in Figure 11-10. As the figures show, the network with batch norm gives nearly the same angles as the best of these options. In terms of convergence speed and accuracy, batch norm is the noticeably better than the other options.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.* 2015. URL: `http://download.tensorflow.org/paper/whitepaper2015.pdf`.

[2] Amir Abboud. personal communication. 2016.

[3] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. "Tight hardness results for LCS and other sequence similarity measures". In: *Symposium on Foundations of Computer Science (FOCS).* 2015.

[4] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. "Subcubic equivalences between graph centrality problems, APSP and diameter". In: *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM. 2015, pp. 1681–1697.

[5] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. "Matching triangles and basing hardness on an extremely popular conjecture". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing.* ACM. 2015, pp. 41–50.

[6] Amir Abboud and Virginia Vassilevska Williams. "Popular conjectures imply strong lower bounds for dynamic problems". In: *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on.* IEEE. 2014, pp. 434–443.

[7] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. "Consequences of faster alignment of sequences". In: *International Colloquium on Automata, Languages, and Programming (ICALP).* 2014.

[8] Alekh Agarwal and Léon Bottou. "A Lower Bound for the Optimization of Finite Sums". In: *International Conference on Machine Learning (ICML).* 2015.

[9] Alekh Agarwal, Sahand Negahban, and Martin J. Wainwright. "Fast global convergence of gradient methods for high-dimensional statistical recovery". In: *The Annals of Statistics* (2012).

[10] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., 1993.

[11] N. Ailon and H. Rauhut. “Fast and RIP-optimal transforms”. In: *Preprint* (2013). `http://arxiv.org/abs/1301.0878`.

[12] Nir Ailon and Bernard Chazelle. “The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors”. In: *SIAM Journal on Computing* 39.1 (2009), pp. 302–322.

[13] Nir Ailon and Holger Rauhut. “Fast and RIP-Optimal Transforms”. In: *Discrete & Computational Geometry* 52.4 (2014), pp. 780–798.

[14] Josh Alman, Timothy M Chan, and Ryan Williams. “Polynomial Representations of Threshold Functions and Algorithmic Applications”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2016.

[15] Josh Alman and Ryan Williams. “Probabilistic polynomials and hamming nearest neighbors”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2015.

[16] Alexandr Andoni and Piotr Indyk. “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2006.

[17] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. “Practical and Optimal LSH for Angular Distance”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2015.

[18] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. “Beyond Locality-Sensitive Hashing”. In: *SODA*. Full version at `http://arxiv.org/abs/1306.1547`. 2014.

[19] Alexandr Andoni and Ilya Razenshteyn. “Optimal data-dependent hashing for approximate near neighbors”. In: *STOC*. Full version at `http://arxiv.org/abs/1501.01062`. 2015.

[20] Alexandr Andoni and Ilya Razenshteyn. *Tight Lower Bounds for Data-Dependent Locality-Sensitive Hashing*. Available at `http://arxiv.org/abs/1507.04299`. 2015.

[21] Yossi Arjevani and Ohad Shamir. “Dimension-Free Iteration Complexity of Finite Sum Optimization Problems”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[22] Yossi Arjevani and Ohad Shamir. “Oracle Complexity of Second-Order Methods for Finite-Sum Problems”. In: *CoRR* abs/1611.04982 (2016). URL: `http://arxiv.org/abs/1611.04982`.

[23] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. “ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms”. In: *Similarity Search and Applications*. 2017.

[24] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[25] Francis Bach. “Sharp analysis of low-rank kernel matrix approximations”. In: *Conference on Learning Theory (COLT)*. 2013.

[26] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. “Optimization with Sparsity-Inducing Penalties”. In: *Foundations and Trends in Machine Learning* 4.1 (2012), pp. 1–106.

[27] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. “Structured Sparsity through Convex Optimization”. In: *Statistical Science* 27.4 (Nov. 2012), pp. 450–468.

[28] Francis Bach and Suvrit Sra. “Stochastic optimization: Beyond stochastic gradients and convexity”. In: *NIPS Tutorial* (2016). `http://suvrit.de/talks/vr_nips16_bach.pdf`.

[29] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. “Tight Hardness Results for Maximum Weight Rectangles”. In: *International Colloquium on Automata, Languages, and Programming*. 2016.

[30] Arturs Backurs and Piotr Indyk. “Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)”. In: *Symposium on Theory of Computing (STOC)*. 2015.

[31] Arturs Backurs and Christos Tzamos. “Improving Viterbi is Hard: Better Runtimes Imply Faster Clique Algorithms”. In: *International Conference on Machine Learning (ICML)* (2017).

[32] B. Bah, L. Baldaserre, and V. Cevher. “Model-Based Sketching and Recovery with Expanders”. In: *Proc. ACM-SIAM Symp. Discrete Alg. (SODA)*. 2014.

[33] Luca Baldassarre, Nirav Bhan, Volkan Cevher, Anastasios Kyrillidis, and Siddhartha Satpathi. “Group-Sparse Model Selection: Hardness and Relaxations”. In: *IEEE Transactions on Information Theory* (2016).

[34] R. Baraniuk. “Optimal Tree Approximation with Wavelets”. In: *Proc. SPIE Ann. Meeting: Wavelet Apps. Signal Imag. Proc.* 1999.

[35] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. “A Simple Proof of the Restricted Isometry Property for Random Matrices”. In: *Constructive Approximation* 28.3 (2008), pp. 253–263.

[36] Richard G. Baraniuk, Volkan Cevher, Marco F. Duarte, and Chinmay Hegde. “Model-Based Compressive Sensing”. In: *IEEE Transactions on Information Theory* 56.4 (2010), pp. 1982–2001.

[37] Richard G. Baraniuk and Douglas L. Jones. “A signal-dependent time-frequency representation: Fast algorithm for optimal kernel design”. In: *IEEE Transactions on Signal Processing* (1994).

[38] MohammadHossein Bateni, Chandra Chekuri, Alina Ene, Mohammad T. Hajiaghayi, Nitish Korula, and Daniel Marx. “Prize-collecting Steiner Problems on Planar Graphs”. In: *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2011, pp. 1028–1049.

[39] Stephen Becker, Volkan Cevher, and Anastasios Kyrillidis. "Randomized Low-Memory Singular Value Projection". In: *SampTA (Conference on Sampling Theory and Applications)*. 2013.

[40] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. Society for Industrial and Applied Mathematics, 2001.

[41] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Communications of the ACM* 18.9 (1975), pp. 509–517.

[42] R. Berinde, A. Gilbert, P. Indyk, H. Karloff, and M. Strauss. "Combining geometry and combinatorics: A unified approach to sparse signal recovery". In: *Allerton*. 2008.

[43] Erik Bernhardsson. "ann-benchmarks". 2015. URL: `https://github.com/erikbern/ann-benchmarks`.

[44] Erik Bernhardsson. *Annoy (Approximate Nearest Neighbors Oh Yeah)*. `https://github.com/spotify/annoy`. 2013.

[45] Srinadh Bhojanapalli, Anastasios Kyrillidis, and Sujay Sanghavi. "Dropping Convexity for Faster Semi-definite Optimization". In: *arXiv preprint 1509.03917* (2015).

[46] Wei Bi and James T. Kwok. "Multi-Label Classification on Tree- and DAG-Structured Hierarchies". In: *Proceedings of the 28th International Conference on Machine Learning (ICML)*. 2011, pp. 17–24.

[47] Avrim Blum and Ronald L. Rivest. "Training a 3-node Neural Network is NP-complete". In: *Workshop on Computational Learning Theory (COLT)*. 1988.

[48] T. Blumensath and M. Davies. "Sampling theorems for signals from the union of finite-dimensional linear subspaces". In: *IEEE Transactions on Information Theory* 55.4 (2009), pp. 1872–1882.

[49] Thomas Blumensath. "Sampling and reconstructing signals from a union of linear subspaces". In: *IEEE Transactions on Information Theory* 57.7 (2011), pp. 4660–4671.

[50] Thomas Blumensath and Mike E. Davies. "Iterative hard thresholding for compressive sensing". In: *Applied and Computational Harmonic Analysis* (2009).

[51] Marko Bohanec and Ivan Bratko. "Trading accuracy for simplicity in decision trees". In: *Machine Learning* (1994).

[52] Léonand Bottou and Olivier Bousquet. "The Tradeoffs of Large Scale Learning". In: *Advances in Neural Information Processing Systems (NIPS)*. 2007.

[53] Jean Bourgain. "An Improved Estimate in the Restricted Isometry Problem". In: *Geometric Aspects of Functional Analysis: Israel Seminar (GAFA) 2011-2013*. 2014.

[54] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[55] David Bremner, Timothy M Chan, Erik D Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. "Necklaces, convolutions, and X+ Y". In: *Algorithms–ESA 2006*. Springer, 2006, pp. 160–171.

[56] Karl Bringmann. “Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2014.

[57] Karl Bringmann and Marvin Künnemann. “Quadratic conditional lower bounds for string problems and dynamic time warping”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2015.

[58] E. J. Candes and Y. Plan. “Tight Oracle Inequalities for Low-Rank Matrix Recovery From a Minimal Number of Noisy Random Measurements”. In: *IEEE Transactions on Information Theory* (2011).

[59] Emmanuel J. Candès and Benjamin Recht. “Exact Matrix Completion via Convex Optimization”. In: *Foundations of Computational Mathematics* (2009).

[60] Emmanuel J. Candès, Justin Romberg, and Terence Tao. “Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information”. In: *IEEE Transactions on Information Theory* (2006).

[61] Emmanuel J. Candès and Terence Tao. “Decoding by linear programming”. In: *IEEE Transactions on Information Theory* 51.12 (2005), pp. 4203–4215.

[62] Emmanuel J. Candès and Terence Tao. “Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?” In: *IEEE Transactions on Information Theory* (2006).

[63] Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. “Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility”. In: *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. ACM. 2016, pp. 261–270.

[64] Coralia Cartis and Andrew Thompson. “An Exact Tree Projection Algorithm for Wavelets”. In: *IEEE Signal Processing Letters* (2013).

[65] Nicolo Cesa-Bianchi, Yishay Mansour, and Ohad Shamir. “On the Complexity of Learning with Kernels.” In: *Conference On Learning Theory (COLT)*. 2015.

[66] Volkan Cevher, Marco F. Duarte, Chinmay Hegde, and Richard Baraniuk. “Sparse Signal Recovery Using Markov Random Fields”. In: *Advances in Neural Information Processing Systems 21 (NIPS)*. 2009, pp. 257–264.

[67] Volkan Cevher, Piotr Indyk, Chinmay Hegde, and Richard G. Baraniuk. “Recovery of clustered sparse signals from compressive measurements”. In: *International Conference on Sampling Theory and Applications (SAMPTA)*. 2009.

[68] Timothy M Chan and Moshe Lewenstein. “Clustered integer 3SUM via additive combinatorics”. In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. 2015.

[69] Venkat Chandrasekaran, Benjamin Recht, Pablo A. Parrilo, and Alan S. Willsky. “The Convex Geometry of Linear Inverse Problems”. In: *Foundations of Computational Mathematics* 12.6 (2012), pp. 805–849.

[70] Moses S. Charikar. “Similarity Estimation Techniques from Rounding Algorithms”. In: *STOC*. 2002.

[71] Moses Charikar and Paris Siminelakis. “Hashing-Based-Estimators for Kernel Density in High Dimensions”. In: *FOCS* (2017).

[72] Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert E. Tarjan, and Virginia Vassilevska Williams. “Better approximation algorithms for the graph diameter”. In: *Symposium on Discrete Algorithms (SODA)*. 2014.

[73] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. “Atomic Decomposition by Basis Pursuit”. In: *SIAM Journal Scientific Computing* (1998).

[74] Yudong Chen and Martin J. Wainwright. “Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees”. In: *arXiv preprint arXiv:1509.03025* (2015).

[75] M. Cheraghchi, V. Guruswami, and A. Velingker. “Restricted isometry of Fourier matrices and list decodability of random linear codes”. In: *Symposium on Discrete Algorithms (SODA)*. 2013.

[76] Richard Cole, Ramesh Hariharan, Moshe Lewenstein, and Ely Porat. “A Faster Implementation of the Goemans-Williamson Clustering Algorithm”. In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2001, pp. 17–25.

[77] T. Cormen, C. Stein, R. Rivest, and C. Leiserson. *Introduction to Algorithms*. 2nd. McGraw-Hill Higher Education, 2001.

[78] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. “Synopses for massive data: Samples, histograms, wavelets, sketches”. In: *Foundations and Trends in Databases* 4.1–3 (2012), pp. 1–294.

[79] M. Crouse, R. Nowak, and R. Baraniuk. “Wavelet-based statistical signal processing using Hidden Markov Models”. In: *IEEE Trans. Sig. Proc.* (1998).

[80] Wei Dai and Olgica Milenkovic. “Subspace Pursuit for Compressive Sensing Signal Reconstruction”. In: *IEEE Transactions on Information Theory* (2009).

[81] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. “Fast locality-sensitive hashing”. In: *KDD*. 2011.

[82] M. Davenport, D. Needell, and M. Wakin. “Signal Space CoSaMP for Sparse Recovery with Redundant Dictionaries”. In: *IEEE Transactions on Information Theory* (2013).

[83] Mark Davenport and Justin Romberg. “An overview of low-rank matrix recovery from incomplete observations”. In: *arXiv preprint 1601.06422* (2016).

[84] Roee David, Karthik C. S., and Bundit Laekhanukit. “On the Complexity of Closest Pair via Polar-Pair of Point-Sets”. In: *Symposium on Computational Geometry (SoCG)*. 2018.

[85] B. Dezs, A. Jüttner, and P. Kovács. “LEMON: an Open Source C++ Graph Template Library”. In: *Electron. Notes Theor. Comput. Sci.* 264.5 (2011), pp. 23–45.

[86] Inderjit S. Dhillon, Pradeep K. Ravikumar, and Ambuj Tewari. "Nearest Neighbor based Greedy Coordinate Descent". In: *NIPS*. 2011.

[87] Persi Diaconis and David Freedman. "A dozen de Finetti-style results in search of a theory". In: *Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques* 23.S2 (1987), pp. 397–423.

[88] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. "Lower Bounds for Sparse Recovery". In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2010, pp. 1190–1197.

[89] D. Donoho. "CART and best-ortho-basis: a connection". In: *Annals of Statistics* (1997).

[90] D. Donoho. "Compressed Sensing". In: *IEEE Transactions on Information Theory* (2006).

[91] M. Duarte, S. Sarvotham, D. Baron, M. Wakin, and R. Baraniuk. "Distributed compressed sensing of jointly sparse signals". In: *Proc. Asilomar Conf. Signals, Sys., Comput.* 2005.

[92] Marco F. Duarte and Yonina C. Eldar. "Structured Compressed Sensing: From Theory to Applications". In: *IEEE Transactions on Signal Processing* 59.9 (2011), pp. 4053–4085.

[93] Moshe Dubiner. "Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem". In: *IEEE Transactions on Information Theory* 56.8 (2010), pp. 4166–4179.

[94] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* (2011).

[95] David Eisenstat, Philip Klein, and Claire Mathieu. "An Efficient Polynomial-time Approximation Scheme for Steiner Forest in Planar Graphs". In: *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2012, pp. 626–638.

[96] Marwa El Halabi and Volkan Cevher. "A totally unimodular view of structured sparsity". In: *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2015.

[97] Kave Eshghi and Shyamsundar Rajaram. "Locality sensitive hash functions based on concomitant rank order statistics". In: *KDD*. 2008.

[98] Uriel Feige and Gideon Schechtman. "On the optimality of the random hyperplane rounding technique for MAX CUT". In: *Random Structures and Algorithms* 20.3 (2002), pp. 403–440.

[99] Paulo Feofiloff, Cristina G. Fernandes, Carlos E. Ferreira, and José Coelho de Pina. "A note on Johnson, Minkoff and Phillips' algorithm for the Prize-Collecting Steiner Tree Problem". In: *Computing Research Repository (CoRR)* abs/1004.1437 (2010).

[100] S. Foucart, A. Pajor, H. Rauhut, and T. Ullrich. "The Gelfand widths of ℓ_p-balls for $0 \leq p \leq 1$". In: *J. Complex.* 26.6 (2010), pp. 629–640.

[101] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing.* Springer Birkhäuser, 2013.

[102] Rahul Garg and Rohit Khandekar. "Gradient Descent with Sparsification: An Iterative Algorithm for Sparse Recovery with Restricted Isometry Property". In: *International Conference on Machine Learning (ICML).* 2009. URL: `http://doi.acm.org/10.1145/1553374.1553417`.

[103] A. Garnaev and E. Gluskin. "On widths of the Euclidean ball." English. Russian original. In: *Sov. Math., Dokl.* 30 (1984), pp. 200–204.

[104] A. Gilbert and P. Indyk. "Sparse Recovery Using Sparse Matrices". In: *Proceedings of the IEEE* 98.6 (2010), pp. 937–947.

[105] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. "Fast, Small-Space Algorithms for Approximate Histogram Maintenance". In: *STOC.* 2002.

[106] Anna C. Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin J. Strauss. "Surfing wavelets on streams: One-pass summaries for approximate aggregate queries". In: *VLDB.* Vol. 1. 2001, pp. 79–88.

[107] R. Giryes and M. Elad. "Iterative Hard Thresholding with Near Optimal Projection for Signal Recovery". In: *Intl. Conf. on Sampling Theory and Appl. (SampTA).* 2013.

[108] Raja Giryes and Deanna Needell. "Greedy signal space methods for incoherence and beyond". In: *Applied and Computational Harmonic Analysis* (2015).

[109] E. Gluskin. "Norms of random matrices and widths of finite-dimensional sets." English. In: *Math. USSR, Sb.* 48 (1984), pp. 173–182.

[110] Michel X. Goemans and David P. Williamson. "A General Approximation Technique for Constrained Forest Problems". In: *SIAM Journal on Computing* 24.2 (1995), pp. 296–317.

[111] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour". 2017. URL: `http://arxiv.org/abs/1706.02677`.

[112] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. "Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality". In: *Theory of Computing* (2012).

[113] Moritz Hardt, Benjamin Recht, and Yoram Singer. "Train Faster, Generalize Better: Stability of Stochastic Gradient Descent". In: *International Conference on Machine Learning (ICML).* 2016.

[114] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations.* Chapman & Hall/CRC, 2015.

[115] Ishay Haviv and Oded Regev. "The Restricted Isometry Property of Subsampled Fourier Matrices". In: *Symposium on Discrete Algorithms (SODA).* 2016.

[116] Lihan He and Lawrence Carin. "Exploiting Structure in Wavelet-Based Bayesian Compressive Sensing". In: *IEEE Transactions on Signal Processing* 57.9 (2009), pp. 3488–3497.

[117] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. "A fast approximation algorithm for tree-sparse recovery". In: *IEEE International Symposium on Information Theory (ISIT)*. 2014.

[118] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. "A Nearly-Linear Time Framework for Graph-Structured Sparsity". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. JMLR Workshop and Conference Proceedings, 2015, pp. 928–937.

[119] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. "Approximation Algorithms for Model-Based Compressive Sensing". In: *IEEE Transactions on Information Theory* 61.9 (2015). Conference version appeard in the *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2014), pp. 5129–5147.

[120] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. "Fast Algorithms for Structured Sparsity". In: *Bulletin of EATCS* 3.117 (2015).

[121] Chinmay Hegde, Piotr Indyk, and Ludwig Schmidt. "Nearly Linear-Time Model-Based Compressive Sensing". In: *Automata, Languages, and Programming (ICALP)*. Vol. 8572. Lecture Notes in Computer Science. 2014, pp. 588–599.

[122] Junzhou Huang, Tong Zhang, and Dimitris Metaxas. "Learning with structured sparsity". In: *The Journal of Machine Learning Research* 12 (2011), pp. 3371–3412.

[123] Russell Impagliazzo and Ramamohan Paturi. "On the Complexity of k-SAT". In: *Journal of Computer and System Sciences* 62.2 (2001), pp. 367–375.

[124] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which Problems Have Strongly Exponential Complexity?" In: *Journal of Computer and System Sciences* 63 (2001), pp. 512–530.

[125] P. Indyk and E. Price. "K-median clustering, model-based compressive sensing, and sparse recovery for Earth Mover Distance". In: *Proc. ACM Symp. Theory of Comput.* San Jose, CA, June 2011.

[126] P. Indyk and I. Razenshteyn. "On Model-Based RIP-1 Matrices". In: *Proc. Intl. Coll. Aut., Lang., and Prog. (ICALP)*. 2013.

[127] Piotr Indyk and Rajeev Motwani. "Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality". In: *STOC*. 1998.

[128] Yannis Ioannidis. "The history of histograms (abridged)". In: *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment. 2003, pp. 19–30.

[129] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *ICML*. 2015.

[130] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. "Group Lasso with Overlap and Graph Lasso". In: *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. 2009, pp. 433–440.

[131] Prateek Jain, Raghu Meka, and Inderjit S. Dhillon. "Guaranteed rank minimization via singular value projection". In: *NIPS*. 2010.

[132] Prateek Jain, Nikhil Rao, and Inderjit S Dhillon. "Structured Sparse Regression via Greedy Hard Thresholding". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[133] Prateek Jain, Ambuj Tewari, and Purushottam Kar. "On Iterative Hard Thresholding Methods for High-dimensional M-Estimation". In: *Advances in Neural Information Processing Systems (NIPS)*. 2014.

[134] Herve Jegou, Matthijs Douze, and Cordelia Schmid. "Product Quantization for Nearest Neighbor Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2011).

[135] David S. Johnson, Maria Minkoff, and Steven Phillips. "The Prize Collecting Steiner Tree Problem: Theory and Practice". In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2000, pp. 760–769.

[136] Michael Jünger and William R. Pulleyblank. "New primal and dual matching heuristics". In: *Algorithmica* 13.4 (1995), pp. 357–380.

[137] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. The IBM Research Symposia Series. 1972, pp. 85–103.

[138] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. "Large-scale Video Classification with Convolutional Neural Networks". In: *CVPR*. 2014.

[139] B. Kasin. "Diameters of some finite-dimensional sets and classes of smooth functions." In: *Math. USSR, Izv.* 11 (1977), pp. 317–333.

[140] Seyoung Kim and Eric P. Xing. "Tree-Guided Group Lasso for Multi-Task Regression with Structured Sparsity". In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. 2010, pp. 543–550.

[141] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference for Learning Representations (ICLR)*. 2014.

[142] Felix Krahmer and Rachel Ward. "New and Improved Johnson–Lindenstrauss Embeddings via the Restricted Isometry Property". In: *SIAM Journal on Mathematical Analysis* (2011).

[143] Andreas Krause and Daniel Golovin. "Submodular Function Maximization". In: *Tractability: Practical Approaches to Hard Problems*. Ed. by Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli. Cambridge University Press, 2014, pp. 71–104.

[144] A. Kyrillidis and V. Cevher. "Combinatorial Selection and Least Absolute Shrinkage via the CLASH Algorithm". In: *IEEE Int. Symp. Inform. Theory (ISIT)*. Cambridge, MA, 2012.

[145] A. Kyrillidis and V. Cevher. "Sublinear time, approximate model-based sparse recovery for all". In: *arXiv:1203.4746* (2012).

[146] Thijs Laarhoven. “Sieving for Shortest Vectors in Lattices Using Angular Locality-Sensitive Hashing”. In: *CRYPTO*. 2015.

[147] Rasmus M. Larsen. *PROPACK*. `http://sun.stanford.edu/~rmunk/PROPACK/`.

[148] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. `http://yann.lecun.com/exdb/mnist/`. 2010.

[149] K. Lee and Y. Bresler. “Admira: Atomic decomposition for minimum rank approximation”. In: *IEEE Transactions on Information Theory* 56.9 (2010), pp. 4402–4416.

[150] E. Levina and P. Bickel. “The Earth Mover’s distance is the Mallows distance: some insights from statistics”. In: *Proc. IEEE Intl. Conf. Comp. Vision (ICCV)*. 2001.

[151] Moshe Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[152] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. “SphereFace: Deep Hypersphere Embedding for Face Recognition”. In: *arXiv preprint arXiv:1704.08063* (2017).

[153] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. “Large-margin Softmax Loss for Convolutional Neural Networks”. In: *ICML*. 2016.

[154] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. “Deep Learning Face Attributes in the Wild”. In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[155] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. “On the computational efficiency of training neural networks”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 855–863.

[156] D Lozovanu and A Zelikovsky. “Minimal and bounded tree problems”. In: *Tezele Congresului XVIII al Academiei Romano-Americane* (1993), pp. 25–26.

[157] M. Lustig, D. Donoho, and J. Pauly. “Sparse MRI: The application of compressed sensing for rapid MR imaging.” In: *Magnetic Resonance in Medicine* (2007).

[158] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. “Multi-probe LSH: efficient indexing for high-dimensional similarity search”. In: *VLDB*. 2007.

[159] Yury A. Malkov and D. A. Yashunin. “Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs”. In: (2016). URL: `http://arxiv.org/abs/1603.09320`.

[160] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1999.

[161] Sofia Mosci, Silvia Villa, Alessandro Verri, and Lorenzo Rosasco. “A Primal-Dual Algorithm for Group Sparse Regularization with Overlapping Groups”. In: *Advances in Neural Information Processing Systems 23 (NIPS)*. 2010, pp. 2604–2612.

[162] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. “Lower bounds on locality sensitive hashing”. In: *SIAM Journal on Discrete Mathematics* 21.4 (2007), pp. 930–935.

[163] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. "An introduction to kernel-based learning algorithms". In: *IEEE transactions on neural networks* 12.2 (2001), pp. 181–201.

[164] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

[165] Cameron Musco and Christopher Musco. "Randomized Block KrylovMethods for Stronger and Faster Approximate Singular Value Decomposition". In: *NIPS*. 2015.

[166] Cameron Musco and Christopher Musco. "Recursive sampling for the Nyström method". In: *Advances in Neural Information Processing Systems (NIPS)* (2016).

[167] S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. "On rectangular partitionings in two dimensions: Algorithms, complexity and applications". In: *ICDT*. 1999, pp. 236–256.

[168] B. K. Natarajan. "Sparse Approximate Solutions to Linear Systems". In: *SIAM Journal on Computing* (1995).

[169] Jesper Nederlof, Erik Jan van Leeuwen, and Ruben van der Zwaan. "Reducing a target interval to a few exact queries". In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 2012, pp. 718–727.

[170] Deanna Needell and Joel A. Tropp. "CoSaMP: Iterative signal recovery from incomplete and inaccurate samples". In: *Applied and Computational Harmonic Analysis* 26.3 (2009), pp. 301–321.

[171] Sahand N. Negahban, Pradeep Ravikumar, Martin J. Wainwright, and Bin Yu. "A Unified Framework for High-Dimensional Analysis of M-Estimators with Decomposable Regularizers". In: *Statistical Science* 27.4 (Nov. 2012), pp. 538–557.

[172] J. Nelson, E. Price, and M. Wootters. "New constructions of RIP matrices with fast multiplication and fewer rows". In: *Symposium on Discrete Algorithms (SODA)*. 2014.

[173] Arkadii S. Nemirovski and David B. Yudin. *Problem Complexity and Method Efficiency in Optimization.* Wiley Interscience, 1983.

[174] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Kluwer Academic Publishers, 2004.

[175] Jorge Nocedal and Stephen Wright. *Numerical Optimization.* Springer, 2006.

[176] Ryan O'Donnell, Yi Wu, and Yuan Zhou. "Optimal lower bounds for locality-sensitive hashing (except when q is tiny)". In: *ACM Transactions on Computation Theory* 6.1 (2014), p. 5.

[177] Samet Oymak, Benjamin Recht, and Mahdi Soltanolkotabi. "Sharp Time-Data Tradeoffs for Linear Inverse Problems". In: *IEEE Transactions on Information Theory* (2017).

[178] Julianus Pfeuffer and Oliver Serang. "A bounded p-norm approximation of max-convolution for sub-quadratic Bayesian Inference on additive factors". In: *Journal of Machine Learning Research* 17.36 (2016), pp. 1–39.

[179] Ali Rahimi and Benjamin Recht. "Random Features for Large-Scale Kernel Machines". In: *Advances in Neural Information Processing Systems (NIPS)*. 2008.

[180] Alexander Rakhlin, Ohad Shamir, and Karthik Sridharan. "Making Gradient Descent Optimal for Strongly Convex Stochastic Optimization". In: *International Conference on Machine Learning (ICML)*. 2012.

[181] Nikhil S. Rao, Ben Recht, and Robert D. Nowak. "Universal Measurement Bounds for Structured Sparse Signal Recovery." In: *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 22. JMLR Proceedings. 2012, pp. 942–950.

[182] H. Rauhut, K. Schnass, and P. Vandergheynst. "Compressed sensing and redundant dictionaries". In: *IEEE Trans. Inform. Theory* (2008).

[183] Ramamurthy Ravi, Ravi Sundaram, Madhav V Marathe, Daniel J Rosenkrantz, and Sekharipuram S Ravi. "Spanning trees-short or small". In: *SIAM Journal on Discrete Mathematics* 9.2 (1996), pp. 178–200.

[184] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". In: *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2014.

[185] Ilya Razenshteyn and Ludwig Schmidt. *FALCONN (FAst Lookups of Cosine and Other Nearest Neighbors)*. `https://github.com/FALCONN-LIB/FALCONN`. 2015.

[186] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. "Guaranteed Minimum-Rank Solutions of Linear Matrix Equations via Nuclear Norm Minimization". In: *SIAM Review* (2010).

[187] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. "On the Convergence of Adam and Beyond". In: *International Conference for Learning Representations (ICLR)*. 2018.

[188] H. Robbins and S. Monro. "A stochastic approximation method". In: *Annals of Mathematical Statistics* (1951).

[189] Liam Roditty and Virginia Vassilevska Williams. "Fast approximation algorithms for the diameter and radius of sparse graphs". In: *Symposium on Theory of Computing (STOC)*. 2013.

[190] Liam Roditty and Uri Zwick. "On dynamic shortest paths problems". In: *Algorithms–ESA 2004*. Springer, 2004, pp. 580–591.

[191] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. "Event Detection in Activity Networks". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2014, pp. 1176–1185.

[192] Aviad Rubinstein. "Hardness of Approximate Nearest Neighbor Search". In: *STOC*. 2018.

[193] Y. Rubner, C. Tomasi, and L. J. Guibas. "The Earth Mover's Distance as a Metric for Image Retrieval". In: *International Journal of Computer Vision* (2000).

[194] Mark Rudelson and Roman Vershynin. "On sparse reconstruction from Fourier and Gaussian measurements". In: *Communications on Pure and Applied Mathematics* (2007). URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.20227`.

[195] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* (1986).

[196] Yousef Saad. "On the Rates of Convergence of the Lanczos and the Block-Lanczos Methods". In: *SIAM Journal on Numerical Analysis* 17.5 (1980), pp. 687–706.

[197] Tim Salimans and Diederik P Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 901–901.

[198] Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[199] L. Schmidt, C. Hegde, and P. Indyk. "The Constrained Earth Mover Distance Model, with Applications to Compressive Sensing". In: *Intl. Conf. on Sampling Theory and Appl. (SampTA)*. 2013.

[200] L. Schmidt, C. Hegde, P. Indyk, J. Kane, L. Lu, and D. Hohl. "Automatic Fault Localization Using the Generalized Earth Movers Distance". In: *ICASSP*. 2014.

[201] Ludwig Schmidt, Chinmay Hegde, Piotr Indyk, Ligang Lu, Xingang Chi, and Detlef Hohl. "Seismic Feature Extraction Using Steiner Tree Methods". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015.

[202] Ludwig Schmidt, Matthew Sharifi, and Ignacio Lopez Moreno. "Large-scale speaker identification". In: *ICASSP*. 2014.

[203] Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[204] Oliver Serang. "Fast Computation on Semirings Isomorphic to $(\times, \max)$ on $\mathbb{R}_+$". In: *arXiv preprint arXiv:1511.05690* (2015).

[205] Parikshit Shah and Venkat Chandrasekaran. "Iterative projections for signal identification on manifolds: Global recovery guarantees". In: *Allerton Conference on Communication, Control, and Computing (Allerton)* (2011).

[206] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. Cambridge, MA: MIT Press, 2005.

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

[208] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. "Pegasos: Primal estimated sub-gradient solver for SVM". In: *International Conference on Machine Learning (ICML)*. 2007.

[209] Jerome M Shapiro. "Embedded image coding using zerotrees of wavelet coefficients". In: *IEEE Trans. Sig. Proc.* (1993).

[210] Anshumali Shrivastava and Ping Li. "Densifying one permutation hashing via rotation for fast near neighbor search". In: *ICML*. 2014.

[211] Anshumali Shrivastava and Ping Li. “Fast near neighbor search in high-dimensional binary data”. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 474–489.

[212] Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. “A Sparse-Group Lasso”. In: *Journal of Computational and Graphical Statistics* 22.2 (2013), pp. 231–245.

[213] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: `http://arxiv.org/abs/1409.1556`.

[214] Malcolm Slaney, Yury Lifshits, and Junfeng He. “Optimal parameters for locality-sensitive hashing”. In: *Proceedings of the IEEE* 100.9 (2012), pp. 2604–2623.

[215] Daniel D. Sleator and Robert E. Tarjan. “A Data Structure for Dynamic Trees”. In: *Journal of Computer and System Sciences* 26.3 (1983), pp. 362–391.

[216] Leslie N. Smith and Nicholay Topin. “Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates”. `http://arxiv.org/abs/1708.07120`. 2017.

[217] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. “Streaming similarity search over one billion tweets using parallel locality-sensitive hashing”. In: *VLDB*. 2013.

[218] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going Deeper with Convolutions”. In: *CVPR*. 2015.

[219] Kengo Terasawa and Yuzuru Tanaka. “Spherical lsh for approximate nearest neighbor search on unit hypersphere”. In: *Algorithms and Data Structures*. Springer, 2007, pp. 27–38.

[220] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. “Dynamic multidimensional histograms”. In: *SIGMOD*. 2002.

[221] Robert Tibshirani. “Regression Shrinkage and Selection Via the Lasso”. In: *Journal of the Royal Statistical Society, Series B* (1994).

[222] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. “Sparsity and smoothness via the fused lasso”. In: *Journal of the Royal Statistical Society Series B* (2005).

[223] Stephen Tu, Ross Boczar, Max Simchowitz, Mahdi Soltanolkotabi, and Benjamin Recht. “Low-rank solutions of linear matrix equations via Procrustes Flow”. In: *ICML*. 2016.

[224] Gregory Valiant. “Finding correlations in subquadratic time, with applications to learning parities and juntas”. In: *Symposium on Foundations of Computer Science (FOCS)*. 2012.

[225] Vlamimir Vapnik. *Statistical learning theory*. Wiley, 1998.

[226] Virginia Vassilevska Williams. “Hardness of easy problems: Basing hardness on popular conjectures such as the Strong Exponential Time Hypothesis (invited talk)”. In: *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 43. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2015.

[227] N. Vaswani and W. Lu. “Modified-CS: Modifying Compressive Sensing for Problems With Partially Known Support”. In: *IEEE Trans. Sig. Proc.* 58.9 (2010), pp. 4595–4607.

[228] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[229] Sudheendra Vijayanarasimhan, Jonathon Shlens, Rajat Monga, and Jay Yagnik. “Deep Networks With Large Output Spaces”. In: *ICLR*. 2015.

[230] Andrew Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *IEEE transactions on Information Theory* 13.2 (1967), pp. 260–269.

[231] Martin J. Wainwright. *High-dimensional statistics: a non-asymptotic viewpoint*. Cambridge University Press, 2018.

[232] Martin J. Wainwright. “Structured Regularizers for High-Dimensional Problems: Statistical and Computational Issues”. In: *Annual Review of Statistics and Its Application* 1.1 (2014), pp. 233–253.

[233] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. “Feature hashing for large scale multitask learning”. In: *ICML*. 2009.

[234] Christopher K. Williams and Matthias Seeger. “Using the Nyström Method to Speed Up Kernel Machines”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2001.

[235] R. Ryan Williams. “On the Difference Between Closest, Furthest, and Orthogonal Pairs: Nearly-Linear vs Barely-Subquadratic Complexity in Computational Geometry”. In: *Symposium on Discrete Algorithms (SODA)*. 2018.

[236] Virginia Vassilevska Williams and Ryan Williams. “Finding, minimizing, and counting weighted subgraphs”. In: *SIAM Journal on Computing* 42.3 (2013), pp. 831–854.

[237] Virginia Vassilevska Williams and Ryan Williams. “Subcubic equivalences between path, matrix and triangle problems”. In: *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*. IEEE. 2010, pp. 645–654.

[238] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[239] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. “The Marginal Value of Adaptive Gradient Methods in Machine Learning”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[240] Blake E. Woodworth and Nathan Srebro. “Tight complexity bounds for optimizing composite objectives”. In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[241] Yun Yang, Mert Pilanci, Martin J Wainwright, et al. "Randomized sketches for kernels: Fast and optimal nonparametric regression". In: *The Annals of Statistics* 45.3 (2017), pp. 991–1023.

[242] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.

[243] Tuo Zhao, Zhaoran Wang, and Han Liu. *A Nonconvex Optimization Framework for Low Rank Matrix Estimation.* 2015.

[244] Qinqing Zheng and John Lafferty. "A Convergent Gradient Descent Algorithm for Rank Minimization and Semidefinite Programming from Random Linear Measurements". In: *NIPS*. 2015.

[245] Uri Zwick. "All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms". In: *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on.* IEEE. 1998, pp. 310–319.