universität wien

DISSERTATION | DOCTORAL THESIS

Titel | Title

Efficient Algorithms for Problems in Clustering and Fairness

verfasst von | submitted by

Maximilian Vötsch BSc MSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr.techn.)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 786 880
Dissertationsgebiet lt. Studienblatt | Field of study as it appears on the student record sheet:	Informatik
Betreut von | Supervisor:	Univ.-Prof. Dr. Monika Henzinger Ass.-Prof. Dr. Kathrin Hanauer B.Sc. M.SC.

Abstract

Driven by the developments of modern machine learning and AI systems, processing and analyzing large datasets has become one of the most important tasks in computer science. The volume and diversity of datasets, from static vector-valued sets to graphs to live data streams, presents numerous challenges from both a theoretical and practical point of view. For example, large datasets usually do not fit into a single computer's memory, contrary to a fundamental theoretical assumption of the classical random access machine model. To model these challenges, a number of new models, such as the streaming, distributed, or online models, were developed. Moreover, data reduction techniques, such as data compression and clustering, have moved to the forefront of application-minded algorithms design.

In this thesis we focus on two important concepts in the study of algorithms for analyzing and reducing large datasets and study them from both theoretical and experimental viewpoints:

The first is clustering, one of the fundamental data analysis tasks. Being able to group similar data points is of crucial importance for many applications. It allows us to compress data, for example via coresets or vector databases, or to find patterns in the data, as well as to identify representative elements in the data.

The other aspect is of a socio-technical nature. As we are often processing datasets containing sensitive personal data and use them in automated decision processes, the lives of people are increasingly governed by digital systems. Therefore, we need to ensure that no user is discriminated against unfairly. Many examples of automated decision making exhibiting racial-, gender- or other biases have been observed, which shows that a focus on algorithmic fairness is tantamount.

Motivated by these two topics, the main results of this thesis are:

- Introduction of the fair paging problem, a natural extension of the well-studied online paging problem under a perspective of algorithmic fairness. We show that the fair problem is strictly harder than previously studied versions of the paging problem, and we introduce the first known upper and lower bounds for this problem.
- The current best-known algorithms under several metrics for the binary matrix factorization problem, where the goal is to find a low-rank approximation of a binary matrix. Additionally, we introduce the first algorithms for this problem in the streaming and distributed settings. We employ an equivalence to k-means clustering, as well as sketching and enumeration to obtain algorithms running in time singly exponential in the rank of the output matrix k. We also experimentally evaluate several algorithms for this problem and show that our algorithmic ideas may be used to obtain better solutions for non-sparse matrices in practice.

- The first algorithm with provable guarantees for the k-means problem which runs in time near linear in the number of input vectors while exhibiting no dependence on the number of clusters k. For sparse matrices it additionally exhibits no running time dependence on the dimensionality of the dataset. We show that aggressive dimensionality reduction to a single dimension yields an algorithm with polynomial approximation ratio that is both theoretically and practically feasible. We also thoroughly evaluate the algorithm experimentally and show that it produces large speedups over baseline algorithms for clustering and coreset construction.
- The first practical graph clustering algorithm applying expander decomposition and the expander hierarchy in practice. Using a random walk-based expander decomposition algorithm, we engineer a novel state of the art algorithm for sparsifying and solving the normalized cut problem on large graphs, and we perform extensive experiments demonstrating its competitive performance.

Kurzfassung

Basierend auf Trends hin zu immer größeren Datenmengen zum Training moderner Machine Learning- und KI-Systeme, ist die Verarbeitung und Analyse großer Datensätze zu einer der wichtigsten Aufgaben in der Informatik geworden. Dabei führen der Umfang und die Vielfalt der Datensätze – von statischen vektorwertigen Mengen, über Graphen, bis hin zu Live-Daten – sowohl aus theoretischer als auch aus praktischer Sicht zu zahlreichen Herausforderungen. Beispielsweise passen derart große Datensätze in der Regel nicht in den Hauptspeicher eines einzelnen Computers, was einer grundlegenden theoretischen Annahme des klassischen Wort-RAM Maschinenmodells widerspricht. Um diesen Herausforderungen gerecht zu werden, wurde eine Reihe neuer algorithmischer Modelle entwickelt, wie z.B. das Streaming-, das Distributed Computing- oder das Online-Modell. Darüber hinaus sind Techniken zur Datenreduzierung, wie zum Beispiel Datenkomprimierung und Clustering, in den Vordergrund der anwendungsorientierten Algorithmenforschung gerückt.

In dieser Arbeit konzentrieren wir uns auf zwei wichtige Konzepte im Design von Algorithmen zur Analyse und Reduzierung großer Datenmengen und untersuchen sie sowohl aus theoretischer als auch aus experimenteller Sicht:

Das erste Konzept bezieht sich auf Clustering, eine der grundlegenden Aufgaben der Datenanalyse. Die Fähigkeit, ähnliche Datenpunkte zu gruppieren, ist für viele Anwendungen von entscheidender Bedeutung. Sie ermöglicht es uns, Daten zu komprimieren, z.B. über Coresets oder Vektordatenbanken, Muster in den Daten zu finden oder repräsentative Elemente in den Daten zu identifizieren.

Der andere Aspekt ist sozio-technischer Art. Viele der Datensätze, die wir analysieren enthalten sensible personenbezogene Daten, welche dann in automatisierten Prozessen als Entscheidungsgrundlage dienen. Da solche Systeme zunehmenden Einfluß auf das alltägliche Leben haben, müssen wir sicherstellen können, dass keinem Nutzer ungerechtfertigte Diskrimination widerfährt. In den letzten Jahren gab es eine Reihe an Beispielen wo KI-basierte Entscheidungsprozesse rassistische, sexistische oder klassistische Entscheidungen trafen, was zeigt, dass der Bedarf für faire Systeme in der Praxis gegeben ist.

Motiviert durch diese beiden Themen sind die wichtigsten Ergebnisse dieser Arbeit:

- Einführung des fairen Paging-Problems, einer natürlichen Erweiterung des klassischen Online-Paging-Problems unter dem Aspekt der algorithmischen Fairness. Wir zeigen, dass das faire Problem strikt schwieriger ist als die zuvor untersuchten Versionen des Paging-Problems, und wir stellen die ersten bekannten oberen und unteren Schranken für dieses Problem vor.

- Die besten bekannten Algorithmen für das binäre Matrixfaktorisierungsproblem unter verschiedenen Metriken. Das Ziel dieses Problems besteht darin eine niedrig-

rangige Approximation einer binärwertigen Matrix zu finden. Zusätzlich stellen wir die ersten Algorithmen für dieses Problem im Streaming- und Distributed-Modell vor. Wir verwenden eine Äquivalenz zu k-means Clustering sowie Sketching und Enumeration, um Algorithmen zu entwickeln, die in einfach exponentieller Zeit zum Rang k der Outputmatrix laufen. Auch evaluieren wir mehrere praktische Algorithmen für dieses Problem auf experimentelle Weise und zeigen, dass unsere algorithmischen Ideen verwendet werden können, um bessere praktische Lösungen für dichte Matrizen zu erhalten.

- Der erste Algorithmus mit beweisbaren Garantien für das k-Means Problem, der in Zeit nahezu linear zur Anzahl der Vektoren im Datensatz läuft, und keine Abhängigkeit von der Anzahl der Cluster k aufweist. Für dünnbesetzte Matrizen zeigen wir außerdem, dass keine Laufzeitabhängigkeit von der Dimensionalität des Datensatzes besteht. Wir zeigen, dass eine aggressive Dimensionalitätsreduktion auf eine einzige Dimension zu einem Algorithmus mit polynomialem Approximationsverhältnis führt, der sowohl theoretische als auch praktische Vorteile aufweist. Weiters evaluieren wir den Algorithmus experimentell und zeigen, dass er im Vergleich zu Basisalgorithmen für Clustering und Coreset-Konstruktion große Geschwindigkeitsvorteile bietet.

- Der erste praktische Graph-Clustering-Algorithmus, der Expander-Zerlegung und die Expander-Hierarchie in der Praxis anwendet. Unter Verwendung eines auf Random Walks basierenden Expander-Zerlegungsalgorithmus entwickeln wir einen Algorithmus zur Sparsifizierung und Lösung des Normalized Cut Problems auf großen Graphen und führen umfangreiche Experimente durch, die seine konkurrenzfähige Leistung zeigen.

Acknowledgements

First, I would like to thank my advisors, Monika Henzinger and Kathrin Hanauer. Without either of you, this thesis would not have taken shape the way it has, and my doctoral studies would not have been nearly as fruitful nor as interesting as they turned out to be. Thank you for the last four years at the University of Vienna and the TAA group, in particular, for fostering amazing international collaboration opportunities and supporting my research interests. Thank you for all your time, care, and energy!

A special thank you to Stefan Neumann and Aristides Gionis for agreeing to serve as reviewers for my thesis. I could not have asked for a better committee.

Furthermore, I would like to thank all of my coauthors for the wonderful opportunities to work with them and gain deep, multifaceted insights into scientific work from all the knowledge and unique perspectives they have shared with me. Thank you to Ashish Chiplunkar, Sagar Kale, Ameya Velingker, David P. Woodruff, Samson Zhou, Moses Charikar, Lunjia Hu, Erik Waingarten, Robin Münk, and Harald Räcke.

The Theory and Applications of Algorithms group (TAA) has provided a fantastic working environment, even though, starting my studies in early 2021, it took a while until we all became acquainted. I thoroughly enjoyed sharing an office with Sricharan AR and Antoine El-Hayek. Further thanks to all my other colleagues, Hendrik Fichtenberger, Lara Ost, Alexander Noe, Alexander Svozil, Ami Paz, Gramoz Goranci, Teresa Anna Steiner, Martin Schirneck, Martin Seybold, Sophia Heck, Éva Szilágyi, and Ali Momeni, for all the lunches, talks, discussions and guidance.

I would also like to thank Pranav, Arjun, Ashwin, Alissa, Amanda, Vishal, Willie, Alaka, and Michelle, along with all the other volunteers and organizers working with and advocating for Queer in AI to provide a venue for critical, interdisciplinary research into AI, as well as a space that I can only describe as a home-away-from-home at conferences.

I am also beyond grateful to my parents, who have been nothing but supportive of my academic endeavors. Thank you, Mom and Dad.

My sincerest thanks to all my friends who kept me grounded throughout the past four years, particularly Jonathan, Patricia, Anna, Ursi, Piro, Amalia, Stefan, Juliana, Hati, and Markus.

I also want to sincerely thank my cats, Lucy and Panko, for all the emotional support they have provided me, even though they have no idea what a computer or science may be, nor will they ever read this paragraph.

Finally, I want to thank min and Anicca for all their love and support. In particular, I want to thank them for always being there for me and keeping me motivated, even when it all seemed too much. Also, I want to thank them for their time and effort in providing invaluable feedback on my drafts and for letting me bounce my ideas off them.

Bibliographic Note

Several results of this thesis have already been published in conference proceedings. The following list indicates the chapters and the materials that they are based on:

- Chapter 3: Ashish Chiplunkar, Monika Henzinger, Sagar Sudhir Kale, and Maximilian Vötsch. Online min-max paging. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1545–1565. SIAM, 2023.

- Chapter 4: Ameya Velingker, Maximilian Vötsch, David P. Woodruff, and Samson Zhou. Fast $(1+\varepsilon)$-approximation algorithms for binary matrix factorization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA, volume 202 of Proceedings of Machine Learning Research*, pages 34952–34977. PMLR, 2023.

- Chapter 5: Moses Charikar, Monika Henzinger, Lunjia Hu, Maximilian Vötsch, and Erik Waingarten. Simple, scalable and effective clustering via one-dimensional projections. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

- Chapter 6: Kathrin Hanauer, Monika Henzinger, Robin Münk, Harald Räcke, and Maximilian Vötsch. Expander hierarchies for normalized cuts on graphs. In Ricardo Baeza-Yates and Francesco Bonchi, editors, *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pages 1016–1027. ACM, 2024.

Contents

1 Introduction

1.1 Motivation and Methodology

Driven by the advances in artificial intelligence, one of the main challenges shaping research in computer science in the last two decades has been the need to perform computations on enormous datasets effectively. Modern artificial intelligence and machine learning approaches hinge on the ability to quickly and effectively perform computational tasks such as gradient descent, regression, or clustering on massive datasets.

The increased adoption of networked devices and the coinciding growth of internet traffic mean that monthly global traffic has grown from 15 EB in 2000 to 42000 EB in 2014.[1] This increase in data and a coinciding increase in data collection has led to diverse and large datasets becoming available for commercial and scientific use. Adding further complexity, these datasets can take many shapes, such as points in Euclidean vector spaces (e.g., multivariate measurements), graph data (social networks or citation graphs), spatial data, or time series data, among many others.

On the other hand, many of the recent advancements in the fields of machine learning and AI have become possible precisely due to this availability of large datasets, which can be used for training and as benchmarks for evaluating models [146, 303, 255] (e.g., the text corpora used for natural language models [315] or the labeled image and video datasets for computer vision [104]). This lets us identify the efficient processing of large datasets as a fruitful area of research with an extensive array of downstream applications.

Most of the public interest in advances in data science and artificial intelligence has been directed at these downstream applications, namely the final machine learning models, such as chatbots and diffusion models for image and video generation. However, these advances rely not only on advanced hardware acceleration techniques and faster optimization algorithms for supervised learning objectives but also on efficiently performing several crucial algorithmic preprocessing tasks during various stages of the training pipelines, such as pre-training and feature engineering. Among these algorithmic tasks, we want to highlight clustering, widely used for quantization and data compression in vector databases [54], as well as data reduction techniques such as dimensionality reduction [170], which are most relevant for this thesis.

These tasks, often grouped under the umbrella term of *unsupervised learning*, are some of the classic algorithmic problems in the field of artificial intelligence and have served as active areas of study for algorithmic research [2]. In this thesis, we continue the algorithmic

[1] According to figures published by Cisco. `https://blogs.cisco.com/sp/the-history-and-future-of-internet-traffic`

[2] They have been studied by algorithms researchers for almost 70 years [301].

investigation of unsupervised learning methods, particularly clustering and data reduction. In doing so, we place particular importance on the scalability of our algorithms.

One of the main approaches for improving the scalability of the algorithms developed in this thesis is the frequent use of data reduction and compression techniques. Intuitively, suppose we can reduce the size of the input dataset while preserving its structure. In that case, it becomes feasible to use slower, classical algorithms while still being able to solve the problem on large datasets within a reasonable amount of time while also using less space. Consequently, we investigate and design algorithms that produce a compressed (or subsampled) instance of the dataset while preserving its relevant structural properties. Countless data reduction techniques exist, with the main ones relevant to this thesis being dimensionality reduction, low-rank approximation, (linear) sketching, and sampling methods, such as coresets and sparsification. See Sections 2.2 and 2.3 for an overview.

We use these data reduction techniques to design algorithms for one of the central problems in machine learning: *clustering*. Given a dataset, a clustering of the data defines some grouping of the individual records, such that the records within the same group are ideally more similar to each other than those from different groups [324]. If we can accomplish this, then a clustering will often provide insights into the data, such as letting us discover hidden commonalities between items or finding the most representative items that summarize the makeup of the data.

With clustering being one of the most natural and fundamental tasks in unsupervised machine learning, clustering algorithms are employed across numerous domains. Clustering methods are used for image processing [63, 106], genetic analysis [31], feature engineering [297], vector databases [172], medical image analysis [245], clustering search results in a text based document system [332, 58], recommender systems [47], and social network analysis [304], among many other problems.

As previously stated, the need to process large datasets has had an outsized influence on algorithmic research's direction in the last two decades. Once datasets grow beyond the scale easily handled by a desktop computer, many new and exciting challenges arise, many of which run counter to the fundamental assumptions made in the standard model of algorithms research: the sequential, random access memory machine model. In particular, one can not generally expect the performance of an algorithm designed in the word-RAM model to scale strictly in accordance with its asymptotic guarantees.

As algorithms in applications become memory-bound rather than CPU-bound, we can no longer expect that datasets fit into the main memory of the computer carrying out the data analysis tasks. More precisely, the word-RAM model assumes unlimited memory size with constant ($\mathcal{O}(1)$) access time, and it does not model any memory hierarchies of fast and slow memory, which are abundant in modern hardware. Furthermore, in this model, algorithms are primarily evaluated in terms of their asymptotic running time and accuracy. In many cases, memory usage is not considered an objective needing optimization.

These facts make it difficult to forecast an algorithm's behavior as it transitions from being CPU-bound to being memory-bound. In many cases, this is due to inefficient memory access patterns, which lead to inefficient cache usage becoming the primary resource cost in practice, as most of the running time is spent waiting for data to be

fetched from slower memory[3]. Another commonly encountered issue is that the constants hidden by the big-O notation can be prohibitively large, leading to the asymptotic running time advantage of many modern algorithms not translating well into practice, leaving these improvements to be only of a theoretical nature [4].

In our research, we employ two main methods to deal with the shortcomings of the standard model we identified above. For one, we employ a number of different models of computation that are more tailored toward designing algorithms dealing with big data. Additionally, we employ the methodology of *algorithm engineering* [276], which seeks to design effective, scalable algorithms using an experimentally guided algorithm design process.

Furthermore, we heavily use randomization for all the algorithms we design, a common feature of modern large data algorithms. In fact, for many problems, randomization is the only way to achieve sublinear guarantees [272]. This is particularly true for many streaming or online problems [225, 230, 100]. Randomization can also be used as a way to hedge against different input sequences, making an algorithm more robust against adversarially chosen input sequences [225, 184, 155]. In many cases, randomized algorithms provide a favorable trade-off, as accepting a small failure probability, in turn, lets us achieve exponentially better theoretical guarantees, which is the property of randomized algorithms that we are most interested in when designing scalable algorithms.

Algorithm engineering, as a methodology of experimental algorithm design, comprises an iterative design process comprised of four steps: design, theoretical analysis, implementation, and experimentation using real-world instances derived from the possible fields of application [276]. The methodology expects us to iterate these four steps repeatedly, and we use the insight gained from experimentally evaluating our algorithms to further refine their design and ultimately converge to an algorithm that performs well in both theory and practice. The asymptotic analysis of algorithms remains a valuable tool in this methodology, and we use it to establish bounds on our algorithms' running time and approximation guarantees. However, algorithm engineering emphasizes the importance of implementability, opening the black box and optimizing the hidden constants in the big-O notation, as well as measuring and improving the performance on real-world instances. When not explicitly engaging in experimentation-first algorithm design, these aspects usually lie outside the scope of modern algorithm research. As algorithm engineering necessitates implementation of the algorithms and for the sake of producing open and reproducible research, our algorithms are made available as open-source libraries, and we have conducted extensive experimental evaluations using publicly available datasets. For this, see the respective sections in Chapters 4 to 6.

While we mainly focus on developing practical and scalable algorithms for unsupervised learning problems, we also identify and study a secondary aspect of algorithms concerned with large-scale data analysis in this thesis.

[3] Accessing data in the L1 cache is about 100 times faster than accessing the main memory, while an SSD read is about 10000 times slower. See `https://colin-scott.github.io/personal_website/research/interactive_latency.html` for details.

[4] See Chapter 6 for an example of this problem.

As mentioned at the beginning of this introduction, one of the main driving forces in the practical successes of machine-learning-based methods has been increased data collection and availability, allowing researchers and engineers to build systems that can learn statistical patterns from vast swathes of data. The availability of these large datasets, often containing sensitive personal data, presents us with several ethical concerns.

One may be concerned about data privacy, as large-scale data collection enables the design of surveillance systems and the possibility of large-scale personal data breaches. In response to this concern, the notion of differential privacy [117] has been adopted by algorithms researchers as a unifying framework since it remedies many of the weaknesses, such as linkage attacks, seen in older approaches.

However, even if data is differentially private, another concern exists – discrimination through automated decision-making. Since predictive machine learning systems can often perform as well as, or even outperform, human analysts, automated decision-making systems have been adopted in many industries, such as forecasting stock market movements, the risk that a released criminal may re-offend, or the risk that a loan might be defaulted on. Due to these capabilities and their adoption, such systems have gained considerable influence over the lives of the general population. However, several high-profile cases have been documented, showing that these systems can operate in a biased fashion. Some notable examples are the COMPAS recidivism algorithm [110], the Amazon hiring algorithm [138], and the word2vec algorithm [48], though many others exist. These cases have shown a crucial need to ensure automated systems do not reproduce or reinforce social biases through their decision-making.

To formalize these requirements, the study of *algorithmic fairness* was initiated [62] and has since developed into a broad field of research, with many different approaches and communities, who all share the same central goals of measuring and improving the fairness of automated systems. To this end, researchers have proposed several definitions of algorithmic fairness, which fall into either group, subgroup, or individual fairness [62]. For more details, see our discussion of algorithmic fairness in Section 2.5. In this thesis, we design fair algorithms for a classical scheduling problem, the so-called paging problem. While our research is concerned with the design of fair algorithms, we realize that algorithmic fairness is a socio-technical problem and thus can only be effective as a field if it is also considered from a policy perspective, if we want to meet the goal of effectively protecting individuals from unjust discrimination stemming from algorithmic systems. For details on the social and legal aspects of the field, we urge the reader to refer to the excellent book by Barocas et al. [37].

1.2 Main Results and Outline

This thesis examines two algorithmic aspects of data science: clustering and algorithmic fairness. In this section, we briefly introduce the concrete problems studied in each chapter, together with the respective main results.

- In Chapter 2 we introduce some preliminary definitions and algorithms necessary for

the later chapters, together with an overview of the problems and related literature.

- In Chapter 3 we examine the classical paging problem [295] from a viewpoint of algorithmic fairness. In the classical problem, the goal is to minimize the total number of evictions of pages from a cache of size k while processing an input sequence that is presented in an online fashion, meaning we only get the next element of the sequence after processing the current element. If a page is not in the cache upon a request, we have to make an (irrevocable) decision to evict another page in order to make space for it. The quality of the algorithm is measured in terms of the competitive ratio, which is the worst-case ratio, taken across all instances, between the number of evictions incurred by the algorithm and the minimal possible number of evictions. By modifying the objective value to be the maximum number of evictions of a single page instead, which corresponds the notion of min-max fairness, we obtain a new constrained convex optimization problem called the *fair paging problem*. We show that the fair paging problem is strictly harder than the classical paging problem, as its competitive ratio depends not only on the cache size parameter k but also on the total number of pages n, by establishing lower bounds for the fair paging problem. The deterministic problem having a lower bound on the competitive ratio of $\mathcal{O}(k \log n / \log k)$ and the randomized problem having a lower bound of $\mathcal{O}(\log n)$, respectively. Additionally, we introduce a deterministic and a randomized algorithm with competitive ratio $\mathcal{O}(k \log k \log n)$ and $\mathcal{O}\left(\log^2 n \log k\right)$, respectively.

- In Chapter 4 we study the binary matrix factorization problem, a special version of the low-rank approximation problem which is closely related to k-means clustering. The goal of the binary matrix factorization problem is, given an input matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and a rank parameter k, to find two matrices $\mathbf{U} \in \mathbb{R}^{n \times k}$ and $\mathbf{V} \in \mathbb{R}^{k \times d}$ such that the difference $\mathbf{A} - \mathbf{UV}$ is minimized under some error measure. We introduce the first $(1+\varepsilon)$-approximation algorithms for the binary matrix factorization problem with Frobenius loss that run in time singly exponential in the rank parameter k. We also introduce bicriteria $(1+\varepsilon)$-approximation algorithms for variants of the binary matrix factorization problem with Frobenius loss on binary fields and L_p loss. Finally, we show that our algorithmic approach can be used to obtain the first big-data algorithms for this problem by translating it to the streaming and distributed models of computation.

- In Chapter 5 we study the classical k-means clustering problem in Euclidean space. Given a data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, the goal is to find a set C of k vectors, called centroids, which minimize the sum of squared distances $\sum_{\mathbf{x} \in \mathbf{X}} \min_{\mathbf{c} \in C} d(\mathbf{x}, \mathbf{c})^2$. We present the first algorithm for the k-means problem running in expected near-linear time $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log n)$, with its approximation ratio being $\tilde{\mathcal{O}}\left(k^4\right)$. Additionally, we prove a structural lemma that states that the approximation ratio of a k-means algorithm is approximately preserved under a class of linear projections, and we show that k-means++ (D^2-sampling) in a single dimension can be implemented to run in expected running time $\mathcal{O}(n \log n)$. We also implement our algorithm and

show that it can be used to significantly speed up coreset construction in practice and that it provides a new trade-off between running time and cluster quality compared to previous state-of-the-art methods.

- In Chapter 6 we turn our attention to graph clustering, and, in particular, the normalized cut objective. Given a graph $G = (V, E)$ and a parameter k, the goal is to find a k-partition of V which minimizes the sum of the ratios between the surface area (number of cut edges) and the volume (sum of degrees) of each component. To this end, we introduce XCut, a solver for the normalized cut objective on large graphs. XCut is based on the expander hierarchy, a powerful theoretical tool for computing graph sparsifiers. To the best of our knowledge, XCut is the first algorithm leveraging the theory of expander decomposition in a practical setting. We do this by showing that the expander decomposition framework of Saranurak and Wang [278] can be adapted to use a random-walk-based cut routine, leading to the first practically feasible near-linear time ($\tilde{\mathcal{O}}(m/\phi)$) algorithm for computing expander decompositions and the expander hierarchy. Our experiments show that XCut outperforms state-of-the-art solvers for this problem in terms of solution quality, while remaining competitive in terms of running time.

2 Preliminaries

In this section, we introduce the preliminaries for the later chapters. It is intended to provide a technical introduction to the main definitions, algorithmic tools, and existing algorithms, which we use to prove theorems and design algorithms in the following chapters, as well as an overview of important literature for the respective topics.

Notation As a convention throughout this thesis, dimensionless variables and functions are denoted by regular lowercase letters $(a, b, f, \dots)$, sets are denoted by capital letters $(A, B, \dots)$, vectors are denoted by lowercase bold letters $(\mathbf{a}, \mathbf{b}, \dots)$, and matrices are denoted by upper case bold letters $(\mathbf{A}, \mathbf{B}, \dots)$. $\mathbf{Pr}[y]$ is used to denote the probability of event y occurring, and $\mathbb{E}[x]$ denotes the expected value of the random variable x.

2.1 Models of Computation

Several alternative models more suited to dealing with large datasets have been introduced to remedy the weaknesses of the standard word-RAM model, as we discussed in Chapter 1. While achieving good parallelism in practice is, of course, tantamount to high-performance software, the main limiting factor when designing scalable big data algorithms, as mentioned before, is often related to memory access. As stated before, this usually is because we can only fit part of the dataset into the main memory of the computer we use to carry out data analysis or because its caches could be utilized more efficiently. Therefore, while algorithms designed in various parallel paradigms, such as the PRAM model, offer tremendous speedups compared to single-threaded, serial algorithms, applications where massive datasets are processed often benefit from being designed in a model conscious of memory usage. The distributed, streaming, and external memory models, as well as the field of cache-oblivious algorithms, all aim to be conscious of memory access by making fundamental assumptions about it in the model. Additionally, algorithms designed in these models can often be parallelized relatively straightforwardly. In this thesis, we mainly study the streaming, distributed, and online models, each of which we will briefly discuss in more detail below.

The *streaming model*, formally introduced by Alon et al. [16], assumes that the algorithm only ever reads the dataset sequentially rather than through random access, which is usually much more efficient. The main goal in this model is to minimize the amount of memory and the number of passes over the dataset that an algorithm needs to utilize, with researchers generally aiming to find single-pass algorithms that use space that is poly-logarithmic in the input size. Streaming algorithms have been developed for many

problems, such as finding frequent items [93, 68, 232], counting the number of distinct elements [16], estimating frequency moments [16], or clustering [4, 248, 77].

In the *distributed model*, we assume that the data is split across several machines, and the goal is to minimize the amount of communication between these machines. This amount can be either the total traffic, measured in terms of bits, or the number of rounds of communication. This focus on the amount of communication is motivated by the fact that synchronization and data transfer present the most significant bottleneck in networked computational tasks [257].

We note that both of these models are well suited to tasks where large amounts of data are analyzed, and they can often be used in conjunction, e.g., in a distributed computational task where each machine employs a streaming algorithm to carry out the local computations.

In some cases, we may not want to store the data generated by some monitoring process, and thus, we may want to analyze it in nearly real-time. The streaming model is often a viable candidate for applications like this, but some alternatives may be better depending on the problem we want to solve. These are the dynamic and online models.

In the *dynamic setting*, we receive the data incrementally through a series of updates to the dataset, such as the insertion and deletion of edges in a graph. In this model, the goal is to minimize the amount of computation performed for each update, and we generally aim for sub-linear or logarithmic update times. See the survey by Hanauer et al. [150] for more details.

On the other hand, in the *online model*, the goal is to make optimal decisions that can not be revoked upon the arrival of a new item. In this model, we usually do not optimize for the running time. A common problem in this model is managing the items in a cache, caching them as they arrive, and deleting them later while maximizing the number of cache hits and not having information about future requests. Applications where the online model has been fruitfully applied are matching bidders to online advertisement auctions [227], caching documents in server caches [221], and scheduling workloads in data-centers [140, 38]. More precisely, denoting a feasible instance of our problem by σ, in this model, we want to optimize a cost function $\text{cost}(\sigma)$ by hedging against all possible future inputs compared to the lowest value that is achievable for this instance $\mathsf{Opt}(\sigma)$. The worst-case ratio between the offline and online solution across all possible instances $\max_\sigma \frac{\text{cost}(\sigma)}{\mathsf{Opt}(\sigma)}$ is the so-called *competitive ratio* and the goal for any problem is to minimize (respectively maximize) it. See [295] for additional details on competitive analysis.

2.2 Clustering, k-Means, and Coresets

As the introduction outlines, clustering is one of the most fundamental and natural tasks in unsupervised machine learning. Finding groups of similar items in a dataset can generate new insights across manifold domains. Of course, one needs to be conscious of a number of factors, and defining what makes a clustering "good" is not something that can be done a priori. For instance, it is often unclear which representation of the data we should use for clustering, especially when the data is not already from a vector space,

since the choice of representation heavily factors into the results. This means that in applications, the choice of (dis)similarity measure is critical, together with whether we choose to scale or normalize the data or some of its attributes [324].

Furthermore, many clustering algorithms require us to set a parameter to determine the number of clusters. This choice is usually application-dependent, and no universally agreed-upon number or way of choosing one exists. However, several algorithmic approaches exist for this problem. These approaches usually formulate an auxiliary, data-dependent optimization problem over the parameter, allowing the algorithm to tune it. See [302, 126, 308, 121, 259] for some of the most well-known strategies for determining a best guess for the number of clusters.

Additionally, the quality of the clustering ultimately depends on the practitioner choosing the correct algorithm for the given dataset. Generally, each algorithm will make different assumptions about the shapes and sizes of clusters. For example, k-means will only allow convex clusters, while DBSCAN [120, 168] presupposes clusters of roughly uniform density while making no assumptions about their shapes. For more details regarding the correct choice of representation, algorithms, and parameters for a particular application, see the excellent books by Aggarwal and Reddy [8] or Gan et al. [133].

When developing clustering algorithms in this thesis, we do not consider these complex matters, as they are heavily application dependent. Our algorithms are evaluated across several different datasets and different parameter ranges, which are generally chosen to demonstrate an algorithm's feasibility for different domains and to let a practitioner make an informed choice when considering applying one of our algorithms.

As many hundreds of clustering algorithms exist, researchers have developed a taxonomy of clustering algorithms, with one division occurring between hierarchical and flat clusterings [133]. Hierarchical clusterings produce a hierarchy of clusters with different degrees of refinement, with phylogenetic trees being a familiar example. The expander hierarchy [139], which we discuss later, is another example of hierarchical clustering. Flat clustering algorithms do not produce a hierarchical structure but assign one or multiple cluster memberships to each data point. Examples of flat clustering algorithms are k-means clustering [301], model based clustering [102] or balanced graph partitioning [186]. In this thesis, we study both hierarchical and flat clustering algorithms.

One of the most widely studied clustering problems is the *k-means problem*, which has been studied for almost 70 years [301]. In this problem, the goal is to find k center points, which minimize the average squared distance of each point to the closest center.

More precisely, given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ of n d-dimensional points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in Euclidean space, the *k-means problem* is defined as the task of finding k points $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ that minimize the k-means cost

$$\operatorname{cost}(\mathbf{X}, C) = \sum_{i=1}^{n} \min_{j \in [k]} \|\mathbf{x}_i - \mathbf{c}_j\|_2^2.$$

Finding the optimal set of centers C^* which minimize this function given $\mathbf{X}$ is NP-hard unless one of k, d or n is 1 or k and d are constant [14, 216].

The clusters corresponding to any solution of the k-means problem are separated by a collection of hyperplanes that partition the space, leading to convex clusters. If more complex decision boundaries are desired, applying the kernel trick leads to a popular variant algorithm, kernel k-means [281].

Many popular algorithms, both theoretical [20, 182, 209, 57, 10, 141, 19, 197, 305, 131, 88, 85, 86] and heuristic [212, 214, 119] exist for the k-means problem, achieving good results in either setting. Among these many variants, people have also developed k-means algorithms suited to big data applications such as [211, 124]. The k-means problem also is closely related to the k-median [183] and k-center [160] problems.

Alternatively, we can formulate the k-means problem as a matrix optimization problem. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the data matrix once again, then finding an optimal solution to k-means clustering is equivalent to the following problem

$$\min_{\substack{\mathbf{C} \in \mathbb{R}^{k \times d}, \mathbf{S} \in \{0,1\}^{n \times k}, \\ \mathbf{S} \text{ has a single 1 per row}}} \|\mathbf{SC} \ - \mathbf{X}|_F^2,$$

where $\| \cdot \|_F$ is the Frobenius norm. In this formulation, given $\mathbf{C}$, we can easily find $\mathbf{S}$, as it is simply the shortest distance assignment. By allowing more general matrices rather than an indicator matrix in place of $\mathbf{S}$, we can obtain various forms of subspace clustering. In particular, if we allow $\mathbf{S}$ to be a real matrix, we obtain low-rank approximation, which is the problem of finding the k-dimensional linear subspace of $\mathbb{R}^d$, which best preserves the data under orthogonal projection. We make heavy use of this connection in Chapter 4.

The most widely used heuristic for solving k-means in practice is *Lloyd's algorithm* [212], which is an iterative EM (expectation-maximization) style algorithm. After initializing the centers using some seeding strategy, it repeats the following two steps until convergence:

1. (E-step): For $j = 1, \ldots, k$ compute the sets $S_j = \{\mathbf{x} \in \mathbf{X} \mid \mathbf{c}_j \text{ is the center closest to } \mathbf{x}\}$, and

2. (M-step): Using the assignment σ, compute the center $\mathbf{c}_j$ as the center of mass of the points in the set S_i.

It has been shown that on some datasets, this algorithm will take $\Omega\left(2^n\right)$ iterations to converge [312], but in practice, it usually only takes a small constant number of rounds. Variants such as the ones by Elkan [119] or MacQueen [214] are usually used in practice, as they tend to converge more rapidly.

Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ set of k' centers $\mathbf{c}_1, \ldots, \mathbf{c}_{k'}$, the D^2-*distribution* is defined as the probability distribution where each point $\mathbf{x}$ is drawn with probability proportionate to $\min_{j \in [k']} \|\mathbf{x} - \mathbf{c}_j\|_2^2$. That is, the probability that a point is drawn from the D^2 distribution is proportionate to its cost in the current k'-means clustering.

The k-means++ algorithm [20] (Algorithm 1) is a widely used approximation algorithm for the k-means problem. The core idea of the algorithm is to repeatedly sample a center from the D^2 distribution, which is then updated in accordance with the new center until k centers have been sampled. Intuitively, this works as the center we sample will not be too close to an existing cluster. Once a center is chosen inside a dense cluster, the

Algorithm 1: k-means++ [20]

Input: Dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$; $k \in \mathbb{N}$ with $k \leq n$.
Output: Centers $\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{R}^d$.

$\mathbf{c}_1 \leftarrow \mathbf{x}$ drawn uniformly from $\mathbf{X}$;
for $t = 2, \ldots, k$ **do**
  Let $\mathbf{c}_t = \mathbf{x}$ with probability $\propto \min_{j \in [t-1]} \|\mathbf{x} - \mathbf{c}_j\|_2^2$;
return $\mathbf{c}_1, \ldots, \mathbf{c}_k$;

cluster's probability mass in the D^2 distribution is low compared to other clusters. This algorithm is widely used in practice in conjunction with iterative algorithms like Lloyd's algorithm and its variants to obtain high-quality clusterings, where we use k-means++ to pick the initial set of centroids. The precise guarantees of k-means++ are as follows:

Theorem 2.1 ([20]). *Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ the k-means++ algorithm produces k centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$ in time $\mathcal{O}(knd)$. These centers give an $\mathcal{O}(\log k)$ approximate solution to the k-means objective in expectation. The algorithm also produces an assignment of each point in $\mathbf{X}$ to its closest center at no extra cost.*

A matching lower bound of $\Omega(\log k)$ for the approximation ratio of k-means++ is known. However, by oversampling the number of centers or performing local search, the approximation ratio can be further reduced to $\mathcal{O}(1)$ [80].

When considering a combinatorial optimization problem defined on some geometric space (such as k-means), *coresets* are often the go-to data-reduction technique. A coreset [7] usually entails a weighted subset of the input points that preserves the cost of all "queries", i.e., given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, we call a subset $\mathbf{A} \in \mathbb{R}^{m \times d}$ of $\mathbf{X}$ together with a weight function $w : [m] \to \mathbb{R}$ a *strong ε coreset* of $\mathbf{X}$ if for any set C of k centers we have

$$(1 - \varepsilon) \operatorname{cost}(\mathbf{X}, C) \leq \operatorname{cost}(\mathbf{A}, w, C) \leq (1 + \varepsilon) \operatorname{cost}(\mathbf{X}, C).$$

Coresets are generally constructed using a sampling approach, though deterministic approaches do exist.

The first approaches used grids to discretize the space and weigh each cell appropriately, but this approach leads to a coreset size that is exponential in d [7].

The coreset construction algorithm we use in Chapters 4 and 5 is called *sensitivity sampling*. This algorithm chooses a weighted sample of the input points, where points are weighted according to

$$\sigma(\mathbf{x}) = \max_C \frac{\min_{\mathbf{c} \in C} \|\mathbf{x} - \mathbf{c}\|_2^2}{\sum_{\mathbf{x} \in \mathbf{X}} \min_{\mathbf{c} \in C} \|\mathbf{x} - \mathbf{c}\|_2^2},$$

the maximum relative contribution to the k-means cost across all possible sets of centers. Intuitively, outliers will have high cost, while points that are part of a large cluster will have low individual sensitivity. Computing the exact sensitivity is hard, forcing us to use some (over)estimation of the sensitivity, the quality of which dictates the minimum

sample size needed to obtain $(1+\varepsilon)$ multiplicative guarantees. This is because the coreset size depends polynomially on the sum of the sensitivities. Usually, getting a good estimate of the sensitivity requires us to solve the k-means problem approximately. For more details on the algorithm see [28, 123, 51] We frequently make use of the following coreset construction:

Theorem 2.2 ([51]). *Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a dataset for k-means clustering. Let $m(\mathbf{x}) > \sigma(\mathbf{x})$ be an upper bound on the sensitivity of each point $\mathbf{x}$ in the dataset and let $t = \sum_{\mathbf{x} \in \mathbf{X}} m(\mathbf{x})$. Furthermore let $\varepsilon, \delta \in (0,1)$. By sampling $\mathcal{O}\left(\frac{t}{\varepsilon^2}\left(d \log t + \log\left(\frac{1}{\delta}\right)\right)\right)$ points from $\mathbf{X}$ proportionally to $m(\mathbf{x})$ and assigning them weight $\frac{1}{m(\mathbf{x})}$ we obtain an ε-coreset with probability at least $1-\delta$.*

Given an (α, β)-bicriteria approximation, it can be shown that $t \leq 6\alpha + 4\beta$, which, in the case of k-means++, gives us $t = \mathcal{O}(\log k)$.

Coresets often come with several desirable qualities, such as composability and straightforward adaptation to the streaming and distributed models, and they can make algorithms that may otherwise be too slow for practical applications [122]. Another desirable property of coreset approaches is that they can usually be used as a black box and speed up any existing algorithm for the problem we are studying [122]. Coresets have been successfully applied for clustering [52, 51, 164, 91, 28, 124] and supervised machine learning [143, 327, 294], among others [124, 98]. There is some overlap between sketching and coresets, with some authors using these terms interchangeably.

2.3 Dimensionality Reduction, Sketching and Low Rank Approximation

Dimensionality reduction has become a standard tool in machine learning, especially when dealing with high-dimensional data. Analyzing high-dimensional data can be difficult, as the data becomes too spread out in the feature space, or there may not be enough data on certain features. When analyzing data in a high-dimensional space, one intuitive idea is to map these vectors into a lower-dimensional space via a structure-preserving map. Among the popular and widely-used algorithms for dimensionality reduction are principal component analysis (PCA) [2], non-negative matrix factorization (NMF) [136], vector quantization [142], random projections [224, 177], or autoencoders [274, 319].

A *random projection* is a randomized function $f : \mathbb{R}^d \to \mathbb{R}^r$, where $r < d$. Generally, these functions are obtained by sampling an r dimensional matrix $\mathbf{A} \in \mathbb{R}^{r \times d}$ and computing the product $\mathbf{Av}$ for the input vector. If $\mathbf{A}$ is drawn from certain distributions, then the random projection can have a number of desirable properties. The most widely known result from this domain is the following:

Lemma 2.3 (Johnson-Lindenstrauss). *Given a set S of n points in $\mathbb{R}^d$, there exists a randomized linear map $f : \mathbb{R}^d \to \mathbb{R}^r$ with $r = \mathcal{O}\left(\frac{\log(n)}{\epsilon^2}\right)$ such that for any two points $\mathbf{u}, \mathbf{v} \in S$, with high probability*

$$(1-\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2^2 \leq \|f(\mathbf{u}) - f(\mathbf{v})\| \leq (1+\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2^2.$$

Intuitively, it tells us that any problem fully defined in terms of the lengths of some vectors can be projected into a problem of exponentially lower dimensionality.

Another approach to reducing the input data, other than reducing the number of observations or dimensions of the data matrix, is to find a *low-rank approximation* [223]. This is usually done by solving a matrix optimization problem of the form $\|\mathbf{A} - \mathbf{UV}\|$, with $\mathbf{A} \in \mathbb{R}^{n \times d}$, $\mathbf{U} \in \mathbb{R}^{n \times k}$ and $\mathbf{V} \in \mathbb{R}^{k \times d}$, the solution of which determines the factors $\mathbf{U}$ and $\mathbf{V}$ of the lower rank matrix. Reducing the rank, in turn, reduces storage costs and allows for faster matrix multiplication. The basic, unconstrained low-rank approximation problem with Frobenius norm can be solved using singular value decomposition. However, in many applications, it makes sense to place restrictions on the factors such as non-negativity [136], making them binary [202], or by weighting the entries differently [269]. These modifications usually yield NP-hard problems.

Linear sketching has proven itself as a powerful tool in the design of algorithms in many domains [321]. It has, for example, been leveraged as a tool to speed up clustering [69, 70], regression [321, 23], computation of frequency moments of streams [134, 18, 53], and it can be used to compute solutions to various graph problems in the semi-streaming model, such as connectivity, graph summarization, or sparsification [12, 13].

The core idea of linear sketching is to sample a matrix $\mathbf{S}$ from a particular distribution, which is chosen such that the distribution D of linear maps $f : \mathbb{R}^d \to \mathbb{R}^r$ induced by it is structure-preserving, for example, preserving the lengths of vectors, the angles between vectors or the linear subspace spanned by a set of vectors. By multiplying the input $\mathbf{A}$ of our problem (as a matrix) with the sketching matrix, we generate a compressed summary $\mathbf{SA}$ of a much smaller dimension. In a post-processing step, we compute the problem's answer with the summary $\mathbf{SA}$ as an input. For example, the summary may be a uniformly random incident edge for each vertex in a graph or a random projection of the input vectors. In many settings, multiplying by a linear sketch is akin to reducing the number of observations [321]. Linear sketching is especially powerful in the streaming setting, as matrix multiplication commutes with additive updates to the input vector, allowing us to compute the summary incrementally. In the turnstile streaming setting, where each token of the stream corresponds to an additive update to the input vector, which includes dynamic graph streams, linear sketching is the primary tool for designing effective algorithms. In fact Li et al. [210] have shown that, under some mild assumptions, there exists a bijection between algorithms in the turnstile streaming model and linear sketches [1].

Before moving on, we will consider least-squares regression as an illustrative example of how sketching is used in practice to obtain faster algorithms for optimization problems and in order to familiarize ourselves with this paradigm, as we make heavy use of it in Chapter 4. Our exposition follows the much more detailed treatment of the topic found in Woodruff [321]. In [321], the interested reader can also find a more detailed treatment of different subspace embeddings, including distributions that produce sparse matrices or matrices that let us compute $\mathbf{SA}$ in time $\mathsf{nnz}(\mathbf{A})$, along with a more detailed treatment of least-squares regression and a discussion of applications of sketching to problems from

[1] This result is non-constructive.

algebra and graph theory.

Example (Least Squares Regression)**.** *As an illustrative example for the sketch and solve paradigm, let us consider the least squares regression problem. In this problem, we are given a matrix* $\mathbf{A} \in \mathbb{R}^{n \times d}$ *of multivariate measurements and a vector* $\mathbf{b} \in \mathbb{R}^n$ *of observed outcomes. Furthermore, we want to assume that* $n > d$*, i.e., the problem consists of an over-constrained system of linear equations. If we assume that there is a linear relationship between our observed variables and the outcomes, our goal becomes finding the vector* $\mathbf{x} \in \mathbb{R}^d$ *which minimizes*

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2,$$

i.e., the vector which best describes the relationship between the observations and outcomes. By taking gradients, we can compute an analytical solution to the quadratic optimization problem, showing that $\mathbf{x} = \mathbf{A}^\dagger \mathbf{b}$ *is the minimizer, where* $\mathbf{A}^\dagger = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top$ *is the pseudo-inverse, assuming the columns are linearly independent. Computing* $\mathbf{A}^\dagger$ *can be done in* $\mathcal{O}\left(nd^2\right)$ *time using singular value decomposition. However, for large datasets, this can be prohibitively slow.*

Let us assume we can find a matrix $\mathbf{S} \in \mathbb{R}^{r \times n}$ *with* $r \ll n$ *such that*

$$(1 - \varepsilon)\|\mathbf{v}\|_2^2 \leq \|\mathbf{S}\mathbf{v}\|_2^2 \leq (1 + \varepsilon)\|\mathbf{v}\|_2^2.$$

This type of matrix is called a subspace embedding*. Then, by multiplying both* $\mathbf{A}$ *and* $\mathbf{b}$ *by* $\mathbf{S}$*, we obtain a new problem*

$$\min_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{S}\mathbf{A}\mathbf{x} - \mathbf{S}\mathbf{b}\|_2^2$$

and it is obvious by the property of $\mathbf{S}$ *that* $\mathbf{x}' = (\mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{A})^{-1} \mathbf{A}^\top \mathbf{S}^\top \mathbf{S}\mathbf{b} = \operatorname{argmin}_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{S}\mathbf{A}\mathbf{x} - \mathbf{S}\mathbf{b}\|_2^2$ *fulfills*

$$(1 - \varepsilon)\|\mathbf{A}\mathbf{x}^* - \mathbf{b}\|_2^2 \leq \|\mathbf{S}\mathbf{A}\mathbf{x} - \mathbf{S}\mathbf{b}|_2^2 \leq (1 + \varepsilon)\|\mathbf{A}\mathbf{x}^* - \mathbf{b}\|_2^2,$$

where $\mathbf{x}^*$ *is the optimal solution to the linear least squares regression problem. Since the number of rows in the sketched problem is* r*, we can compute a* $(1 + \varepsilon)$*-approximate solution in time* $\mathcal{O}\left(rd^2\right)$*. Using Lemma 2.3, which is a subspace embedding, we obtain* $r = \mathcal{O}\left(\varepsilon^{-2} \log(n)\right)$*, which yields an exponential reduction of our running time dependence on* n*. However, multiplying with* $\mathbf{S}$ *from the Johnson-Lindenstraus Lemma is slow, and we do not gain any asymptotic running time advantage in this case. It is known, however, that there are fast JL-transforms [321] based on Hadamard matrices, which allow us to compute* $\mathbf{S}\mathbf{A}$ *in time* $\tilde{\mathcal{O}}\left(nd\right)$*, finally letting us exploit the lower dimensionality of the sketched problem.*

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ with $d < n$, let $\mathbf{U}\Sigma\mathbf{V}^\top$ be its singular value decomposition, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices containing the left and right singular vectors respectively, and Σ is a diagonal matrix containing the singular values. We then define $\lambda_i = \|\mathbf{U}_i\|_2^2$, the norm of the ith row of $\mathbf{U}$ as the ith *leverage score*. Intuitively, the leverage score of a row measures how influential the row of $\mathbf{A}$ is in the solution to a

least-squares regression problem or, in other words, the level of non-uniformity induced by that row.

As $\|\mathbf{U}\|_F^2 = d$, we can define a probability distribution over the rows of $\mathbf{U}$ according to $\frac{\lambda_i}{d}$. Sampling rows from $\mathbf{A}$ according to this distribution is known as *leverage score sampling*. This distribution has been widely studied in the field of randomized matrix algorithms, as these matrices are powerful sketching tools, not only providing a subspace embedding but also approximately preserving matrix multiplication [82].

While computing the exact leverage scores requires us to compute the SVD, it has been shown that it is sufficient to approximate the leverage scores within a multiplicative constant to preserve the properties mentioned above. Drineas et al. [112] have shown that approximating the leverage scores in this fashion can be done in time $\mathcal{O}(nd \log n)$, i.e., time that is near-linear in the number of entries of the matrix.

We use approximate leverage score sampling in Chapter 4 to design algorithms for the binary matrix factorization problem.

2.4 Expander Graphs and Spectral Graph Theory

We generally denote graphs by $G = (V, E)$ and, by convention, use n to denote the number of vertices and m to denote the number of edges. We call $S \subset V$ a cut since it splits the set of vertices into $(S, \bar{S} = V \setminus S)$, and we denote the edges going from S to $\bar{S}$ by $E(S, \bar{S}) = \{(u, v) \in E \mid u \in S, v \notin S\}$ and call them the cut edges. Furthermore, we define the volume $\mathrm{vol}(S) = \sum_{v \in S} \deg(v)$ to be the sum of the degrees of the vertices in the subset.

As many real-world applications pertain to network or relational data, which is often best modeled as a graph, with nodes constituting members of the dataset and their attributes and edges representing some relational information between nodes, it becomes a natural question to ask how we can cluster graph data, especially as classical clustering algorithms are designed to cluster sets of points lying inside some metric space. We commonly encounter graph data that we may want to perform clustering on in the design of recommender systems [166, 165, 326, 316], social network analysis algorithms [283], and infrastructure applications [105, 103].

Many classical approaches for clustering graphs either rely on spectral methods [239] (discussed in greater detail below as well as in Chapter 6), or cut (resp. flow) based methods [279]. In practice, both approaches may encounter problems with scalability when we want to use them to analyze massive graph datasets if applied naively. When using spectral methods, the general approach is to embed a graph into some metric feature space and cluster it using algorithms such as k-means. Variations on this idea have become popular approaches for graph clustering in recent years, such as utilizing deep learning or graph neural networks to learn an embedding into a feature space [60]. However, these methods lose some edge information by approximating a discrete object's geometry in Euclidean space. For an example of the limitations this leads to, it has been shown that GNN-based methods can not distinguish non-isomorphic graphs that the Weisfeiler-Lemann heuristic fails to distinguish [235]. This fact motivates the need to develop

scalable graph clustering algorithms that directly work with a graph's combinatorial structure to produce high-quality clusterings. Work in this direction has focused on multilevel graph partitioning [186, 277], where graphs are successively contracted until a small representation of the original graph is obtained, allowing for rapid computation of a clustering.

A popular way of measuring the quality of graph clusterings is in terms of the number of edges crossing between clusters. While [158] found that the global minimum cut is usually not a meaningful objective for graph clustering in real-world datasets, introducing additional constraints or normalization terms, for example, via the cluster sizes, leads to meaningful clusterings [289]. However, objectives such as normalized cut or ratio cut are NP-hard, while minimum cuts can be computed in almost linear time.

We define the conductance of a cut S by $\Phi(S) = \frac{|E(S,\bar{S})|}{\min(\text{vol}(S),\text{vol}(\bar{S}))}$. The conductance of the graph G is then $\Phi(G) = \min_{S \subset V} \Phi(S)$. We call a graph a ϕ-expander if $\Phi(G) \geq \phi$, where $\phi \geq \mathbb{R}$ is some positive constant.

The expander graphs we are interested in are usually sparse, with average degree at most $\mathcal{O}(\log(n))$. By definition, they must be internally well-connected, meaning their internal structure approximates, in some sense, that of the complete graph or a random graph.

Another view is that an expander is a graph where each subgraph has a significant fraction of edges leaving the subgraph, meaning a random walk will quickly leave the component and diffuse into the rest of the graph. In other words, the graph can not be split into two parts by removing a few edges that are incident to some subset of vertices. Analogously, these graphs are defined by an absence of bottlenecks. We will make this intuition more precise below. Since expanders have a lot of favorable properties, some of which we quickly outline in the following few paragraphs (for further details, see the book by Kowalski [200]), designing approximation algorithms for many graph problems becomes significantly easier on an expander.

We let $\mathbf{A}$ denote the adjacency matrix and let $\mathbf{D}$ be the $n \times n$ matrix, which contains the degrees of each vertex on the diagonal. We define the *graph Laplacian matrix* as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. This matrix is symmetric and positive semi-definite, which means all its eigenvalues are real and non-negative. The eigenvalues and eigenvectors of $\mathbf{L}$ are particularly important in *spectral graph theory* as they encode information about the graph's connectivity.

In particular, the second smallest eigenvalue λ_2, after the eigenvalue $\lambda_1 = 0$, is related to the expansion properties of the graph G and called the *algebraic connectivity*. The eigenvector $\mathbf{v}_2$ associated with λ_2 is called the *Fiedler vector* and gives an approximation to the sparsest cut of our graph as follows:

Theorem 2.4 (Cheeger's inequality)**.** *Given a graph $G = (V, E)$, let λ_2 be the second smallest eigenvalue of the Laplacian matrix $\mathbf{L}$. Then*

$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}.$$

This theorem and its proof, which is done constructively using a sweep cut routine

(see Algorithm 20) on the Fiedler vector, forms the theoretical basis of various spectral graph partitioning routines and establishes a link between the cut structure of a graph and its spectrum. This widely used family of graph clustering algorithms uses the eigenvectors corresponding to the k smallest eigenvalues of the Laplacian matrix to obtain an embedding of a graph into Euclidean space, which then allows the use of classical clustering algorithms such as k-means to obtain a clustering of the graph [239]. The bisection algorithm using Cheeger's inequality corresponds to the case $k = 1$. Spectral methods tend to produce clusterings related to cuts with low conductance in the graph [239].

Another important property is that random walks on expander graphs mix rapidly, i.e., they quickly converge to the uniform distribution. Intuitively, this behavior is explained by the fact that there are no bottlenecks in the graph; thus, probability mass spreads quickly throughout the graph. More precisely, this is expressed by the following theorem:

Lemma 2.5 (Mixing time in expander graphs)**.** *Let G be a d-regular graph, and let $\mathbf{P}$ be the transition matrix of a random walk on this graph. Then, after t iterations of the random walk*

$$\|\mathbf{u} - \mathbf{P}^t\mathbf{a}\| \leq \sqrt{n}(\lambda_2/d)^t,$$

where $\mathbf{a}$ is any distribution over the vertices of G and $\mathbf{u}$ is the uniform distribution where each entry has value $1/n$.

An essential class of graph clusterings in theoretical computer science is that of *expander decompositions* [178]. An expander decomposition consists of a graph partitioned into expanders with relatively few edges crossing between them.

More precisely, a *ϕ-expander decomposition* is a partition $(V_1, \ldots, V_j)$ of V, such that the subgraph $G[V_i]$ induced by any part V_i of the partition is a ϕ-expander. Computing an expander decomposition is a bicriteria objective, as we want to minimize both the number of inter-partition edges as well as maximize ϕ. Most algorithms in the literature [299, 322, 237, 238, 278] follow the structure of the meta-algorithm outlined in Algorithm 2. The main challenge in obtaining good algorithms for this approach is to develop faster routines for identifying sparse cuts and methods to limit the recursion depth. See Chapter 6 for further discussion.

Algorithm 2: Expander decomposition meta-algorithm

Input: $G = (V, E), \phi \in \mathbb{R}$
Output: ϕ-expander decomposition $V_1, \ldots, V_j$

```
V_1, V_2 ← FindCut(G, φ);
if V_2 = ∅ then
    return V_1;
else
    return ExpanderDecomposition(G_1 = (V_1, E), φ) ∪
      ExpanderDecomposition(G_2 = (V_2, E), φ);
```

The most widely used expander decomposition algorithm in theoretical computer science is that of Saranurak and Wang [278]. This algorithm uses the *cut-matching game* [192]

to identify sparse cuts or, if none are found, certify that the graph is a near-ϕ-expander. In the case where the graph is a near expander, the algorithm of Saranurak and Wang uses a *trimming step* to identify a $\mathcal{O}(\phi)$-expander inside the near-ϕ-expander. In either case, a specific single commodity flow problem must be solved, configured according to the parameter ϕ and the degrees of vertices in the graph. Solving these problems can be done in $\tilde{\mathcal{O}}(m)$ time using specific max-flow algorithms, like Unit-Flow [159].

A data reduction technique commonly used in graph algorithms is *sparsification*.

Given a graph $G = (V, E)$ a graph $H = (V', E')$ with $V \subseteq V'$ is called a cut (resp. flow) sparsifier of G if $|E'| = \tilde{\mathcal{O}}(n)$ and we have that for a small constant q and any two disjoint sets S, T of vertices that

$$\underset{G}{\text{mincut}}(S,T) \leq \underset{H}{\text{mincut}}(S,T) \leq q \cdot \underset{G}{\text{mincut}}(S,T),$$

respectively for a small constant c and any multicommodity flow demand $D : V \times V \to \mathbb{R}$

$$\underset{G}{\text{congestion}}(D) \geq \underset{H}{\text{congestion}}(D) \geq \frac{1}{c} \cdot \underset{G}{\text{congestion}}(D).$$

Here, $\text{mincut}_G(S,T)$ indicates the minimum number of edges that we need to cut to separate S and T in G, and $\text{congestion}_G(D)$ is the maximum congestion across any edge when routing the demand D. This definition shows that a sparsifier is a special graph that preserves an important structural property of some other graph while having fewer edges. Sparsifiers can preserve more properties than just the size of cuts [41]. For example, they can preserve pairwise distances, in the case of spanners [11], or the eigenvalues of the graph Laplacian, in the stronger notion of spectral sparsifiers [300]. As V' can be a superset of V, sparsifiers can radically alter the underlying structure of a graph, as is the case with Räcke trees [265]. These trees reduce the graph to a tree where the original vertices become the leaves, and the minimum cut between any set of leaves in the tree can approximate the minimum cut between these vertices in the graph up to a logarithmic factor [266].

There also exists a connection between sparsification and linear sketching. By sketching the incidence matrix of a graph, it has been shown that sparsification can also be performed in the streaming setting. Ahn et al. [12] showed this connection for cut sparsifiers and Ahn et al. [13] showed it for spectral sparsification. Similar to the way sketches and dimensionality reduction can provide black-box running time improvements, many graph problems whose running time depends on the number of edges can immediately have their running time reduced using sparsification. See [41] for details.

A *(α, ϕ)-boundary linked expander decomposition* of a graph G is an expander decomposition, where each component V_i of the expander decomposition is a ϕ expander while adding α/ϕ-self loops for each edge leaving the expander. This is a stronger property than being a regular ϕ expander decomposition if $\alpha/\phi > 1$, as it requires all vertices to be more strongly connected internally than externally.

Given a graph G, the expander hierarchy is defined as the sequence of graphs $G_0 = G, G_1, \ldots, G_\ell$, where G_{i+1} is obtained by computing a (α, ϕ)-boundary linked expander decomposition of G_i and then contracting all of the expanders. We call ℓ the *depth* of the

expander hierarchy. This can be used to construct a tree from the bottom up, where the set of vertices at height i consists of the vertices of G_i, denoted by $v_{i,j}$, and there is an edge from $v_{i,j}$ to $v_{i+1,j'}$ if v_j in G_i is part of expander $V_{k'}$ in the expander decomposition of G_i. Each such edge is assigned weight $\deg_{G_i}(v_j)$.

Theorem 2.6 ([139]). *The expander hierarchy of depth $\tilde{\mathcal{O}}\left(\log^{1/4}(n)\right)$ a graph can be computed in $\tilde{\mathcal{O}}\left(m^{1+o(1)}\right)$ time, and can be maintained under edge insertions and deletions in amortized time $\tilde{\mathcal{O}}\left(n^{o(1)}\right)$ for $\alpha = 1/\operatorname{poly}\log(n)$ and $\phi = n^{o(1)}$. Additionally, the tree defined by the expander hierarchy is a flow-sparsifier (and in particular a Räcke tree) of quality $n^{o(1)}$.*

2.5 Algorithmic Fairness

In this final section, we give a brief overview of the topic of algorithmic fairness, define different notions, and introduce common techniques used to achieve fairness. We use our understanding of algorithmic fairness to design fair algorithms in Chapter 3.

Group fairness notions aim at equalizing the probability of some outcome across the different possible values for the protected attribute. Generally, these notions depend on some form of (conditional) statistical independence of the outcome of the system and the protected attributes [62]. This can be the likelihood of a positive outcome in the case of demographic parity [176], the true positive rate in the case of equal opportunity [153], the true and false positive rate in the case of equalized odds [43], and the false positive and false negative rate in the case of test fairness [43]. Achieving calibration is also seen as a level of fairness in the literature [194].

Individual fairness aims to give similar outcomes to similar people and depends on a similarity measure. Popular definitions of individual fairness are fairness through awareness [118] and counterfactual fairness [203]. When no explicit protected attribute is present, we can include min-max fairness [267] in this category. In min-max fairness, we want to maximize the outcome of the worst individual in the dataset, i.e., we try to establish a lower bound on the worst outcome.

Subgroup fairness [190] aims to extend considerations of group fairness to subsets of the population to achieve more thorough fairness guarantees and ensure that individuals in the subgroups have similar outcomes to the overall population.

Fairness can be achieved through a number of interventions in a data processing task or a machine learning pipeline where the goal is to train a model for some task, such as classification, on existing data. The approaches roughly fall into pre-processing, in-processing, and post-processing interventions [62].

Pre-processing approaches aim to remove biases in the training data [62]. Intuitively, this is done by equalizing the distributions of outcomes of different protected classes. These approaches can range from simple omission of protected variables [193, 115, 153], relatively simple transforms such as reweighing [125], or changing the labels of some instances [175], as well as sampling subsets of the data that leads to fairer outcomes [310, 287], to sophisticated approaches that employ representation learning to create a new synthetic

dataset that does not exhibit biases regarding the protected variables [331, 336, 94, 333] or causal models that offer thorough explanations of confounding variables [203, 220].

In-processing approaches usually modify the formulation of the loss function or the optimization problem we want to solve. Commonly, this involves introducing a fairness regularization term that penalizes a model for unfair behavior [176, 125, 43] or by adding additional constraints to the optimization problem [330, 64, 5]. Alternatively, adversarial approaches can be used to detect and nullify biases in the models that are being trained [333], or, more recently, bandits can be used to learn a fair distribution of actions to take in an online fashion [173].

Post-processing for fairness is usually done by either learning different thresholds for different protected groups or by applying a transformation to the output distribution so that it achieves statistical calibration [153]. However, this approach has been criticized [263].

For more detailed fairness definitions and algorithmic approaches to fairness, see the book by Barocas et al. [37] or the surveys by Caton and Haas [62] or Mehrabi et al. [226].

While most literature on algorithmic fairness is concerned with classification using supervised learning approaches and models trained on personal data, the relevance of algorithmic fairness extends to other data processing tasks. In particular, in the context of (unsupervised) clustering algorithms, which often select representatives in machine learning pipelines, the question of fairness becomes relevant. When given some data with each point belonging to some group, the goal of obtaining clusters containing a representative fraction of each group or representatives that fulfill similar balancing criteria motivates the fair clustering problem [79]. Fairness in clustering has been studied for k-means clustering [79, 30, 42, 135, 241, 163, 280], k-medians clustering [1, 241, 163], k-center clustering [75, 152, 195], and hierarchical clustering [9, 78]. Additionally, fairness has been studied for PCA [275, 174, 196], recommender systems [320], regression [6, 291], scheduling [167, 306] and network algorithms [101, 162]. In a part of this thesis, we study algorithmic fairness in networking algorithms.

3 Online Min-Max Paging

Motivated by fairness requirements in communication networks, we introduce a natural variant of the online paging problem, called *min-max* paging, where the objective is to minimize the maximum number of faults on any page. While the classical paging problem, whose objective is to minimize the total number of faults, admits k-competitive deterministic and $\mathcal{O}(\log k)$-competitive randomized algorithms, we show that min-max paging does not admit a $c(k)$-competitive algorithm for any function c. Specifically, we prove that the randomized competitive ratio of min-max paging is $\Omega(\log(n))$ and its deterministic competitive ratio is $\Omega(k \log(n)/\log(k))$, where n is the total number of pages ever requested.

We design a fractional algorithm for paging with a more general objective – minimize the value of an n-variate differentiable convex function applied to the vector of the number of faults on each page. This gives an $\mathcal{O}(\log(n)\log(k))$-competitive fractional algorithm for min-max paging. We show how to round such a fractional algorithm with at most a k factor loss in the competitive ratio, resulting in a deterministic $\mathcal{O}(k \log(n)\log(k))$-competitive algorithm for min-max paging. This matches our lower bound modulo a $\operatorname{poly}\log(k)$ factor. We also give a randomized rounding algorithm that results in a $\mathcal{O}\left(\log^2 n \log k\right)$-competitive algorithm.

3.1 Introduction

Paging is a decades-old, classical computer science problem. Suppose a computer process working on n *pages* of data has access to two levels of memory: a fast memory, called the *cache*, that can hold a small amount k of pages, and a slow memory containing all n pages. Typically, k is much smaller than n, and initially, all pages are in slow memory. Whenever the process accesses a page, it is read from the cache; if it is not already in the cache, a *page fault* occurs, and the page must be brought into the cache, which possibly necessitates *evicting* another page from the cache to make room. This is called *serving* the request. In the *online* setting, each request must be served before the algorithm sees the subsequent request. The goal is to minimize the *total number of page faults* incurred while serving a sequence of requests.

The paging problem has found new applications in communication networks, where caching is ubiquitous and is used to minimize energy usage, communication latency, and network traffic. Consider, for example, TCP connections that are kept alive on a router [83] or optical links in reconfigurable data center topologies [129, 46]. Every user application prefers to have an active connection, as re-establishing a TCP connection or link takes time and slows down communication or computation. Another example is

content on web pages that is cached in a content delivery network, such as Akamai. In practice, the cache servers in these networks rely on dynamic, eviction-based algorithms for managing cache contents that solve the so-called content placement problem [307]. Web pages in the cache have a clear advantage as they can be served faster to the user than web pages that must be re-fetched from the server. Ideally, all applications (of the same priority) and all web pages should be treated equally. This motivates us to propose the study of a *fair* variant of paging, which we call *min-max paging*. Its goal is to minimize the number of page faults on *any* page, i.e., to minimize the *maximum number of page faults of any single page*.

In the *online* setting, the page requests are revealed one by one without knowledge of the future, so the description of how to serve each request must depend only on the request sequence thus far and the current cache contents. Naturally, for many problems, an online algorithm cannot output an optimal solution to a given instance – something an *offline* algorithm having access to the entire input can produce. The sub-optimality of an online algorithm is usually measured using competitive analysis. Informally, we say that an online algorithm has a competitive ratio of c if, on every problem instance, it produces a solution with (expected) cost at most c times the cost of an optimal solution. For the classic online paging problem the competitive ratio has been well-studied: It is $\mathcal{O}(k)$ for deterministic algorithms [295, 185] and $H_k \approx 0.577 + \ln(k)$ for randomized algorithms [127, 225]. To the best of our knowledge, the min-max paging problem has not been studied before. While an efficient offline algorithm for the classical paging problem is known, we neither have an efficient offline algorithm for min-max paging nor a proof of NP-hardness. In this chapter, we focus on the min-max paging problem in the *online* setting and give both upper and lower bounds on its competitive ratio.

Our results We first propose an algorithm for the *fractional paging problem* with objective function f, where pages can be held in the cache fractionally, subject to having a total volume of at most k pages at all times. The objective function f is an arbitrary function from an appropriately defined subclass of convex functions applied to the *fault vector*. Here, the fault vector refers to the n-dimensional vector of the number of faults incurred on each page, where n is the number of pages. We use the theory of convex programming and properties of f to analyze our algorithm and establish the following bound.

Theorem 3.1 (Stated formally as Theorem 3.19)**.** *For the fractional paging problem with objective function f, there exists a $(2q\log(k+1))^q$-competitive algorithm, provided f grows no faster than a degree-q polynomial.*

In particular, when instantiating f to be the q'th power of the q-norm, we get the following bound:

Theorem 3.2 (Stated formally as Theorem 3.20)**.** *For the fractional paging problem with the objective of minimizing the L_q-norm of the fault-vector, there exists a $2q\log(k+1)$-competitive algorithm.*

Note that the above theorem does not give a sensible result for the L_∞-norm, which is the objective function we are interested in. However, using the fact that the L_∞-norm of

an n-dimensional vector is well-approximated by its $L_{\log(n)}$-norm, we get the following result.

Theorem 3.3 (Stated formally as Theorem 3.21)**.** *For the fractional paging problem with the objective of minimizing the L_∞-norm of the fault-vector (a.k.a. fractional min-max paging), there exists an $\mathcal{O}(\log(n)\log(k))$-competitive algorithm.*

Next, we propose two approaches for rounding solutions of fractional min-max paging algorithms online and obtain the following two results. Note that the bound of the latter result is better than the former in the $k = \omega(\log n)$ regime, and it also rules out a lower bound linear in k for randomized algorithms.

Theorem 3.4 (Stated formally as Corollary 3.23)**.** *There exists an $\mathcal{O}(k\log(k)\log(n))$-competitive* deterministic *algorithm for min-max paging.*

Theorem 3.5. *There exists an $\mathcal{O}\left(\log^2 n \log k\right)$-competitive* randomized *integral algorithm for min-max paging.*

We complement the above upper bounds by the following impossibility results.

Theorem 3.6 (Stated formally as Theorem 3.8)**.** *Every deterministic algorithm for min-max paging is $\Omega(k\log(n)/\log(k))$-competitive.*

Theorem 3.7 (Stated formally as Theorem 3.9)**.** *Every algorithm for min-max paging is $\Omega(\log(n))$-competitive.*

Note that we only have a $\mathcal{O}\left(\log^2 k\right)$ discrepancy between our deterministic bounds, i.e., the bounds are tight up to a poly-logarithmic in k factor. Moreover, our lower bounds show that min-max paging is fundamentally more difficult than classical paging and its several generalizations (see Section 3.6), which admit competitive ratios independent of n, the total number of pages.

We now present some intuition why algorithms for the classical paging problem and a simple algorithm for min-max paging fail to achieve anything better than a trivial competitive ratio for min-max paging. Algorithms for the classical paging problem are oblivious to the number of faults a single page has incurred while processing the sequence σ up to a given point in time t. Consider the Least Recently Used (LRU) algorithm, which evicts the page whose last request was before the requests to other pages in the cache. Let $p_0 \in P = \{p_0, p_1, \ldots, p_n\}$ and assume that $n = |P| - 1 = mk$ is a large multiple of k. The sequence $\sigma = (p_0, p_1, p_2, \ldots, p_k, p_0, p_{k+1}, p_{k+2}, \ldots, p_{2k}, p_0, p_{2k+1}, \ldots, p_{mk}, p_0)$ will cause the LRU algorithm to fault $m+1$ times on page p_0, while the optimal algorithm faults exactly once per page. We pair each request to p_0 with requests to a set of k pages. After processing these k requests, LRU will have ejected p_0, so the next request to p_0 will result in a page fault, yielding in a competitive ratio of $\Omega(n/k)$.

Another obvious strategy is to greedily keep the k pages which have incurred the most faults thus far in the cache. In this case, there also exists a request sequence for which the strategy is no better than $\frac{n}{k}$-competitive. For simplicity's sake, let us assume that $k = 2$. Then the request sequence is constructed as follows:

1. Request p_1, p_2, p_3 in this order N times, where N is a parameter.
2. Request p_4, p_5 until the algorithm includes *both* of them into the cache.
3. Request p_4, p_5, p_6 in this order N times.
4. Repeat steps 2 and 3 with pages p_7, p_8, and p_9 next, then with pages p_{10}, p_{11}, and p_{12}, and so on.

After step 3, the greedy algorithm will hold two pages of cost $(r+1)N$, where r is the number of times we have repeated steps 2 and 3. In step 2, we request a set of new pages, and the greedy algorithm will fault on them until they reach cost $(r+1)N$. During step 3, the algorithm will fault N times on each page, making it so that it has cost $(r+2)N$ on the pages introduced in step 2. Meanwhile, the cost of the optimal offline algorithm is no greater than N, obtained by immediately adding the pages of step 2 to the cache.

Our techniques Our lower bound of $\Omega\left(k \log(n)/\log(k)\right)$ is established by generalizing the above construction, using the following approach: The adversary fixes a sequence of requests over a set of n pages, which can be served while keeping the number of faults on any page small. The core idea is to successively reduce the set of pages that we request in the future in such a way that the algorithm *cannot predict which pages will stop being requested.* A clairvoyant adversary processes the sequence so that she initially incurs a small number of faults on pages that will be requested many times in the future. This causes the adversary to have roughly uniform cost over all pages, while the algorithm has one page on which it has faulted many times.

To design an online algorithm one could try to use standard techniques to transform a max-based objective function into a linear program and solve the corresponding linear program online. However, this does not work as all known online algorithms for linear programs only work with *exclusively packing or exclusively covering constraints* and can not handle a mix of constraints, except for Azar et al. [24], which cannot handle *box constraints,* i.e., an upper bound on the variables as required for paging.

Thus, to solve the online min-max paging problem, we solve a more general problem: We give a $\mathcal{O}\left((q \log k)^q\right)$-competitive algorithm for a *fractional paging problem, which minimizes a convex, differentiable function with q-bounded growth* and *an upper bound constraint (i.e., a box constraint) on each variable.* A function with q-bounded growth behaves like a polynomial function of degree q. To the best of our knowledge, this problem has not been studied before, and no non-trivial online algorithm is known.

As our cost function is not linear, the combinatorial technique of potential functions used for server problems with linear cost functions breaks down. Informally, a potential function captures the advantage accumulated by the adversary at any time, which she can use to make the algorithm "pay" more than herself in the future. The potential function is a function on the *state space* of the problem, where the state of the algorithm, at any time, fully determines its future behavior. The state space is usually a small set when the objective is linear. On the contrary, in the case of min-max paging, a state must capture the vector of faults accumulated on each page and its current cache, and there can be

multiple fault vectors for the same current cache, which makes the state space blow up with every request, thus, making the use of potential functions challenging, messy, and inelegant.

Instead, we build on the work of Azar et al. [25], who minimize a convex cost function with linear constraints of *row sparsity* ρ. Their approach requires the variables $x_{p,j}$ to be unbounded, and for q-bounded growth functions, it gives an $\mathcal{O}\left((q \log \rho)^q\right)$-competitive algorithm.

We also draw on ideas from Bansal et al. [35], who studied the weighted paging problem with linear cost functions. They first compute a fractional solution using a primal-dual approach and then show how to round it. As they have a linear cost function, they can show that the rate of increase of the primal, i.e., the fractional algorithm's cost, is proportional to the rate of increase of the dual. In our setting, the cost function is not linear, and we have to use the theory of duality of convex programs and conjugate duals. To do so, we extend their analysis to the convex program setting, which requires solving various technical hurdles. This results in a $(2q \log(k+1))^q$-competitive algorithm for fractional paging with any convex, differentiable function with q-bounded growth and box constraints. Furthermore, for norm-objective functions, more specifically for q-norms, we achieve a competitive ratio of $2q \log k$. Since the cost function of min-max paging is the L_∞-norm of the vector of page-wise costs, we approximate it by $L_{\log n}$-norm, resulting in a $2e \log n \log(k+1)$-competitive algorithm for fractional min-max paging.

We round our solution deterministically using for every page p a threshold for x_p of $1 - 1/k$, resulting in the upper bound of $\mathcal{O}\left(k \log n \log k\right)$. It might be tempting to apply the randomized rounding algorithm of Bansal et al. [35] directly, but it does not apply as it crucially uses the fact that the cost of the algorithm is the sum of the fractional values of all pages. Instead, we adapt the scheme of Bansal et al. [36] from the weighted paging setting to the min-max setting. Specifically, this requires to "charge" the cost of each rounding step to each individual page, as opposed to the sum of the changes in the fractional solution over all pages. This charging to individual pages has not been done before in online paging and might be interesting in other settings.

Organization of the chapter. In Section 3.2, we give all definitions. In Section 3.3, we show our lower bounds, in Section 3.4, we present and analyze our algorithm for paging with convex objective functions. In Section 3.5 we round the fractional algorithm to obtain an $\mathcal{O}\left(k \log(n) \log(k)\right)$-competitive deterministic and $\mathcal{O}\left(\log^2(n) \log(k)\right)$-competitive randomized algorithms for (integral) min-max paging.

3.2 Preliminaries

The problems in this chapter are studied in the online setting, where an adversary fixes a request sequence σ ahead of time, and the requests in this sequence are presented to an algorithm one by one. When the algorithm receives a new request from the sequence, it can only use its knowledge of the requests seen thus far to make a decision. In particular, the algorithm does not have any knowledge of future requests.

In this setting, we use *competitive analysis* [295] to measure the quality of an algorithm. In competitive analysis, we study the *competitive ratio* of an online algorithm, which compares the worst-case ratio between the cost of the algorithm and the cost of an optimum offline solution over all possible σ.

More formally, for a deterministic algorithm ALG, the competitive ratio of ALG is the smallest $c \in \mathbb{R}$, such that for all instances σ of an online minimization problem, we have

$$\mathrm{ALG}(\sigma) \leq c \cdot \mathsf{Opt}(\sigma) + d,$$

where $\mathrm{ALG}(\sigma)$ is the cost of the algorithm, $\mathsf{Opt}(\sigma)$ is the cost of the optimum offline solution, and d is some constant independent of σ. We will call an algorithm fulfilling the above definition a c-competitive algorithm. If ALG is a randomized algorithm, then the competitive ratio is defined as the smallest $c \in \mathbb{R}$ such that

$$\mathbb{E}\left[\mathrm{ALG}(\sigma)\right] \leq c \cdot \mathsf{Opt}(\sigma) + d.$$

We study a variant of the paging problem called *min-max paging*. In any paging problem the request sequence σ is made up of requests to a set of pages $P = \{p_1, p_2, \ldots, p_n\}$ of size n. We will assume that σ is of finite length, denoted by T. The algorithm is given a cache C of size k, which always is a subset of P and is empty when the algorithm begins processing σ.

When page p is requested during round t, we must add p to the cache C if it is not already contained in C. If adding the page causes C to be of size $k+1$, we must evict a page other than p from the cache before we are allowed to process the next request. The situation where a request to page p arrives while p is not in C is called a *page fault*.

Whenever a page fault occurs, we incur some cost. The objective of the classical paging problem is to minimize the total number of page faults. In the case of min-max paging, the objective is to minimize the maximum number of page faults occurring for any page. More precisely, if we let $x_{p,j}$ be a zero-one variable, which denotes that a page fault occurs upon the j-th request to page p, then we seek to minimize

$$\max_{p \in P} \sum_{j} x_{p,j},$$

where the summation is over all requests to p. We can think of this as minimizing the L_∞-norm of the vector $\mathbf{c}(\sigma) = (\sum_j x_{p_1,j}, \ldots, \sum_j x_{p_n,j})^\top$, whereas the classical paging problem is equivalent to minimizing the L_1-norm of $\mathbf{c}(\sigma)$.

In Section 3.4 we solve a fractional version of the paging problem for convex objective functions $f(x)$, where x is the vector consisting of the variables $x_{p,j}$, under the assumption that $f(x)$ is well behaved. Of particular interest is the case where $f(x) = \|\mathbf{c}(\sigma)\|_q$, i.e., the L_q-norm. We refer to this case as *q-paging*. For details refer to the beginning of Section 3.4.

Remark. *Paging problems are studied in the eviction cost model, where fetching a page incurs no cost, and the algorithm pays for evicting a page, and in the fetching cost model,*

where evicting a page comes without an associated cost, and the algorithm pays for fetching a page. For min-max paging, the cost of these models differs by at most 1*. Said difference occurs on the set of pages contained in the cache at time T that the algorithm does not have to evict anymore.*

Because of this equivalence, we use both models interchangeably in this chapter. The lower bounds of Section 3 use the fetching cost model, and the upper bounds of Section 4 use the eviction cost model, as the choice of the respective model simplifies the proofs.

3.3 Lower Bounds

We show a deterministic lower bound of $\Omega\left(k(\log n)/\log k\right)$ and a randomized lower bound of $\Omega\left(\log n\right)$ (for $k = 2$) on the competitive ratio for min-max paging. Our lower bounds are based on a simple construction that is cleanly demonstrated with $k = 2$ and can be generalized for $k \geq 2$. The interested reader will find complete proof for the deterministic lower bound in Section 3.7.

Theorem 3.8. *Any deterministic algorithm for min-max paging with cache size k is at least $\frac{k-1}{2}\log_{k+1} n$-competitive, where n is the number of pages.*

Theorem 3.9. *The randomized competitive ratio of min-max paging is $\Omega\left(\log n\right)$, where n is the number of pages when the cache size is $k = 2$.*

By Yao's principle, it suffices to exhibit a probability distribution on input instances, forcing every deterministic online algorithm to perform a factor $\Omega\left(\log n\right)$ worse in expectation than the optimum cost. Let $n = 3^m$ for some large integer m. Our adversarial strategy takes a parameter $N \gg n$ and is defined as follows.

Algorithm 3: An adversarial strategy for min-max paging

```
1 Let L_m = {p_0^m, ..., p_{n-1}^m} be a set of n = 3^m pages;
2 for ℓ = m to 1 do
3     L_{ℓ-1} ← ∅;
4     for i = 0 to 3^{ℓ-1} − 1 do
5         Give N requests to each of p_{3i}^ℓ, p_{3i+1}^ℓ, p_{3i+2}^ℓ in a round-robin manner;
6         p_i^{ℓ-1} ← a uniformly random page from {p_{3i}^ℓ, p_{3i+1}^ℓ, p_{3i+2}^ℓ};
7         Add p_i^{ℓ-1} to L_{ℓ-1};
```

We call each iteration of the outer for-loop a *layer* and each of the inner for-loop a *phase*. We number the layers $m, m-1, \ldots, 1$.

Lemma 3.10. *The adversary can serve all requests while faulting at most $m + N$ times on every page with probability one.*

Proof. Consider an arbitrary phase of an arbitrary layer ℓ. Let q be the page added to $L_{\ell-1}$ at the end of the phase, and let q_1, q_2 be the other two pages requested in the phase. On the first request to q, the adversary will add q to its cache and keep it there until the

end of the phase. It uses the remaining cache slot to serve all requests to q_1 and q_2. Thus, the adversary faults only once on q and N times on q_1 and q_2 each.

Consider an arbitrary page p. In all phases where p is requested except the last one, the adversary faults only once on p (p is the page q in the above argument). In the last phase, the adversary faults N times on p. Every layer contains at most one phase in which p is requested. Since the number of layers is m, the algorithm faults at most $m+N$ times on p. □

To analyze the algorithm's performance, let the random variable x_i^ℓ be the number of the algorithm's faults on the randomly chosen page p_i^ℓ at the beginning of layer ℓ.

Lemma 3.11. *For every layer ℓ and every $i \in \{0, \ldots, 3^\ell - 1\}$ we have $\mathbb{E}\left[x_i^\ell\right] \geq (m-\ell) \cdot N/2$.*

Proof. We prove the claim by reverse induction on ℓ. Recall the numbering of phases and observe that $x_i^m = 0$ for all $i \in \{0, \ldots, n-1\}$. Thus, the claim is true for $\ell = m$. Assuming as induction hypothesis that for every $i \in \{0, \ldots, 3^\ell - 1\}$ we have $\mathbb{E}\left[x_i^\ell\right] \geq (m-\ell) \cdot N/2$, we prove that for every $j \in \{0, \ldots, 3^{\ell-1} - 1\}$ we have $\mathbb{E}\left[x_j^{\ell-1}\right] \geq (m-\ell+1) \cdot N/2$.

In any phase, since the cache size is 2 and three pages are requested in a round-robin manner N times each, the total number of faults is at least $3N/2$. This is evident if we consider the behavior of the optimal algorithm for (usual) paging that always evicts the page needed farthest in the future. Consider the j'th phase of layer ℓ, and recall that $p_j^{\ell-1}$ is defined at the end of this phase. The total number of faults in this phase is at least $3N/2$, and these faults are distributed over the three pages, $p_{3j}^\ell, p_{3j+1}^\ell, p_{3j+2}^\ell$. Since $p_j^{\ell-1}$ is uniformly random among these three pages, the expected number of faults on $p_j^{\ell-1}$ during layer ℓ is at least $N/2$. Again, since $p_j^{\ell-1}$ is uniformly random among $\{p_{3j}^\ell, p_{3j+1}^\ell, p_{3j+2}^\ell\}$, by linearity of expectation we have,

$$\mathbb{E}\left[x_j^{\ell-1}\right] \geq \frac{\mathbb{E}\left[x_{3j}^\ell\right] + \mathbb{E}\left[x_{3j+1}^\ell\right] + \mathbb{E}\left[x_{3j+2}^\ell\right]}{3} + \frac{N}{2}.$$

By the induction hypothesis, each of $\mathbb{E}\left[x_{3j}^\ell\right]$, $\mathbb{E}\left[x_{3j+1}^\ell\right]$, $\mathbb{E}\left[x_{3j+2}^\ell\right]$ is at least $(m-\ell) \cdot N/2$. Thus, $\mathbb{E}\left[x_j^{\ell-1}\right] \geq (m-\ell+1) \cdot N/2$, as required. □

Having proven Lemma 3.10 and Lemma 3.11, we are ready to prove the claimed lower bound.

Proof of Theorem 3.9. By Lemma 3.10, the cost of the adversary's solution to the random instance generated by the adversarial strategy is $m+N$ with probability one. Note that at the end of the adversarial strategy, we are left with the singleton set L_0 containing the page p_0^0. The number of faults of the algorithm on page p_0^0 is a lower bound on the algorithm's cost with probability one. Thus, the algorithm's expected cost is at least the expectation of the number of algorithm's faults on p_0^0. By Lemma 3.11, this quantity is $\mathbb{E}\left[x_0^0\right] \geq mN/2$. Thus, the ratio of the algorithm's expected cost to the

adversary's cost is at least $mN/(2 \cdot (m+N))$, which approaches $m/2 = (\log_3 n)/2$ as $N \to \infty$. Thus, the competitive ratio of any randomized algorithm for min-max paging is at least $(\log_3 n)/2 = \Omega(\log n)$. □

3.4 A Fractional Algorithm for General Paging

We study a general class of convex objective functions for the paging problem to arrive at a competitive algorithm for min-max paging. Let $\mathbf{x} \in \mathbb{R}^T$ be the vector consisting of the variables $\mathbf{x}_{p,j}$ in order of appearance in σ. The objective functions $f : \mathbb{R}^T \to \mathbb{R}$ which we consider in this section have the following properties:

1. $f(0) = 0$,
2. $f(\mathbf{x})$ is a monotonically increasing function in $\mathbf{x}$,
3. $\nabla f(\mathbf{x})$ is monotonically increasing in each coordinate, and
4. $f(\mathbf{x})$ has *q-bounded growth*, i.e., there exists a positive integer q such that for all $\mathbf{x} \in \mathbb{R}^T_+$, $\langle \nabla f(\mathbf{x}), \mathbf{x} \rangle \leq q f(\mathbf{x})$.

In particular, any polynomial function of $\mathbf{x}$ of degree q will fulfill these requirements.

We formulate the general paging problem as an online convex program. Given a convex function $f : \mathbb{R}^n_+ \to \mathbb{R}$ and a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, a general (offline) convex programming problem is to minimize $f(\mathbf{x})$ subject to $\mathbf{A}\mathbf{x} \geq \mathbf{1}$ and $\mathbf{x} \geq \mathbf{0}$.

In online convex programming, the rows of the constraint matrix $\mathbf{A}$ are revealed one by one, corresponding to the request sequence σ. Upon receiving the tth row $\mathbf{A}_t$ of the constraint matrix, the task of the algorithm is to increase the variables in $\mathbf{x}$ until the constraint $\mathbf{A}_t\mathbf{x} \geq 1$ is fulfilled. The algorithm is never allowed to decrease any of the variables in $\mathbf{x}$.

In the fractional convex program for paging, we denote by p_t the page requested in round t. Furthermore, we let $r(p,t)$ indicate the number of requests to page p up to and including round t, and let $t(p,j)$ be the round during which the page p is requested for the j'th time. As each round corresponds uniquely to a pair (p,j), we have $\sum_p r(p,t) = T$. We let $B(t) = \{p \in P | r(p,t) \geq 1\}$ be the set of distinct pages encountered up to, and including, round t. The variables $\mathbf{x}_{p,j}$ can now take values in the interval $[0,1]$ and indicate the fraction of the page the algorithm has removed from the cache between the j'th and $j+1$'st times it was requested. Using this notation, the convex program for general paging looks as follows:

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & \sum_{p \in P \setminus \{p_t\}} \mathbf{x}_{p,r(p,t)} \geq |B(t)| - k & \forall t \in [T] \\ & 0 \leq \mathbf{x}_{p,j} \leq 1 & \forall p \in [n], j \in [r(p,T)] \end{array} \tag{3.1}$$

By using a convex objective function, this formulation generalizes prior work on online paging, including weighted paging [35]. Crucially, the box constraint $0 \leq \mathbf{x}_{p,j} \leq 1$ means

that the online convex programming framework of Azar et al. [25] can not be used to solve this program.

At the beginning of round t, we are given a new variable $\mathbf{x}_{p_t,r(p_t,t)}$, which is initialized to 0 along with the constraint $\sum_{p\in B(t)\setminus\{p_t\}} \mathbf{x}_{p,r(p,t)} \geq |B(t)| - k$. This constraint ensures that after each round t, at least $|B(t)| - k$ fractional page mass has been ejected, or, equivalently, at most k fractional page mass is inside the cache. We observe that the variable $\mathbf{x}_{p,j}$ will only appear in the constraints corresponding to rounds $t \in \{t(p,j) + 1, t(p,j) + 2, \dots t(p,j+1) - 1\}$, i.e., the variable $\mathbf{x}_{p,j}$ does not appear in round $t(p,j)$ when it is requested. This is because we are not allowed to increase $\mathbf{x}_{p,j}$ during this round, as page p is required to be fully inside the cache in round $t(p,j)$, in order to serve the request.

In order to define a dual for the convex program 3.1, we will need the following definition:

Definition 3.12. *Given a request sequence σ of length T consisting of pages from the set P, we can uniquely, up to relabeling of pages, define a constraint matrix $\mathbf{A} \in \{0,1\}^{T\times T}$ as*

$$\mathbf{A}_{t,(p,j)} = \begin{cases} 1 & \text{if } t \in [t(p,j)+1, t(p,j+1)-1] \\ 0 & \text{otherwise.} \end{cases}$$

In round t, we can determine all non-zero entries, as they only depend on the variables encountered up to round t. Additionally, we can implicitly set the columns corresponding to future variables to 0. If we order both the columns and rows by order of appearance, then the constraint matrix will be lower triangular, see Fig. 3.1 in the appendix.

Definition 3.13. *Let $f : \mathbb{R}_+^T \to \mathbb{R}$ be a convex function. The Fenchel dual $f^* : \mathbb{R}_+^T \to \mathbb{R}$ of f is defined as*

$$f^*(\mathbf{y}) = \sup_{\mathbf{w}\in\mathbb{R}_+^T} \left(\langle \mathbf{w}, \mathbf{y}\rangle - f(\mathbf{w})\right),$$

where $\langle \mathbf{w}, \mathbf{y}\rangle = \sum_{i=0}^{T} \mathbf{w}_i \cdot \mathbf{y}_i$ denotes the Euclidean scalar product.

We need the following property of the Fenchel dual in the analysis of our algorithm:

Property 3.14. *The Fenchel dual $f^*(\mathbf{y}) : \mathbb{R}_+^T \to \mathbb{R}$ of a convex function f is monotonically increasing in $\mathbf{y}$.*

The dual will consist of two sets of T variables each, denoted by $\mathbf{y}_t$ and $\mathbf{z}_{p,j}$, respectively. We let $\mathbf{y}$ be the vector consisting of the $\mathbf{y}_t$ ordered increasingly in t and $\mathbf{z}$ being the vector consisting of the $\mathbf{z}_{p,j}$ ordered the same way as $\mathbf{x}$.

We will use the following *conjugate dual* $D(\mathbf{y},\mathbf{z})$ for our primal-dual algorithm. For the convex primal (3.1), the conjugate dual is:

$$\begin{aligned} \text{maximize} \quad & D(\mathbf{y},\mathbf{z}) = \textstyle\sum_{t=1}^{T}(|B(t)| - k)\mathbf{y}_t - \sum_{p,j} \mathbf{z}_{p,j} - f^*(\mathbf{A}^\top \mathbf{y} - \mathbf{z}), \\ \text{subject to} \quad & 0 \leq \mathbf{y}_t \quad \forall t \in [T] \\ & 0 \leq \mathbf{z}_{p,j} \quad \forall p \in [n], j \in [r(p,T)], \end{aligned}$$

This dual differs from the dual used by Bansal et al. [35] by the inclusion of the Fenchel dual term $f^*(\mathbf{A}^\top \mathbf{y} - \mathbf{z})$ and from the dual used by Azar et al. [25] by the use of non-uniform coefficients for the $\mathbf{y}_t$ variables and the inclusion of the variables $\mathbf{z}_{p,j}$. The dual will only be used to obtain a lower bound on Opt for the analysis of our algorithm, and it does not influence the primal solution the algorithm produces. It remains to show that the stated dual fulfills this property for the convex program (3.1):

Lemma 3.15 (Weak Duality)**.** *For any feasible* $\mathbf{x} \in [0,1]^T$ *and* $\mathbf{y}, \mathbf{z} \in \mathbb{R}_+^T$*, we have*

$$f(\mathbf{x}) \geq \sum_{t \in [T]} (|B(t)| - k)\mathbf{y}_t - \sum_{p,j} \mathbf{z}_{p,j} - f^*(\mathbf{A}^\top \mathbf{y} - \mathbf{z}).$$

Our online algorithm, given in Algorithm 4 uses a continuous time τ, which is 0 initially and increases throughout the algorithm. Let $\tau(t)$ denote the value of τ when we finish processing the t'th constraint and let $\tau(0) = 0$. As all variables are 0 at creation and increase at a rate dependent on τ, we use $\mathbf{x}_{p,j}(\tau)$, $\mathbf{y}_t(\tau)$, $\mathbf{z}_{p,j}(\tau)$ to denote the values of the variables $\mathbf{x}_{p,j}$, $\mathbf{y}_t$, $\mathbf{z}_{p,j}$ respectively at time τ. Let $t(\tau)$ be the unique t such that $\tau \in [\tau(t-1), \tau(t))$. The algorithm uses parameters r and $s_{p,j}(\tau)$ fixed later. Note that $s_{p,j}(\tau)$ depends on the value of τ, p and j and, thus, is not a constant parameter.

Our algorithm maintains dual variables $\mathbf{y}$ and $\mathbf{z}$ such that $\mathbf{A}^\top \mathbf{y} - \mathbf{z} \leq \delta \nabla f(\mathbf{x})$ is approximately fulfilled, i.e., $\mathbf{A}^\top \mathbf{y} - \mathbf{z} \leq r\delta \nabla f(\mathbf{x})$ for some constant $r \geq 1$, which will then appear in the competitive ratio. We observe that if $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$, the gradient is c and we get $\mathbf{A}^\top \mathbf{y} - \mathbf{z} \leq \mathbf{c}$, which is the dual constraint in the linear program for weighted paging. The reason why this point-wise upper bound is necessary is because, together with Property 3.14, it allows us to upper-bound the convex conjugate term $f^*(\mathbf{A}^\top \mathbf{y} - \mathbf{z})$ in $D(\mathbf{y}, \mathbf{z})$ in terms of the primal function $f(\mathbf{x})$. For general $\mathbf{w}$ the conjugate $f^*(\mathbf{w})$ may be arbitrarily large as it is a convex function in $\mathbf{w}$.

Next, to bound the conjugate term in the dual, it is necessary to obtain a bound on the dual "constraints" $\mathbf{A}^\top \mathbf{y} - \mathbf{z}$, which we obtain by relating the constant growth of $\mathbf{y}_t$ and $\mathbf{z}_{p,j}$ to the exponential growth of the $\mathbf{x}_{p,j}$:

Lemma 3.16. *Let* $\bar{\mathbf{x}}$ *denote the value of* $\mathbf{x}$ *after processing the complete request sequence* σ*, and similarly for* $\bar{\mathbf{y}}$ *and* $\bar{\mathbf{z}}$*. If* $s_{p,j}(\tau)$ *is monotonically decreasing in* τ*, then*

$$\bar{\mathbf{x}}_{p,j} \geq \frac{1}{k}\left(\exp\left(\frac{s'_{p,j}}{r}\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right)\right) - 1\right), \tag{3.2}$$

where $s'_{p,j}$ *is the minimum value that* $s_{p,j}(\tau)$ *takes on during the execution of the algorithm.*

The following is an immediate consequence of the previous lemma and the fact that $\mathbf{x}_{p,j} \leq 1$:

Lemma 3.17. *The* $\mathbf{x}(\tau(t))$ *produced by Algorithm 4 throughout its execution are feasible for the primal for all* $t \in [T]$ *and the vector* $(\mathbf{A}^\top \bar{\mathbf{y}} - \bar{\mathbf{z}})_{p,j} \leq \frac{r}{s'_{p,j}} \ln(k+1)$.

Algorithm 4: A fractional algorithm for min-max paging

Require: $r > 0$ and $s_{p,j}(\tau) > 0$, $s_{p,j}(\tau)$ monotonically decreasing in τ

$\tau \leftarrow 0$;

for *each round* $t \in [T]$ **do**

  Let p_t be the page requested in this round;

  $\mathbf{x}_{p_t,r(p,t)}(\tau) \leftarrow 0$, $\mathbf{y}_t(\tau) \leftarrow 0$, $\mathbf{z}_{p_t,r(p,t)}(\tau) \leftarrow 0$;

  $\frac{d\mathbf{y}_t(\tau)}{d\tau} \leftarrow r$;

  $\frac{d\mathbf{x}_{p,r(p,t)}(\tau)}{d\tau} \leftarrow \begin{cases} s_{p,r(p,t)}(\tau)\left(\mathbf{x}_{p,r(p,t)}(\tau) + \frac{1}{k}\right) & \text{if } \mathbf{x}_{p,r(p,t)}(\tau) < 1 \\ 0 & \text{otherwise} \end{cases}$;

  $\frac{d\mathbf{z}_{p,r(p,t)}(\tau)}{d\tau} \leftarrow \begin{cases} r & \text{if } \mathbf{x}_{p,r(p,t)}(\tau) = 1 \\ 0 & \text{otherwise} \end{cases}$;

  Increase τ, $\mathbf{y}_t(\tau)$, $\mathbf{x}_{p,r(p,t)}(\tau)$ and $\mathbf{z}_{p,r(p,t)}(\tau)$ for all $p \in B(t) \setminus \{p_t\}$ simultaneously as per the above differential equations until $\sum_{p \in B(t) \setminus \{p_t\}} \mathbf{x}_{p,r(p,t)}(\tau) \geq |B(t)| - k$;

The conjugate of a convex function with bounded growth can be bounded in terms of the original function and q, using the following lemma:

Lemma 3.18 ([25]). *Let $f : \mathbb{R}^T_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a monotone, convex, differentiable function satisfying $f(0) = 0$. If there is a $q > 1$ such that $\langle \nabla f(\mathbf{x}), \mathbf{x} \rangle \leq q f(\mathbf{x})$, then for any $0 < \gamma < 1$, $\mathbf{y} \in \mathbb{R}^T_{\geq 0}$, $f^*(\gamma \mathbf{y}) \leq \gamma^{\frac{q}{q-1}} \cdot f^*(\mathbf{y})$ and $f^*(\gamma \nabla f(\mathbf{y})) \leq \gamma^{\frac{q}{q-1}}(q-1) f(\mathbf{y})$.*

Theorem 3.19. *Let $f(\mathbf{x})$ be a convex function satisfying the requirements stated at the beginning of this section, and let σ be any request sequence. If we set $s_{p,j}(\tau) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{p,j}}^{-1}$ and $r = \frac{1}{\ln(k+1)(2q\ln(k+1))^{q-1}}$, then Algorithm 4 produces a $(2q\log(k+1))^q$-competitive solution $\bar{\mathbf{x}}$ for fractional paging with objective function $f(\mathbf{x})$ in an online manner.*

Proof. By weak duality, it suffices to show that the primal is no larger than $\mathcal{O}\left((q\log(k+1))^q\right)$ times the dual, which is a lower bound on the cost of an optimal solution $\mathbf{x}^*$ by weak duality. We will bound the primal and the dual growth rates for each round t. It suffices to only consider the case $\mathbf{A}_t \mathbf{x}(\tau) < |B(t)| - k$, as otherwise, the round is finished, and nothing needs to be done.

The processing of round t begins at time $\tau(t-1)$ and will last until $\tau(t)$, so we assume that $\tau \in (\tau(t-1), \tau(t)]$ for the remainder of this proof. Let

$$C_t(\tau) = \{(p,j) \mid t(p,j) < t < t(p,j+1) \text{ and } \mathbf{x}_{p,j}(\tau) < 1\}$$

be the set of indices of variables $\mathbf{x}_{p,j}(\tau)$ in round t which correspond to a page that is (partially) in the cache, i.e., the indices of the variables corresponding to the latest request of a given p, which are increasing and have not been fully removed from the cache. Similarly, let

$$D_t(\tau) = \{(p,j) \mid t(p,j) < t < t(p,j+1) \text{ and } \mathbf{x}_{p,j}(\tau) = 1\}$$

be the set of indices of the variables $\mathbf{x}_{p,j}(\tau)$ which have been fully removed from the cache since they have been last requested and which correspond to the latest request to a given page p. Note that $|C_t(\tau)| + |D_t(\tau)| = |B(t)| - 1$, as the sets $C_t(\tau)$ and $D_t(\tau)$ are disjoint and include a variable for each page except the page p_t. While processing the t-th constraint, we have, by the choice of $s_{p,j}(\tau)$, and the fact that $\mathbf{x}_{p,j}(\tau)$ is constant if $(p,j) \in D_t(\tau)$:

$$G_1 := \frac{df(\mathbf{x})}{d\tau} = \sum_{p,j} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{p,j}} \frac{\partial \mathbf{x}_{p,j}(\tau)}{\partial \tau} = \sum_{(p,j)\in C_t(\tau)} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{p,j}} \left(s_{p,j}(\tau) \left(\mathbf{x}_{p,j}(\tau) + \frac{1}{k} \right) \right) \tag{3.3}$$

$$= \sum_{(p,j)\in C_t(\tau)} \left(\mathbf{x}_{p,j}(\tau) + \frac{1}{k} \right) \tag{3.4}$$

$$\leq |B(t)| - k - |D_t(\tau)| + \frac{|C_t(\tau)|}{k}. \tag{3.5}$$

The first equality is due to the chain rule for vector-valued functions. The second equality uses the definition of $\frac{\partial \mathbf{x}_{p,j}(\tau)}{\partial \tau}$ and the fact that $\mathbf{x}_{p,j}$ does not change for $(p,j) \notin C_t(\tau)$. And, the last inequality follows from the fact that the variables in $D_t(\tau)$ are all equal to 1.

Note that only the y_t corresponding to round t may increase during round t. For the linear term $\sum_t (|B(t)| - k)\mathbf{y}_t(\tau) - \sum_{p,j} \mathbf{z}_{p,j}(\tau)$ in the dual, it holds that in round t

$$G_2 := \frac{d}{d\tau} \left(\sum_t (|B(t)| - k)\mathbf{y}_t(\tau) - \sum_{p,j} \mathbf{z}_{p,j}(\tau) \right) = (|B(t)| - k)r - \sum_{(p,j)\in D_t(\tau)} r \tag{3.6}$$

$$= r(|B(t)| - k - |D_t(\tau)|). \tag{3.7}$$

We note that the right-hand side of Eq. (3.7) is r-times the first term of the right-hand side of Eq. (3.5). Furthermore, $\frac{|C_t(\tau)|}{k} \leq |C_t(\tau)| - k + 1 = |B(t)| - 1 - |D_t(\tau)| - k + 1 = \frac{1}{r} G_2$, since $|C_t(\tau)| \geq k$. By adding together Eq. (3.7) and the last inequality, we obtain

$$G_1 \leq |B(t)| - k - |D_t(\tau)| + |C_t(\tau)|/k \leq G_2/r + G_2/r = 2G_2/r. \tag{3.8}$$

Since both the primal and the linear term of the dual initially have value 0 at time $\tau = 0$, their overall competitive ratio after processing all elements will be $\frac{2}{r}$. Thus for the choice of $r(\delta) = \frac{\delta}{\ln(k+1)}$, where δ is a parameter which we will optimize later, from Eq. (3.8) it follows that $\sum_t (B(t) - k)\bar{\mathbf{y}}_t - \sum_{p,j} \bar{\mathbf{z}}_{p,j} \geq \frac{\delta}{2\ln(k+1)} f(\bar{\mathbf{x}})$. Plugging $s_{p,j}(\tau) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{p,j}}^{-1}$ and $r(\delta) = \frac{\delta}{\ln(k+1)}$ into the second statement of Lemma 3.17, we obtain that $\mathbf{A}^T\mathbf{y} - \mathbf{z} \leq \delta \nabla f(\bar{\mathbf{x}})$, which allows us to bound the conjugate term of the dual as

$$f^*(\mathbf{A}^T\bar{\mathbf{y}} - \bar{\mathbf{z}}) \leq f^*(\delta \nabla f(\bar{\mathbf{x}})) \leq \begin{cases} \delta^{\frac{q}{q-1}} \cdot (q-1) \cdot f(\bar{\mathbf{x}}) & \text{if } q > 1, \\ 0 & \text{if } q = 1, \end{cases}$$

where the first inequality is due to Property 3.14 and the second inequality uses Lemma 3.18. Hence the relationship between the final value $D(\bar{\mathbf{y}}, \bar{\mathbf{z}})$ of the dual and the final value $f(\bar{\mathbf{x}})$ of the primal is

$$\begin{aligned} D(\bar{\mathbf{y}}, \bar{\mathbf{z}}) &= \sum_t (|B(t)| - k)\bar{\mathbf{y}}_t - \sum_{p,j} \bar{\mathbf{z}}_{p,j} - f^*(\mathbf{A}^T\bar{\mathbf{y}} - \bar{\mathbf{z}}) \\ &\geq \left(\frac{\delta}{2\ln(1+k)} - \delta^{\frac{q}{q-1}} \cdot (q-1) \right) \cdot f(\bar{\mathbf{x}}). \end{aligned}$$

The term $h(\delta) = \left(\frac{\delta}{2\ln(k+1)} - \delta^{\frac{q}{q-1}} \cdot (q-1) \right)$ is a polynomial in δ, which governs our competitive ratio. The best competitive ratio is obtained if we find $\delta \in (0,1)$ such that $h(\delta)$ is maximized. We find a local maximum at $\delta^* = \frac{1}{(2q\ln(k+1))^{q-1}}$, yielding $h(\delta^*) = \frac{1}{(2q\ln(k+1))^q}$. By rearranging and weak duality (Lemma 3.15) we obtain

$$f(\bar{\mathbf{x}}) \leq (2q\ln(k+1))^q \, D(\bar{\mathbf{y}}, \bar{\mathbf{z}}) \leq (2q\ln(k+1))^q \, f(\mathbf{x}^*),$$

where $\mathbf{x}^*$ is an optimal solution. □

The L_q-norm does not lie in our class of objective functions, as a coordinate of $\nabla f(\mathbf{x})$ can decrease while we increase all coordinates of $\mathbf{x}$, hence we can not apply Theorem 3.19 straight away.

Theorem 3.20. *Let $q \in [1, \infty)$. Then there exists a $2q\log(k+1)$-competitive algorithm for fractional q-paging with a cache of size k.*

Proof. Let us fix $q \in [1, \infty)$. We apply Theorem 3.19 with the target function

$$f(\mathbf{x}) = \sum_{p \in P} \left(\sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j} \right)^q,$$

which is the qth power of the L_q-norm. This produces a solution $\bar{\mathbf{x}}$, which is $(2q\log(k+1))^q$-competitive for the paging problem with target function $f(\mathbf{x})$.

Let $g : \mathbb{R} \to \mathbb{R}$ be a monotone function, then a solution $\mathbf{x}$ to the paging problem with target function $f(\mathbf{x})$ will also be a feasible solution to the paging problem with target function $g(f(\mathbf{x}))$. In particular, as g preserves the standard ordering on the reals, an optimal solution to paging with target function $f(\mathbf{x})$ will remain an optimal solution to the problem with target function $g(f(\mathbf{x}))$.

If we let $g(y) = y^{\frac{1}{q}}$ and we let $\mathbf{x}^*$ be an optimal solution to the paging problem with target function $f(\mathbf{x})$, then $g(f(\mathbf{x}))$ will be the L_q-norm and we find that

$$g(f(\bar{\mathbf{x}})) \leq g((2q\log(k+1))^q f(\mathbf{x}^*)) = 2q\log(k+1) g(f(\mathbf{x}^*)),$$

which establishes our theorem. □

Remark. *Note that if the gradient of $f(\mathbf{x})$ is 0 at $\mathbf{x} = 0$, then we start the algorithm at $\varepsilon \cdot \mathbf{1}$ for a small $\varepsilon > 0$ instead, which can be chosen sufficiently small, so it does not influence the competitive ratio.*

We use Theorem 3.20 to show that we can obtain a $2e\log(n)\log(k+1)$-competitive fractional solution for ∞-paging by reducing it to $\log n$-paging.

Theorem 3.21. *There exists a $2e\log(n)\log(k+1)$-competitive algorithm for fractional min-max paging.*

Proof. Let $\mathbf{x}^*$ be the optimal solution to the ∞-paging problem for the request sequence σ. We denote the cost of this solution by Opt_∞. Let $\bar{\mathbf{x}}$ denote the fractional solution obtained using Algorithm 4. By Theorem 3.20 and $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_{\log n} \leq e\|\mathbf{x}\|_\infty$, we know that this solution has cost

$$\begin{aligned}\max_{p\in P}\sum_{j=1}^{r(p,T)}\bar{\mathbf{x}}_{p,j} &\leq \left(\sum_{p\in P}\left(\sum_{j=1}^{r(p,T)}\bar{\mathbf{x}}_{p,j}\right)^{\log n}\right)^{\frac{1}{\log n}}\\ &\leq 2\log(n)\log(k+1)\cdot\mathsf{Opt}_{\log n}\\ &\leq 2e\log(n)\log(k+1)\cdot\mathsf{Opt}_\infty,\end{aligned}$$

where $\mathsf{Opt}_{\log n}$ denotes the cost of an optimal solution to the $\log n$-paging problem with input σ. This implies that $\bar{\mathbf{x}}$ is a $2e\log(n)\log(k+1)$-competitive solution for ∞-paging. □

3.5 Rounding Fractional Solutions Online

3.5.1 An $\mathcal{O}(k\log(n)\log(k))$-competitive Deterministic Algorithm

This section shows how to round a fractional solution for min-max paging to an integral solution online. The rounding procedure is deterministic and, when coupled with a fractional min-max paging algorithm, gives a deterministic min-max paging algorithm.

Theorem 3.22. *If there exists an α-competitive algorithm for fractional min-max paging with cache size k, then there exists a (αk)-competitive deterministic algorithm for min-max paging with cache size k.*

Proof. Without loss of generality, we assume that the fractional min-max paging algorithm is lazy. That is, it loads a page only when the page is requested. Indeed, an arbitrary solution can be converted into a lazy solution online without increasing the cost by delaying page loads as much as possible.

The deterministic integral algorithm maintains the following invariant: it always has a page p in its cache whenever the fractional algorithm has more than a $1 - 1/k$ fraction of p in its cache. We observe that the fractional algorithm must always fully have at least one page in its cache: the most recently requested page. Therefore, at any time, the number of pages p such that the fractional algorithm contains more than a $1 - 1/k$

fraction of p is less than $1 + (k-1)/(1-1/k) = k+1$, and therefore, this number is at most k.

Consider an arbitrary request to some page p. If the integral algorithm already has p in its cache, it ignores the request, whereas the fractional algorithm possibly serves the request by evicting some pages fractionally. On the other hand, suppose the integral algorithm does not already have p in its cache, then this implies that the fractional algorithm has at most a $1-1/k$ fraction of p in its cache. After the fractional algorithm brings p into its cache, the integral algorithm must have a page q in its cache such that the fractional algorithm has at most a $1-1/k$ fraction of q in its cache. (Otherwise, the fractional algorithm has more than a $1-1/k$ fraction of $k+1$ pages in its cache, namely, the k pages in the integral algorithm's cache and the page p, thus contradicting the observation from the last paragraph.) The integral algorithm replaces one such page q by p to serve the request and thus, maintains the invariant. In this process, the integral and the fractional algorithms incur 1 and at least $1/k$ faults, respectively, on page p.

Thus, at the end of the request sequence, for every page p, the number of faults of the integral algorithm on p is at most k times the number of faults of the fractional algorithm on p. Thus, the cost of the integral algorithm is at most k times the cost of the fractional algorithm. Since the latter is at most α times the cost of the optimum, the cost of the integral algorithm is at most αk times the cost of the optimum solution. □

Corollary 3.23. *There exists a $2ek\log(n)\log(k+1)$-competitive deterministic algorithm for min-max paging.*

Proof. Follows from Theorems 3.21 and 3.22. □

It is noteworthy that the trick in the proof of Theorem 3.22 can also be used for the derandomization of randomized algorithms. Specifically, suppose an α-competitive randomized algorithm exists for min-max paging. Then there also exists a fractional one with the same competitive ratio. Thus, by Theorem 3.22, there exists a αk-competitive deterministic algorithm for min-max paging.

3.5.2 An $\mathcal{O}\left(\log^2(n)\log(k)\right)$-competitive Randomized Algorithm.

Using a more sophisticated rounding approach, we obtain a randomized algorithm whose competitive ratio no longer depends linearly on k, in exchange for an additional $\log(n)$ factor. This result rules out a lower bound of $\Omega(k)$. This algorithm is of interest in the regime where $\log(n) \leq k$, which is often the case in applications.

We can obtain a randomized algorithm for min-max paging by using the rounding scheme for weighted paging of Bansal et al. [36]. The simplified rounding scheme is presented in Algorithm 5. Each online rounding step only depends on the previous, and current fractional cache states $\mathbf{x}(t-1)$ and $\mathbf{x}(t)$ as well as the previous integral cache state and on a parameter β, which indicates how aggressively we eject pages from the cache. The rounding scheme works for any caching scheme that fulfills the condition that

1. at any time t, for any page $p \neq p_r$, $\mathbf{x}_p(t) - \mathbf{x}_p(t-1) \geq 0$, and

2. the total fraction of pages evicted upon any request is at most 1.

Algorithm 4 indeed has these properties, so we can use the rounding scheme as long as we can relate the rounding costs to our target function, even though we solve a different paging problem than they do.

Let $\mathbf{x}$ be a fractional solution produced by Algorithm 4. After processing round t, the algorithm will produce a fractional value $\mathbf{x}_p(t)$ for each page, indicating the fraction of page p in the cache in this round. In other words, the process of solving the fractional problem online produces, whenever Algorithm 4 finishes processing a round at time $\tau(t)$, the vector

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{x}_{p_1,r(p,t)}(\tau(t)) \\ \vdots \\ \mathbf{x}_{p_n,r(p,t)}(\tau(t)) \end{bmatrix}.$$

We let $\mathbf{y}_p(t) = \min\{\beta \cdot \mathbf{x}_p(t), 1\}$ be the solution in which every coordinate is scaled up by a factor of β. The factor β governs how much more aggressively pages are evicted from the cache.

Algorithm 5 may evict pages and incur costs in two separate places. The first type we need to account for is the cost incurred via the random evictions of pages in the for-loop in lines 4-5 of the algorithm. The second type is the cost incurred by fixing the cache size in lines 6-7 if no page was evicted in the for-loop. We will bound these costs separately and combine them in our upper bound.

Algorithm 5: The randomized rounding scheme of [36] adapted to our problem.

Input: $\mathbf{x}(t)$, $\mathbf{x}(t-1)$, and $C(t-1)$

`/* Add the page` p_t `to the cache, if it is not already in it. */`

1 **if** $p_t \notin C(t-1)$ **then**
2 $\quad C(t-1) \leftarrow C(t-1) \cup \{p_t\}$;
3 **for** $p \in C(t-1) \setminus \{p_t\}$ **do**
4 $\quad$ Evict p from $C(t-1)$ independently with probability $\frac{\mathbf{y}_p(t)-\mathbf{y}_p(t-1)}{1-\mathbf{y}_p(t-1)}$;
5 **if** $|C(t-1)| > k$ **then**
6 $\quad$ Evict an arbitrary page $p \neq p_t$ from $C(t-1)$;
7 $C(t) \leftarrow C(t-1)$;

For the first type, it is easy to see that the cost incurred for evicting a page in lines 4-5 depends only on the sequence of fractional values $\mathbf{y}_p(1), \mathbf{y}_p(2), \ldots, \mathbf{y}_p(T)$ that this page takes on and it is independent of the values $\mathbf{y}_{p'}(t)$ for all t and $p' \neq p$. In particular, the

probability $\mathbf{y}_{p,j}$ that a page is evicted in lines 4-5, between its j-th and $j+1$-st request is

$$\begin{aligned}
&\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \mathbf{Pr}\left[\text{page } p \text{ is evicted in round } t\right] \\
&\quad= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{\mathbf{y}_p(t)-\mathbf{y}_p(t-1)}{1-\mathbf{y}_p(t-1)} \mathbf{Pr}\left[\text{page } p \text{ is not evicted until round } t\right] \\
&\quad= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{\mathbf{y}_p(t)-\mathbf{y}_p(t-1)}{1-\mathbf{y}_p(t-1)} \prod_{t'=t(p,j)+1}^{t-1} 1-\frac{\mathbf{y}_p(t')-\mathbf{y}_p(t'-1)}{1-\mathbf{y}_p(t'-1)} \\
&\quad= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{\mathbf{y}_p(t)-\mathbf{y}_p(t-1)}{1-\mathbf{y}_p(t-1)} \prod_{t'=t(p,j)+1}^{t-1} \frac{1-\mathbf{y}_p(t')}{1-\mathbf{y}_p(t'-1)} \\
&\quad= \sum_{t=t(p,j)+1}^{t(p,j+1)-1} \frac{\mathbf{y}_p(t)-\mathbf{y}_p(t-1)}{1-\mathbf{y}_p(t(p,j))} = \frac{\mathbf{y}_p(t(p,j+1)-1)}{1-\mathbf{y}_p(t(p,j))} = \mathbf{y}_p(t(p,j+1)-1).
\end{aligned}$$

The second equation holds because of the independence of the probability of eviction in different rounds; the fourth holds because it is a telescoping product, and the last equation holds as $\mathbf{y}_p(t(p,j))=0$. Let $Y_{p,j}$ be a Bernoulli random variable that is 1 with probability $\mathbf{y}_p(t(p,j+1)-1)$ and let $Y_p=\sum_j Y_{p,j}$ be the sum of all $Y_{p,j}$ for fixed p. We let these variables track the expected cost of evictions for each page. By linearity of expectation, we immediately see that

$$\begin{aligned}
\mathbb{E}\left[Y_p\right] = \mathbb{E}\left[\sum_{j=1}^{r(p,T)} Y_{p,j}\right] &= \sum_{j=1}^{r(p,T)} \mathbf{Pr}\left[p \text{ is evicted between request } j \text{ and } j+1\right] \\
&= \sum_{j=1}^{r(p,T)} \mathbf{y}_{p,j} \leq \beta \sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j}.
\end{aligned}$$

It follows that

$$\max_p \mathbb{E}\left[Y_p\right] \leq \beta \max_p \sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j},$$

where the right-hand side is β times the cost of the fractional solution $\mathbf{x}$. It remains to relate the left side of this inequality with $\mathbb{E}\left[\max_p Y_p\right]$.

Lemma 3.24. *Let $Y_{p,j}$ be Bernoulli random variables which are 1 with probability $\mathbf{y}_{p,j}$. Let $Y_p=\sum_j Y_{p,j}$ and assume there are n such sums, then*

$$\mathbb{E}\left[\max_p Y_p\right] \leq e \cdot \max_p \mathbb{E}\left[Y_p\right] + \log(n).$$

Proof. Let $Y = \max_p Y_p$. Using Jensen's inequality, we get the first inequality in the following chain of inequalities:

$$\begin{aligned}\exp(\mathbb{E}[Y]) &\leq \mathbb{E}[\exp(Y)] = \mathbb{E}\left[\max_p \exp(Y_p)\right] \\ &\leq \sum_p \mathbb{E}[\exp(Y_p)] = \sum_p \prod_{j=1}^{r(p,T)} \mathbb{E}[\exp(Y_{p,j})] \\ &= \sum_p \prod_{j=1}^{r(p,T)} (1 - \mathbf{y}_{p,j} + e\mathbf{y}_{p,j}) \leq \sum_p \prod_{j=1}^{r(p,T)} e^{e\cdot\mathbf{y}_{p,j}} \\ &= \sum_p e^{e\cdot\sum_{j=1}^{r(p,T)}\mathbf{y}_{p,j}} \leq n \cdot \max_p e^{e\cdot\mathbb{E}[Y_p]}.\end{aligned}$$

The first equality follows as exp is a monotone function, and the second equality follows by the independence of the $Y_{p,j}$. After taking logarithms, we obtain

$$\mathbb{E}[Y] \leq e \cdot \max_p \mathbb{E}[Y_p] + \log(n),$$

which yields the desired result. □

Therefore, by the above lemma, the expected cost of the first type of costs is bounded by

$$\mathbb{E}\left[\max_p Y_p\right] \leq \mathcal{O}(1) \cdot \beta \max_p \sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j}. \tag{3.9}$$

Lemma 3.25 ([36])**.** *Let* $\mathbf{x}$ *be a fractional solution for a general paging problem. The expected cost of resets is at most* $16ke^{-\beta/4} \cdot \sum_{p\in P}\sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j}$.

By choosing $\beta = 4\log(nk)$ in Lemma 3.25, the expected total cost of resets for the solution y becomes

$$16ke^{-\log(nk)} \sum_{p\in P}\sum_{j=1}^{r(p,T)} \mathbf{y}_{p,j} \leq \frac{16}{n}\sum_{p\in P}\sum_{j=1}^{r(p,T)} \mathbf{y}_{p,j} \tag{3.10}$$

$$\leq \frac{16\beta}{n}\sum_{p\in P}\sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j} \leq 16\beta \max_{p\in P}\sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j} \tag{3.11}$$

where the last inequality follows due to the fact that the average cost per page $\frac{1}{n}\sum_{p\in P}\sum_{j=1}^{r(p,T)} \mathbf{x}_{p,j}$ is a lower bound on the maximum cost of a single page in the solution $\mathbf{x}$.

Crucially, Lemma 3.25 depends on the following helper lemma:

Lemma 3.26. *Given a fractional solution* $\mathbf{x}$ *to the paging problem, we can find a fractional solution* $\mathbf{x}^*$ *in which every variable is a multiple of* $\delta = \frac{1}{4k}$*, and the cost of which is no more than* 3 *times the cost of* $\mathbf{x}$.

Taking the bounds on the two types of costs, namely Eqs. (3.9) and (3.11), we have shown the following:

Theorem 3.5. *There exists an* $\mathcal{O}\left(\log^2 n \log k\right)$*-competitive* randomized *integral algorithm for min-max paging.*

3.6 Further related work

Sleator and Tarjan [295] defined the framework of online algorithms and competitive analysis, and paging is one of the earliest problems studied in the online setting. Several deterministic algorithms, such as "Least Recently Used" (LRU) and "First In First Out" (FIFO), among others, are known to achieve the optimal deterministic competitive ratio of k [295], where k is the maximum number of pages that can be inside the cache at any point in time. The randomized competitive ratio is known to be H_k, where the upper bound is due to Achlioptas et al. [3] and the lower bound is due to Fiat et al. [127].

Several practical generalizations of the paging problem have been studied and they are known to have a deterministic competitive ratio of k [81, 328] and randomized competitive ratio $\Theta(\log k)$ [35, 34]. These include weighted paging – where pages have arbitrary loading costs, the *bit model* – where pages have arbitrary sizes and loading cost proportional to size, the *fault model* – where pages have arbitrary sizes but unit loading cost, and generalized paging – where pages have arbitrary loading costs as well as sizes. Interestingly, all these results are robust in the sense that they all extend to the resource-augmentation setting, where the adversary has fewer servers than the algorithm. It is noteworthy that the line of work in search of a randomized algorithm for these paging variants by Bansal, Buchbinder, and Naor led to the development of the online primal-dual framework for designing fractional algorithms for online problems, whose solutions can often be rounded to an integral solution online.

A simple-looking but intriguing generalization of paging is the k-server problem defined by Manasse, McGeogh, and Sleator [221], which concerns moving k mobile servers on a metric space to serve requests while minimizing total movement. (The paging problem is the k-server problem on the uniform metric over the set of pages.) While Manasse et al. [221] proved a lower bound of k on the deterministic competitive ratio for every metric space with more than k points, the existence of a k-competitive algorithm is still unknown, and this is popularly called the k-server conjecture. The best-known k-server algorithm that works for all metrics called the *Work Function Algorithm* by Koutsoupias and Papadimitriou [199], achieves a competitive ratio of $2k - 1$. For randomized algorithms, surprisingly, neither a better upper bound than the deterministic $2k - 1$ nor a better lower bound of $\Omega(\log k)$ arising from paging is known. Koutsoupias [198] presents a more comprehensive discussion on the k-server problem.

3.7 Further Details on the Lower Bounds

This section provides further details for the deterministic lower bound for min-max paging. We begin with a proof of the lower bound for $k = 2$, which neatly highlights the core construction lying at the heart of the lower bound.

Lemma 3.27. *Any deterministic algorithm* ALG *for min-max paging with cache size* $k = 2$ *is at least* $\Omega(\log n)$*-competitive.*

Proof. Suppose $n = 3^\ell$ and let the set of pages be $\{p_1, p_2, \ldots, p_n\}$. We construct a bad request sequence σ for ALG in ℓ layers. Each layer is further divided into phases. Let $N \gg n$ be a large integer parameter.

We call the number of page faults incurred by page p up to round t the *cost* of p at t. Similarly, the *cost of the min-max paging algorithm* ALG is the maximum cost over all pages $p \in P$.

We now iteratively construct the adversarial sequence σ, going layer by layer.

Layer 1 will use all pages, that is the set $\{p_1, p_2, \ldots, p_n\}$.

- In the first phase, we request pages p_1, p_2, and p_3 in such a way that ALG faults on every request, such a cruel sequence composed of $k + 1$ pages exists for every deterministic algorithm for paging. We stop this phase once the cost of the algorithm becomes N, which must happen before sending $3N$ requests. Without loss of generality, we assume that the cost of p_1 first reaches N, and hence the costs of p_2 and p_3 are $< N$. These costs are the same as the number of requests to the respective pages because the algorithm always faults.

- In the second phase, we repeat this step with pages p_4, p_5, and p_6. Without loss of generality, we assume that the cost of p_4 is N.

- Repeat this process until the set of pages $\{p_1, p_2, \ldots, p_n\}$ is exhausted.

In all phases, the adversary always keeps the lowest numbered page, that is pages $p_1, p_4, p_7, \ldots$, respectively, in the cache, incurring a cost of only 1 on them, while the cost of the other pages is at most N. We *promote* pages $p_1, p_4, p_7, \ldots$ to Layer 2.

Layer 2 with universe of pages $\{p_1, p_4, p_7, ..., p_{n-2}\}$ is constructed exactly in the same way as Layer 1. After this layer, the cost of ALG is $2N$, whereas the cost of the adversary is $\leq N$, with costs of pages $p_1, p_{10}, p_{19}, \ldots$ having cost ≤ 2, and we *promote* them to Layer 3, and so on.

After phase i, the number of pages in the universe becomes $n/3^i$, the cost of ALG becomes iN, whereas the adversary's cost is always $\leq \max\{N, i\}$. This gives us the desired lower bound using $\log_3(n)$ layers by choosing $N \geq \ell$. □

Generalizing the above construction The idea behind the lower bound of $\Omega\left(\frac{k(\log n)}{\log k}\right)$ is as follows. We generalize the construction above by using $n = (k+1)^\ell$ pages and $\ell = \log_{k+1} n$ layers. In each phase, we use $k+1$ pages and force the cost of the algorithm

on one of these pages to increase by N. The adversary's cost increases by at most 1 on the page she will promote to the next layer. By using a smarter *offline* algorithm, the cost of the adversary increases by at most $\mathcal{O}(N/k)$ on the k pages of this phase that will not be promoted. So, in the end, the adversary's cost is $\mathcal{O}(\ell + N/k)$, whereas the cost of the algorithm is $\Omega(N\ell)$. We obtain the desired lower bound by choosing $N \geq ck\ell$ for a large enough constant c.

For any paging algorithm ALG and any request sequence σ, we define $\text{cost}(\text{ALG}, \sigma, p, t)$ to be the the number of page faults incurred on page p after processing the first t requests of σ. Furthermore, we define

$$\text{cost}(\text{ALG}, \sigma) = \max_{p \in P} \text{cost}(\text{ALG}, \sigma, p, T),$$

to be the overall cost incurred by ALG while processing request sequence σ.

The optimal offline algorithm Optfor min-max paging is not known to us, so we use Algorithm 6 (GreedyLFD) to obtain an upper bound on the cost of Opt. Intuitively, this algorithm avoids increasing its maximum cost for as long as possible by greedily keeping the most expensive pages in its cache. As GreedyLFD is an offline algorithm, it has access to the complete request sequence σ and can always eject the page that is *furthest in the future*. That is, in round t, it ejects the page $p \in C$ whose next occurrence comes last in the remainder of σ after t. If a page does not occur in the remainder of σ, it is treated as being infinitely far in the future, and the algorithm will always prefer to eject this page over one that will still occur in σ.

Lemma 3.28. *Let σ be a request sequence for min-max paging using $k+1$ unique pages. Then*

$$\text{cost}(\text{GreedyLFD}, \sigma) \leq \frac{2(\text{len}(\sigma) - 2k - 1)}{2k + k(k+1)} + 2.$$

Proof. We fix σ to be an arbitrary request sequence of length T using pages from the set $P = \{p_1, p_2, \ldots, p_{k+1}\}$. Let

$$t_i = \min \left\{ t \in [T] \mid \max_{p \in P} \text{cost}(\text{GreedyLFD}, \sigma, p, t) \geq i \right\}$$

be the first time GreedyLFD faults on a page for the ith time. We note that at time t_i, there is only one page of cost i.

Furthermore, we note that $t_1 = 1$, as the algorithm starts with an empty cache and $t_2 \geq 2k+1$, as the algorithm will fault on the first $k+1$ distinct pages it encounters, and it will then eject the page of cost 1 which occurs farthest in the future. As there are k pages in its cache, at least one page will not occur in the next $k-1$ time steps, so the shortest sequence that can cause GreedyLFD to fault two times on a single page is of length $2k+1$.

We now show $t_{i+1} \geq t_i + k + \frac{k(k+1)}{2}$ for all $i \geq 2$. Let us fix $i \geq 2$ and assume we are currently at time t_i. This means that there exists some page $p \in P$ for which $\text{cost}(\text{GreedyLFD}, \sigma, p, t_i) = i$ and it is the only page of cost i. In order to make room

Algorithm 6: The offline algorithm GreedyLFD

```
   Input: σ
1  C ← ∅;                                              // Initialize the cache.
2  t ← 1;
3  for p ∈ P do
4  |  c_p ← 0;      // Counter variables for the number of faults on page p.
   |   At any point in time c_p = cost(GreedyLFD, σ, p, t).
5  while t < T do
6  |  if the tth element p_t of σ is not in C then
7  |  |  c_p ← c_p + 1;
8  |  |  if |C| < k then
9  |  |  |  C ← C ∪ {p};
10 |  |  else
11 |  |  |  S ← {q ∈ C | c_q < max_r c_r};      // Obtain the set of pages whose
   |  |  |   cost is less than the current maximum.
12 |  |  |  if S = ∅ then // This happens if all pages in C are of the
   |  |  |   same cost.
13 |  |  |  |  S ← C;
14 |  |  |  Evict q ∈ S which next occurs farthest in the future;
15 |  |  t ← t + 1;
```

for p, GreedyLFD will evict a page of cost at most $i-1$ from its cache. As there are k such pages in the cache at time t_i, at least one of them will not occur for the next $k-1$ requests. As there are only $k+1$ pages in total, this means that the next page fault occurs at time $t_i + k$ or later.

In general, when the jth page of cost i is added to GreedyLFD's cache at time $t_{i,j}$, there are $k-j+1$ pages of cost at most $i-1$ in its cache, and so the next page fault will not occur until time $t_{i,j} + k - j + 1$, which gives a lower bound on $t_{i,j+1}$. Note that $t_{i,1} = t_i$

As GreedyLFD's cost can only increase to $i+1$ once it evicts a page of cost i, we find that t_{i+1} must occur after k pages of cost i have been added to its cache. GreedyLFD can choose which of the k pages of cost i to evict, so we find $t_{i+1} \geq t_{i,k} + k$. This yields

$$t_{i+1} \geq t_{i,k} + k \geq t_{i,k-1} + k + 1 \geq t_{i,k-2} + k + 1 + 2 \geq \cdots \geq t_i + k + \sum_{i=1}^{k} i.$$

By expanding the recurrence for $i \geq 2$, we find that

$$t_i \geq (i-2)\left(k + \frac{k(k+1)}{2}\right) + 2k + 1.$$

Using this expression, we derive an upper bound on $\text{cost}(\text{GreedyLFD}, \sigma)$, by finding the minimum i for which $\text{len}(\sigma) \leq t_i$. From our expression we find that $t_i \geq \text{len}(\sigma)$ if $i \geq \frac{2(T-2k-1)}{2k+k(k+1)} + 2$, and so

$$\text{cost}(\text{GreedyLFD}, \sigma) \leq \frac{2(\text{len}(\sigma) - 2k - 1)}{2k + k(k+1)} + 2.$$

□

Lemma 3.29. *Any deterministic algorithm ALG for min-max paging with cache size k can not have competitive ratio better than $\frac{k}{2}$.*

Proof. Let $n = k+1$, that is $P = \{p_1, p_2, \ldots, p_{k+1}\}$. We initialize ALG and GreedyLFD with empty caches. Once ALG's cache is full, at any time step t, there is always one page that is not present in the cache. The adversary's strategy is always to request this page. We call this the *cruel* strategy. Since ALG is deterministic, the adversary always knows which page ALG will evict from its cache if a page fault occurs, so such a sequence must always exist.

Let σ be a sequence of length T generated by the cruel strategy of Algorithm 7. We note that σ causes a page fault at every step, so the total number of page faults incurred by ALG will be T. By a simple averaging argument, there must be at least one page that has incurred $\frac{T}{k+1}$ page faults and so $\frac{T}{k+1} \leq \mathit{cost}(\text{ALG}, \sigma)$. On the other hand, by Lemma 3.28, GreedyLFD will incur a cost of at most $\frac{2(T-2k-1)}{2k+k(k+1)} + 2$ while processing σ.

Algorithm 7: An algorithm to generate an adversarial sequence for ALG of length $T \geq k+1$, on which ALG faults T times.

```
Output pages p_1, p_2, ..., p_{k+1};
i ← k + 1;
while i < T do
    p ← the page p_i which is currently not in the cache of ALG;
    Output p;
    i ← i + 1;
```

This immediately yields

$$\begin{aligned}\text{cost}(\mathsf{Opt}, \sigma) \leq \text{cost}(\textsc{GreedyLFD}, \sigma) &\leq \frac{2(T-2k-2)}{2k+k(k+1)} + 2 \\ &\leq \frac{2T}{k(k+1)} + 2 \leq \frac{2}{k}\,\text{cost}(\text{ALG}, \sigma) + 2,\end{aligned}$$

and so the competitive ratio is $\frac{cost(\text{ALG},\sigma)}{\text{cost}(\mathsf{Opt},\sigma)} \geq \frac{k}{2}$, since the constant vanishes as the cost grows large. □

Finally, we strengthen the lower bound to $\Omega\left(\frac{k}{\log k}\log n\right)$, introducing a dependence on the number of pages. We do this using the strategy presented in Algorithm 8.

Algorithm 8: An adversarial strategy for min-max paging

```
Let L_m = {p_0^m, ..., p_{n-1}^m} be a set of n = (k+1)^m pages;
for ℓ = m, ..., 1 do
    L_{ℓ-1} ← ∅;
    for i = 0, ..., (k+1)^{ℓ-1} − 1 do
        Use the cruel strategy of Algorithm 7 on the set of k+1 pages
          {p^ℓ_{(k+1)i}, p^ℓ_{(k+1)i+1}, ..., p^ℓ_{(k+1)i+k}} until one page p' has incurred N page
          faults since the start of this loop.;
        p_i^{ℓ-1} ← p';
        L_{ℓ-1} ← L_{ℓ-1} ∪ {p_i^{ℓ-1}};
```

Intuitively, our strategy consists of splitting the $n = (k+1)^m$ pages into $(k+1)^{m-1}$ disjoint sets of $k+1$ variables. We then present the algorithm ALG with a cruel sequence for each set until one of the pages reaches cost N. We repeat this process layer by layer until we obtain one final page. ALG will have faulted mN times on this page, while Optwill have faulted no more than roughly $\frac{N}{k} + m$ times on any page.

Theorem 3.8. *Any deterministic algorithm for min-max paging with cache size k is at least $\frac{k-1}{2}\log_{k+1} n$-competitive, where n is the number of pages.*

Proof. We use the strategy defined in Algorithm 8 to generate our request sequence. We observe that at the beginning of iteration ℓ of the outer for-loop in Algorithm 8, ALG will have faulted $(m-\ell)N$ times on each page in $\{p_0^\ell, \dots, p_{(k+1)^\ell-1}^\ell\}$, because we only add a page to the next level once it has incurred N faults during the current level. Hence, once ALG has processed the complete sequence provided by Algorithm 8, it has cost mN, witnessed by page p_0^0.

On the other hand, while processing the ith set of variables in the inner loop, the optimal offline algorithm Optwill keep page p' in its cache. When p' is requested for the first time in this iteration of the loop, Optwill fault once on p'. Afterward, p' will remain in the cache of Optuntil the current iteration of the inner loop finishes, incurring no more cost.

This shows that Optwill have cost $m-\ell$ for each page $p_0^\ell, \dots, p_{(k+1)^\ell-1}^\ell$ at the beginning of the ℓth iteration of the outer loop.

During an iteration of the inner loop, we use the remaining $k-1$ slots in Opt's cache, which are not occupied by p', to process the remaining k pages in each iteration. As the cruel sequence causes ALG to fault on every request and we end it as soon as one page has faulted N times, each of the remaining pages may be requested $N-1$ times. It follows that the sub-sequence σ' of the cruel sequence, defined on the remaining k pages, is of length at most $k(N-1)$. By Lemma 3.28, we get

$$\text{cost}(\mathsf{Opt}, \sigma') \leq \frac{2(k(N-1)-2(k-1)-1)}{2(k-1)+(k-1)k} + 2 \leq \frac{2(N-1)}{k-1} + 2.$$

Thus we find that for any page p, the total cost consists of the level it is raised to plus the cost incurred while processing σ' on its last level and thus $\text{cost}(\mathsf{Opt}, \sigma, p, T) \leq m + \frac{2(N-1)}{k-1} + 2$ and so

$$\frac{\text{cost}(\text{ALG}, \rho)}{\text{cost}(\mathsf{Opt}, \rho)} \geq \frac{mN}{m+2+\frac{2(N-1)}{k-1}} = \frac{(k-1)mN}{(k-1)(m+2)+2(N-1)}.$$

As N grows large, the right hand side will converge to $\frac{(k-1)m}{2} = \frac{k-1}{2}\log_{k+1} n$. □

3.8 Deferred Proofs

This section contains some deferred proofs from earlier in the chapter.

Property 3.14. *The Fenchel dual $f^*(\mathbf{y}) : \mathbb{R}_+^T \to \mathbb{R}$ of a convex function f is monotonically increasing in $\mathbf{y}$.*

Proof. Indeed, let $\mathbf{c} \in \mathbb{R}_+^T$, then

$$f^*(\mathbf{y}) = \sup_{\mathbf{w}\in\mathbb{R}_+^T} \langle \mathbf{y}, \mathbf{w}\rangle - f(\mathbf{w}) \leq \sup_{\mathbf{w}\in\mathbb{R}_+^T} \langle \mathbf{y}, \mathbf{w}\rangle + \langle \mathbf{c}, \mathbf{w}\rangle - f(\mathbf{w}) = f^*(\mathbf{y}+\mathbf{c}),$$

as $\langle \mathbf{c}, \mathbf{w}\rangle$ is always non-negative. □

Lemma 3.15 (Weak Duality)**.** *For any feasible* $\mathbf{x} \in [0,1]^T$ *and* $\mathbf{y}, \mathbf{z} \in \mathbb{R}_+^T$*, we have*

$$f(\mathbf{x}) \geq \sum_{t \in [T]} (|B(t)| - k)\mathbf{y}_t - \sum_{p,j} \mathbf{z}_{p,j} - f^*(\mathbf{A}^\top \mathbf{y} - \mathbf{z}).$$

Proof. Let $\mathbf{b} = \begin{bmatrix} B(1) - k \\ \vdots \\ B(T) - k \end{bmatrix}$. As $\mathbf{x}$ is feasible, it must satisfy $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, which gives $\mathbf{b} - \mathbf{A}\mathbf{x} \leq \mathbf{0}$, and similarly from $\mathbf{x} \leq \mathbf{1}$, we get $\mathbf{x} - \mathbf{1} \leq \mathbf{0}$. As all entries are negative, taking the inner product of these vectors with the non-negative vectors $\mathbf{y}$ and $\mathbf{z}$, respectively, will yield a negative number. Hence, we get the first inequality in the following chain of inequalities:

$$\begin{aligned}
f(\mathbf{x}) &\geq f(\mathbf{x}) + \mathbf{y}^\top((|B(t)| - k)\mathbf{1} - \mathbf{A}\mathbf{x}) + \mathbf{z}^\top(\mathbf{x} - \mathbf{1}) \\
&= \sum_{t=1}^{T} (|B(t)| - k)\mathbf{y}_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} \mathbf{z}_{p,j} - (\mathbf{y}^\top \mathbf{A}\mathbf{x} - \mathbf{z}^\top \mathbf{x} - f(\mathbf{x})) \\
&= \sum_{t=1}^{T} (|B(t)| - k)\mathbf{y}_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} \mathbf{z}_{p,j} - (\langle \mathbf{A}^\top \mathbf{y} - \mathbf{z}, \mathbf{x} \rangle - f(\mathbf{x})) \\
&\geq \sum_{t=1}^{T} (|B(t)| - k)\mathbf{y}_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} \mathbf{z}_{p,j} - \sup_{\mathbf{w} \in \mathbb{R}_+^T} (\langle \mathbf{A}^\top \mathbf{y} - \mathbf{z}, \mathbf{w} \rangle - f(\mathbf{w})) \\
&= \sum_{t=1}^{T} (|B(t)| - k)\mathbf{y}_t - \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} \mathbf{z}_{p,j} - f^*(\mathbf{A}^\top \mathbf{y} - \mathbf{z}).
\end{aligned}$$

All equalities are obtained via simple rearranging of terms, and the final inequality is due to the definition of the supremum. □

Lemma 3.16. *Let* $\bar{\mathbf{x}}$ *denote the value of* $\mathbf{x}$ *after processing the complete request sequence* σ*, and similarly for* $\bar{\mathbf{y}}$ *and* $\bar{\mathbf{z}}$*. If* $s_{p,j}(\tau)$ *is monotonically decreasing in* τ*, then*

$$\bar{\mathbf{x}}_{p,j} \geq \frac{1}{k} \left(\exp\left(\frac{s'_{p,j}}{r} \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j} \right) \right) - 1 \right), \tag{3.2}$$

where $s'_{p,j}$ *is the minimum value that* $s_{p,j}(\tau)$ *takes on during the execution of the algorithm.*

Proof. First, observe that $\mathbf{x}_{p,j}$ starts increasing at time $\tau(t(p,j))$, the time at which we finish processing the j'th request to p. $\mathbf{x}_{p,j}$ keeps increasing until one of the two events happens: $\mathbf{x}_{p,j}$ reaches 1, or we get the next request to p, after which it remains constant till the end. Let τ^{end} denote the time at which either of these events happens.

For $\tau \in [\tau(t(p,j)), \tau^{\text{end}})$, we have

$$\frac{d\mathbf{x}_{p,j}(\tau)}{d\tau} = s_{p,j}(\tau) \cdot \left(\mathbf{x}_{p,j}(\tau) + \frac{1}{k}\right) \geq s'_{p,j} \cdot \left(\mathbf{x}_{p,j}(\tau) + \frac{1}{k}\right),$$

and therefore,

$$\frac{1}{s'_{p,j}} \frac{d}{d\tau} \ln\left(\mathbf{x}_{p,j}(\tau) + \frac{1}{k}\right) = \frac{1}{s'_{p,j} \cdot (\mathbf{x}_{p,j}(\tau) + 1/k)} \cdot \frac{d\mathbf{x}_{p,j}(\tau)}{d\tau} \geq 1.$$

Integrating over the interval $[\tau(t(p,j)), \tau^{\text{end}})$, we get,

$$\frac{1}{s'_{p,j}} \ln\left(\frac{\bar{\mathbf{x}}_{p,j} + 1/k}{1/k}\right) \geq \tau^{\text{end}} - \tau(t(p,j)). \tag{3.12}$$

Next, observe that the variable y_t starts increasing from 0 at the uniform rate r at time $\tau(t-1)$ and stops increasing at time $\tau(t)$. Thus, $\bar{\mathbf{y}}_t = r \cdot (\tau(t) - \tau(t-1))$. Summing over all t from $t(p,j)+1$ to $t(p,j+1)-1$, we get,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t\right) = \tau(t(p,j+1)-1) - \tau(t(p,j)). \tag{3.13}$$

Finally, consider the variable $\mathbf{z}_{p,j}$. If $\mathbf{x}_{p,j}$ stops increasing because the next request to p arrives, then $\bar{\mathbf{z}}_{p,j} = 0$ and $\tau^{\text{end}} = \tau(t(p,j+1)-1)$. Using Eq. (3.13), we get,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right) = \tau(t(p,j+1)-1) - \tau(t(p,j)) = \tau^{\text{end}} - \tau(t(p,j)).$$

On the other hand, if $\mathbf{x}_{p,j}$ stops increasing because it reaches 1, then $\mathbf{z}_{p,j}$ starts increasing from 0 at the uniform rate r at time τ^{end}, and stops increasing at time $\tau(t(p,j+1)-1)$. Thus, $\bar{\mathbf{z}}_{p,j} = r \cdot (\tau(t(p,j+1)-1) - \tau^{\text{end}})$. Again, using Eq. (3.13), we get,

$$\begin{aligned}\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right) &= (\tau(t(p,j+1)-1) - \tau(t(p,j))) - (\tau(t(p,j+1)-1) - \tau^{\text{end}}) \\ &= \tau^{\text{end}} - \tau(t(p,j)).\end{aligned}$$

Thus, in either case, we have,

$$\frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right) = \tau^{\text{end}} - \tau(t(p,j)). \tag{3.14}$$

From Eqs. (3.12) and (3.14), we get,

$$\frac{1}{s'_{p,j}} \ln\left(\frac{\bar{\mathbf{x}}_{p,j} + 1/k}{1/k}\right) \geq \tau^{\text{end}} - \tau(t(p,j)) = \frac{1}{r} \cdot \left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right).$$

Rearranging, we get,

$$\bar{\mathbf{x}}_{p,j} \geq \frac{1}{k}\left(\exp\left(\frac{s'_{p,j}}{r}\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right)\right) - 1\right),$$

as required. □

Lemma 3.17. *The* $\mathbf{x}(\tau(t))$ *produced by Algorithm 4 throughout its execution are feasible for the primal for all* $t \in [T]$ *and the vector* $(\mathbf{A}^\top \bar{\mathbf{y}} - \bar{\mathbf{z}})_{p,j} \leq \frac{r}{s'_{p,j}} \ln(k+1)$.

Proof. The first statement follows immediately from the definition of the algorithm and the fact that we can always fulfill the primal constraints, for example, by setting all the variables to 1.

To show the second statement, we note that $\bar{\mathbf{x}} \leq \mathbf{1}$, which with Eq. (3.2) gives us

$$\frac{1}{k}\left(\exp\left(\frac{s'_{p,j}}{r}\left(\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j}\right)\right) - 1\right) \leq \bar{\mathbf{x}}_{p,j} \leq 1.$$

Taking the left- and right-hand sides gives us, after rearranging and taking logarithms,

$$\sum_{t=t(p,j)+1}^{t(p,j+1)-1} \bar{\mathbf{y}}_t - \bar{\mathbf{z}}_{p,j} \leq \frac{r}{s'_{p,j}} \ln(k+1),$$

where the left hand side is $(\mathbf{A}^\top \bar{\mathbf{y}} - \bar{\mathbf{z}})_{p,j}$ as a 1 only appears in the (p,j)th column of $\mathbf{A}$ from row $t(p,j)+1$ through row $t(p,j+1)-1$. □

Lemma 3.26. *Given a fractional solution* $\mathbf{x}$ *to the paging problem, we can find a fractional solution* $\mathbf{x}^*$ *in which every variable is a multiple of* $\delta = \frac{1}{4k}$*, and the cost of which is no more than* 3 *times the cost of* $\mathbf{x}$.

Proof. We define our rounded solution $\mathbf{x}^*$ as

$$\mathbf{x}^*_{p,j} = \begin{cases} 0 & \text{if } \mathbf{x}_{p,j} < \frac{1}{8k}, \\ \min\{\frac{1}{4k}\lceil 8k\mathbf{x}_{p,j}\rceil, 1\} & \text{otherwise.} \end{cases}$$

That is, we round every variable $\mathbf{x}_{p,j}$ of value less than $\frac{1}{8k}$ to 0, and every variable of greater value will be doubled and then rounded up to the nearest multiple of $\frac{1}{4k}$.

As each variable's value is at most doubled and then rounded up, the fractional cost for min-max paging can at most triple, as each variable in $\mathbf{x}^*$ is no larger than 3 times the corresponding variable in $\mathbf{x}$ and the objective function $f(\mathbf{x})$ in min-max paging satisfies $f(c\mathbf{x}) = cf(\mathbf{x})$ for any $c \in \mathbb{R}$. It remains to show that the solution $\mathbf{x}^*$ is feasible.

We note that the only variables whose value in $\mathbf{x}^*$ can decrease during the rounding are those of value less than $\frac{1}{8k}$. The total number of such variables other than p_t in a feasible solution is at most k, as otherwise.

$$\sum_{B(t)\setminus\{p_t\}} \mathbf{x}_{p,r(p,t)} \leq n - 1 - k + \frac{1}{8k}k \leq n - k.$$

Hence we note that the total contribution L of these variables to the constraint of round t is at most $\frac{1}{8}$.

As $n - k$ is an integer, and the fractional algorithm stops ejecting pages as soon as the constraint is satisfied, we know that $\sum_{p\in B(t)\setminus\{p_t\}} \mathbf{x}_{p,r(p,t)}$ is integral.

We further note that the contribution of all variables that are rounded up in this bound must therefore be at least $n - k - \frac{1}{8}$

If any variable $\mathbf{x}_{p,j}$ in $\mathbf{x}$ fulfills $\frac{1}{8} \leq \mathbf{x}_{p,j} \leq \frac{7}{8}$, then the rounding step will increase this variable by at least $\frac{1}{8}$, which compensates the value lost by rounding down all small variables.

Otherwise, if we let

$$A = \{\mathbf{x}_{p,r(p,t)} \in B(t) \setminus \{p_t\} \mid \mathbf{x}_{p,r(p,t)} \leq \frac{1}{8}\}$$

and $\sum_{z\in A} z \geq \frac{1}{8}$, then the doubling of these variables compensates for the rounding down of small variables.

Finally, if neither of these is the case, the variables of size greater than $\frac{7}{8}$ must sum up to at most $n - k - L$, where L denotes the amount of mass lost by rounding the small variables to 0, and at least $n - k - 2L > n - k - 1$. The sum of the rounded-up variables is an integer, as all variables of value greater than $\frac{1}{2}$ are rounded to 1, so we gain at least L from rounding up the large variables. □

3.9 Conclusion

In this chapter we introduced the *fair paging problem*, in which the objective is to minimize the maximum number of page faults incurred by any page.

We showed the first competitive deterministic and randomized algorithms, with competitive ratios of $\mathcal{O}(k \log n \log k)$ and $\mathcal{O}\left(\log^2(n)\log(k)\right)$, respectively. Additionally, we showed a deterministic lower bound of $\Omega(k \log_k(n))$ and a randomized lower bound of $\Omega(\log(n))$.

Our study of the min-max fair paging problem motivates the following open questions:

1. Can the lower bound be strengthened by a $\log(k)$ factor, which seems to be inherent to the way our lower bound is constructed?
2. Can the additional $\log(n)$ factor in the randomized upper bound be removed?
3. Can we further generalize our techniques to paging with arbitrary convex objective functions?

	$\dots$	$\mathbf{x}_{p_1,7}$	$\mathbf{x}_{p_2,9}$	$\mathbf{x}_{p_3,3}$	$\mathbf{x}_{p_1,8}$	$\mathbf{x}_{p_4,5}$	$\mathbf{x}_{p_2,10}$	$\mathbf{x}_{p_5,3}$	$\mathbf{x}_{p_1,9}$	$\dots$
$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$t(p_1,7)$	$\dots$	0	0	0	0	0	0	0	0	$\dots$
$t(p_2,9)$	$\dots$	1	0	0	0	0	0	0	0	$\dots$
$t(p_3,3)$	$\dots$	1	1	0	0	0	0	0	0	$\dots$
$t(p_1,8)$	$\dots$	0	1	1	0	0	0	0	0	$\dots$
$t(p_4,5)$	$\dots$	0	1	1	1	0	0	0	0	$\dots$
$t(p_2,10)$	$\dots$	0	0	1	1	1	0	0	0	$\dots$
$t(p_5,3)$	$\dots$	0	0	1	1	1	1	0	0	$\dots$
$t(p_1,9)$	$\dots$	0	0	1	0	1	1	1	0	$\dots$
$\vdots$	$\dots$	0	0	1	0	1	1	1	1	$\dots$
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Figure 3.1: An example of the structure of a constraint matrix of the paging problem, corresponding to the sub-sequence of requests $p_1, p_2, p_3, p_1, p_4, p_2, p_5, p_1$ starting at time $t' = t(p_1, 7)$. The matrix is lower triangular, due to the columns being in order of the appearance of variables. In this example, we assume that the pages p_1, p_2, p_3, p_4 and p_5 have been requested $6, 8, 2, 4$, and 2 times before the appearance of this sequence respectively.

4 Fast $(1+\varepsilon)$-Approximation Algorithms for Binary Matrix Factorization

We introduce efficient $(1+\varepsilon)$-approximation algorithms for the binary matrix factorization (BMF) problem, where the inputs are a matrix $\mathbf{A} \in \{0,1\}^{n\times d}$, a rank parameter $k > 0$, as well as an accuracy parameter $\varepsilon > 0$, and the goal is to approximate $\mathbf{A}$ as a product of low-rank factors $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$. Equivalently, we want to find $\mathbf{U}$ and $\mathbf{V}$ that minimize the Frobenius loss $\|\mathbf{UV} - \mathbf{A}\|_F^2$. Before this work, the state-of-the-art for this problem was the approximation algorithm of Kumar et al. [ICML 2019], which achieves a C-approximation for some constant $C \geq 576$. We give the first $(1+\varepsilon)$-approximation algorithm using running time singly exponential in k, where k is typically a small integer. Our techniques generalize to other common variants of the BMF problem, admitting bicriteria $(1+\varepsilon)$-approximation algorithms for L_p loss functions and the setting where matrix operations are performed in $\mathbb{F}_2$. Our approach can be implemented in standard big data models, such as the streaming or distributed models.

4.1 Introduction

Low-rank approximation is a fundamental tool for factor analysis. The goal is to decompose several observed variables stored in the matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ into a combination of k unobserved and uncorrelated variables called factors, represented by the matrices $\mathbf{U} \in \mathbb{R}^{n\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times d}$. In particular, we want to solve the problem

$$\min_{\mathbf{U}\in\mathbb{R}^{n\times k},\mathbf{V}\in\mathbb{R}^{k\times d}} \|\mathbf{UV} - \mathbf{A}\|,$$

for some predetermined norm $\|\cdot\|$. Identifying the factors can often decrease the number of relevant features in an observation and thus significantly improve interpretability. Another benefit is that low-rank matrices allow us to approximate the matrix $\mathbf{A}$ with its factors $\mathbf{U}$ and $\mathbf{V}$ using only $(n+d)k$ parameters rather than the nd parameters needed to represent $\mathbf{A}$. Moreover, for a vector $\mathbf{x} \in \mathbb{R}^d$, we can approximate the matrix-vector multiplication $\mathbf{Ax} \approx \mathbf{UVx}$ in time $(n+d)k$, while computing $\mathbf{Ax}$ requires nd time. These benefits make low-rank approximation one of the most widely used tools in machine learning, recommender systems, data science, statistics, computer vision, and natural language processing. In many of these applications, discrete or categorical datasets are typical. In this case, restricting the underlying factors to a discrete domain for interpretability often makes sense. For example, Kumar et al. [202] observed that nearly half of the datasets in the UCI repository [116] are categorical and thus can be represented as binary matrices, possibly using multiple binary variables to represent each category.

In the binary matrix factorization (BMF) problem, the input matrix $\mathbf{A} \in \{0,1\}^{n\times d}$ is binary. Additionally, we are given an integer range parameter k, with $0 < k \ll n, d$. The goal is to approximate $\mathbf{A}$ by the factors $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$ such that $\mathbf{A} \approx \mathbf{UV}$. The BMF problem restricts the general low-rank approximation problem to a discrete space, making finding good factors more challenging (see Section 4.1.3).

4.1.1 Our Contributions

We present $(1+\varepsilon)$-approximation algorithms for the binary low-rank matrix factorization problem for several standard loss functions used in the general low-rank approximation problem. Table 4.1 summarizes our results.

Reference	Approximation	Runtime	Other
[202]	$C \geq 576$	$2^{\tilde{\mathcal{O}}(k^2)}\operatorname{poly}(n,d)$	Frobenius loss
[130]	$1+\varepsilon$	$2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log^2\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$	Frobenius loss
Our work	$1+\varepsilon$	$2^{\tilde{\mathcal{O}}(k^2/\varepsilon^4)}\operatorname{poly}(n,d)$	Frobenius loss
[202]	$C \geq 122^{2p-2}+2^{p-1}$	$2^{\operatorname{poly}(k)}\operatorname{poly}(n,d)$	L_p loss, $p \geq 1$
Here	$1+\varepsilon$	$2^{\operatorname{poly}(k/\varepsilon)}\operatorname{poly}(n,d)$	L_p loss, $p \geq 1$, bicriteria
[130]	$1+\varepsilon$	$2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log^2\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$	Binary field
[32]	$1+\varepsilon$	$2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$	Binary field
[202]	$C \geq 40001$	$2^{\operatorname{poly}(k)}\operatorname{poly}(n,d)$	Binary field, bicriteria
Our work	$1+\varepsilon$	$2^{\operatorname{poly}(k/\varepsilon)}\operatorname{poly}(n,d)$	Binary field, bicriteria

Table 4.1: Summary of related work on binary matrix factorization

Binary matrix factorization. We first consider the minimization of the Frobenius norm, defined by $\|\mathbf{A}-\mathbf{UV}\|_F^2 = \sum_{i\in[n]}\sum_{j\in d}|\mathbf{A}_{i,j}-(\mathbf{UV})_{i,j}|^2$, where $[n] := \{1,\ldots,n\}$ and $\mathbf{A}_{i,j}$ denotes the entry in the ith row and the jth column of $\mathbf{A}$. Intuitively, we can view this as finding a least-squares approximation of $\mathbf{A}$.

We introduce the first $(1+\varepsilon)$-approximation algorithm for the BMF problem that runs in singly exponential time. That is, we present an algorithm that, for any $\varepsilon > 0$, returns $\mathbf{U}' \in \{0,1\}^{n\times k}, \mathbf{V}' \in \{0,1\}^{k\times d}$ with

$$\|\mathbf{A}-\mathbf{U}'\mathbf{V}'\|_F^2 \leq (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{A}-\mathbf{UV}\|_F^2.$$

For $\varepsilon \in (0,1)$, our algorithm uses $2^{\tilde{\mathcal{O}}(k^2/\varepsilon^4)}\operatorname{poly}(n,d)$ runtime and for $\varepsilon \geq 1$, our algorithm uses $2^{\tilde{\mathcal{O}}(k^2)}\operatorname{poly}(n,d)$ runtime, where $\operatorname{poly}(n,d)$ denotes a polynomial in n and d.

By comparison, Kumar et al. [202] gave a C-approximation algorithm for the BMF problem also using runtime $2^{\tilde{\mathcal{O}}(k^2)}\operatorname{poly}(n,d)$, but for some constant $C \geq 576$. Though they did not attempt to optimize for C, their proofs employ multiple triangle inequalities

that present a constant lower bound of at least 2 on C. See Section 4.1.2 for a more thorough discussion of the limitations of their approach. Fomin et al. [130] introduced a $(1+\varepsilon)$-approximation algorithm for the BMF problem with rank-k factors. However, their algorithm uses time doubly exponential in k, specifically $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log^2\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$, which Ban et al. [32] later improved to doubly exponential runtime $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$, while also showing that time $2^{k^{\Omega(1)}}$ is necessary even for constant-factor approximation, under the Small Set Expansion Hypothesis and the Exponential Time Hypothesis.

BMF with L_p loss. We also consider the more general problem of minimizing for L_p loss for a given p, defined as the optimization problem of minimizing $\|\mathbf{A}-\mathbf{UV}\|_p^p = \sum_{i\in[n]}\sum_{j\in d}|\mathbf{A}_{i,j}-(\mathbf{UV})_{i,j}|^p$. Of particular interest is the case $p=1$, which corresponds to robust principal component analysis, and which has been proposed as an alternative to Frobenius norm low-rank approximation that is more robust to outliers, i.e., values that are far away from the majority of the data points [188, 189, 204, 338, 56, 222, 298, 253, 32, 217]. On the other hand, for $p>2$, low-rank approximation with L_p error increasingly places higher priority on outliers, i.e., the larger entries of $\mathbf{UV}$.

We present the first $(1+\varepsilon)$-approximation algorithm for the BMF problem that runs in singly exponential time, albeit at the cost of incurring logarithmic increases in the rank k, making it a bicriteria algorithm. Specifically, for any $\varepsilon>0$, our algorithm returns $\mathbf{U}'\in\{0,1\}^{n\times k'}, \mathbf{V}'\in\{0,1\}^{k'\times d}$ with

$$\|\mathbf{A}-\mathbf{U}'\mathbf{V}'\|_p^p \le (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{A}-\mathbf{UV}\|_p^p,$$

where $k'=\mathcal{O}\left(\frac{k\log^2 n}{\varepsilon^2}\right)$. For $\varepsilon\in(0,1)$, our algorithm uses $2^{\operatorname{poly}(k/\varepsilon)}\operatorname{poly}(n,d)$ runtime and for $\varepsilon\ge 1$, our algorithm uses $2^{\operatorname{poly}(k)}\operatorname{poly}(n,d)$ runtime.

Previous work by Kumar et al. [202] gave a C-approxmiation algorithm for this problem, using singly exponential runtime $2^{\operatorname{poly}(k)}\operatorname{poly}(n,d)$, without incurring a bicriteria loss in the rank k. However, their constant $C\ge 122^{2p-2}+2^{p-1}$ is large and depends on p. Again, their use of multiple triangle inequalities in their argument bars this approach from being able to achieve a $(1+\varepsilon)$-approximation. To our knowledge, no prior works achieved $(1+\varepsilon)$-approximation to BMF with L_p loss in singly exponential time.

BMF on binary fields. Finally, we consider the case where all arithmetic operations are performed modulo two, i.e., in the finite field $\mathbb{F}_2$. Specifically, the (i,j)th entry of $\mathbf{UV}$ is the inner product $\langle\mathbf{U}_i,\mathbf{V}^{(j)}\rangle$ of the ith row of $\mathbf{U}$ and the jth column of $\mathbf{V}$, taken over $\mathbb{F}_2$. This model has been frequently used for dimensionality reduction for high-dimensional data with binary attributes [201, 288, 171, 96] and independent component analysis, especially in the context of signal processing [325, 144, 251, 252]. This problem is also known as bipartite clique cover, the discrete basis problem, or minimal noise role mining and has been well-studied in applications to association rule mining, database tiling, and topic modeling [285, 293, 311, 231, 40, 213, 66, 73].

We introduce the first bicriteria $(1+\varepsilon)$-approximation algorithm for the BMF problem on binary fields that runs in singly exponential time. Specifically, for any $\varepsilon > 0$, our algorithm returns $\mathbf{U}' \in \{0,1\}^{n\times k'}, \mathbf{V}' \in \{0,1\}^{k'\times d}$ with

$$\|\mathbf{A} - \mathbf{U}'\mathbf{V}'\|_p^p \leq (1+\varepsilon) \min_{\mathbf{U}\in\{0,1\}^{n\times k}, \mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{A} - \mathbf{U}\mathbf{V}\|_p^p,$$

where $k' = \mathcal{O}\left(\frac{k\log n}{\varepsilon}\right)$ and all arithmetic operations are performed in $\mathbb{F}_2$. For $\varepsilon \in (0,1)$, our algorithm has running time $2^{\mathrm{poly}(k/\varepsilon)}\,\mathrm{poly}(n,d)$ and for $\varepsilon \geq 1$, our algorithm has running time $2^{\mathrm{poly}(k)}\,\mathrm{poly}(n,d)$.

By comparison, Kumar et al. [202] gave a bicriteria C-approximation algorithm for the BMF problem on binary fields with running time $2^{\mathrm{poly}(k)}\,\mathrm{poly}(n,d)$, for some constant $C \geq 40001$. Even though their algorithm also gives a bicriteria guarantee, their approach, once again, inherently cannot achieve $(1+\varepsilon)$-approximation. On the other hand, Fomin et al. [130] achieved a $(1+\varepsilon)$-approximation without a bicriteria guarantee, but their algorithm uses doubly exponential running time $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log^2\frac{1}{\varepsilon}}\,\mathrm{poly}(n,d)$, which Ban et al. [32] later improved to doubly exponential running time $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log\frac{1}{\varepsilon}}\,\mathrm{poly}(n,d)$, while also showing that running time doubly exponential in k is necessary for $(1+\varepsilon)$-approximation on $\mathbb{F}_2$.

Applications to big data models. We remark that our algorithms are conducive to big data models. Specifically, our algorithmic ideas facilitate a two-pass algorithm in the streaming model, where either the rows or the columns of the input matrix arrive sequentially, and the goal is to perform binary low-rank approximation while using space sublinear in the size of the input matrix. Similarly, our approach can be used to achieve a two-round protocol in the distributed model, where either the rows or the columns of the input matrix are partitioned among several players, and the goal is to perform binary low-rank approximation while using total communication sublinear in the size of the input matrix. See Section 4.5 for a formal description of the problem settings and additional details.

4.1.2 Overview of Our Techniques

This section briefly overviews our approaches to achieving $(1+\varepsilon)$-approximation to the BMF problem. Alongside our techniques, we discuss why prior approaches for BMF fail to achieve $(1+\varepsilon)$-approximation.

The BMF problem under the Frobenius norm is stated as follows: Let $\mathbf{U}^* \in \{0,1\}^{n\times k}$ and $\mathbf{V}^* \in \{0,1\}^{k\times d}$ be optimal low-rank factors, so that

$$\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_F^2 = \min_{\mathbf{U}\in\{0,1\}^{n\times k}, \mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2. \tag{4.1}$$

Our approach relies on the sketch-and-solve paradigm, and we ask of our sketch matrix $\mathbf{S}$ that it is an *affine embedding*, that is, given $\mathbf{U}^*$ and $\mathbf{A}$, for all $\mathbf{V} \in \{0,1\}^{k\times d}$,

$$(1-\varepsilon)\|\mathbf{U}^*\mathbf{V} - \mathbf{A}\|_F^2 \leq \|\mathbf{S}\mathbf{U}^*\mathbf{V} - \mathbf{S}\mathbf{A}\|_F^2 \leq (1+\varepsilon)\|\mathbf{U}^*\mathbf{V} - \mathbf{A}\|_F^2.$$

Observe that if $\mathbf{S}$ is an affine embedding, then we obtain a $(1+\varepsilon)$-approximation by solving for the minimizer $\mathbf{V}^*$ in the sketched space. That is, given $\mathbf{S}$ and $\mathbf{U}^*$, instead of solving Eq. (4.1) for $\mathbf{V}^*$, it suffices to solve

$$\operatorname*{argmin}_{\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{S}\mathbf{U}^*\mathbf{V} - \mathbf{S}\mathbf{A}\|_F^2.$$

Guessing the sketch matrix S. A general approach taken by Razenshteyn et al. [269], Kumar et al. [202] and Ban et al. [33] for various low-rank approximation problems is first to choose $\mathbf{S}$ in a way so that there are not too many possibilities for the matrices $\mathbf{S}\mathbf{U}^*$ and $\mathbf{S}\mathbf{A}$ and then find the minimizer $\mathbf{V}^*$ for all guesses of $\mathbf{S}\mathbf{U}^*$ and $\mathbf{S}\mathbf{A}$. Note that this approach is delicate because it depends on the choice of the sketch matrix $\mathbf{S}$. For example, if we chose $\mathbf{S}$ to be a dense matrix with random Gaussian entries, then since there are 2^{nk} possibilities for the matrix $\mathbf{U}^* \in \{0,1\}^{n\times k}$, we cannot enumerate the possible matrices $\mathbf{S}\mathbf{U}^*$. Prior work [269, 202, 33] made the key observation that if $\mathbf{A}$ (and thus $\mathbf{U}^*$) has a small number of unique rows, then a matrix $\mathbf{S}$ that samples a small number of rows of $\mathbf{A}$ has only a small number of possibilities for $\mathbf{S}\mathbf{A}$.

To ensure that $\mathbf{A}$ has a small number of unique rows for the BMF problem, Kumar et al. [202] first find a 2^k-means clustering solution $\widetilde{\mathbf{A}}$ for the rows of $\mathbf{A}$. Instead of solving the problem on $\mathbf{A}$, they then solve BMF on the matrix $\widetilde{\mathbf{A}}$, where each row is replaced by the center the point is assigned to, yielding at most 2^k unique rows. Finally, they note that $\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_F^2$ is at least the 2^k-means cost, as $\mathbf{U}^*\mathbf{V}^*$ has at most 2^k unique rows. Now that $\widetilde{\mathbf{A}}$ has 2^k unique rows, they can make all possible guesses for both $\mathbf{S}\mathbf{U}^*$ and $\mathbf{S}\widetilde{\mathbf{A}}$ in time $2^{\tilde{\mathcal{O}}(k^2)}$. By using an algorithm of Kanungo et al. [182] that achieves roughly a 9-approximation to k-means clustering, [202] ultimately obtain a C-approximation to the BMF problem, for some $C \geq 576$.

Shortcomings of previous work for $(1+\varepsilon)$-approximation. While Kumar et al. [202] do not optimize for C, their approach fundamentally cannot achieve $(1+\varepsilon)$-approximation for BMF for the following reasons. First, they use a k-means clustering subroutine [182], (achieving roughly a 9-approximation) which due to hardness-of-approximation results [87, 206] can never achieve $(1+\varepsilon)$-approximation, as there cannot exist a 1.07-approximation algorithm for k-means clustering unless P=NP. Moreover, even if a $(1+\varepsilon)$-approximate k-means clustering could be found, there is no guarantee that the cluster centers obtained by this algorithm are binary. That is, while $\mathbf{U}\mathbf{V}$ has a specific form induced by the requirement that each factor must be binary, a solution to k-means clustering offers no such guarantee and may return Steiner points. Finally, Kumar et al. [202] utilize a sketching matrix $\mathbf{S}$ that roughly preserves $\mathbf{S}\mathbf{U}^*$ and $\mathbf{S}\mathbf{A}$. By generalizations of the triangle inequality, one can show that $\|\mathbf{S}\mathbf{U}^*\mathbf{V}^* - \mathbf{S}\mathbf{A}\|_F^2$ preserves a constant factor approximation to $\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_F^2$, but not necessarily a $(1+\varepsilon)$-approximation.

Another related work by Fomin et al. [130] reduces instances of BMF to constrained k-means clustering instances, where the constraints demand that the selected centers are linear combinations of binary vectors. The core part of their work is to design a sampling-based algorithm for solving binary-constrained clustering instances, and the

result on BMF is a corollary. Constrained clustering is a harder problem than BMF with Frobenius loss, so it is unclear how one might improve the doubly exponential running time using this approach.

Our approach: computing a strong coreset. We first reduce the number of unique rows in $\mathbf{A}$ by computing a strong coreset $\widetilde{\mathbf{A}}$ for $\mathbf{A}$. The strong coreset has the property that for any choices of $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$, there exists $\mathbf{X} \in \{0,1\}^{n\times k}$ such that

$$(1-\varepsilon)\|\mathbf{UV}-\mathbf{A}\|_F^2 \le \|\mathbf{XV}-\widetilde{\mathbf{A}}\|_F^2 \le (1+\varepsilon)\|\mathbf{UV}-\mathbf{A}\|_F^2.$$

Therefore, we instead first solve the low-rank approximation problem on $\widetilde{\mathbf{A}}$ first. Crucially, we choose $\widetilde{\mathbf{A}}$ to have $2^{\mathrm{poly}(k/\varepsilon)}$ unique rows so then for a matrix $\mathbf{S}$ that samples $\mathrm{poly}(k/\varepsilon)$ rows, there are $2^{\mathrm{poly}(k/\varepsilon)}$ possibilities for $\mathbf{S}\widetilde{\mathbf{A}}$, so we can make all possible guesses for both $\mathbf{SU}^*$ and $\mathbf{S}\widetilde{\mathbf{A}}$. Unfortunately, we still have the problem that $\|\mathbf{SU}^*\mathbf{V}^*-\mathbf{S}\widetilde{\mathbf{A}}\|_F^2$ does not even necessarily give a $(1+\varepsilon)$-approximation to $\|\mathbf{U}^*\mathbf{V}^*-\widetilde{\mathbf{A}}\|_F^2$.

Binary matrix factorization. To that end, we show that when $\mathbf{S}$ is a leverage score sampling matrix, then $\mathbf{S}$ also satisfies an approximate matrix multiplication property. Therefore $\mathbf{S}$ can effectively be used for an affine embedding. That is, the minimizer to $\|\mathbf{SU}^*\mathbf{V}^*-\mathbf{S}\widetilde{\mathbf{A}}\|_F^2$ produces an $(1+\varepsilon)$-approximation to the cost of the optimal factors $\|\mathbf{U}^*\mathbf{V}^*-\widetilde{\mathbf{A}}\|_F^2$. Thus, we can then solve

$$\begin{aligned}\mathbf{V}' &= \operatorname*{argmin}_{\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{SU}^*\mathbf{V}-\mathbf{S}\widetilde{\mathbf{A}}\|_F^2\\ \mathbf{U}' &= \operatorname*{argmin}_{\mathbf{U}\in\{0,1\}^{n\times k}} \|\mathbf{UV}'-\mathbf{A}\|_F^2,\end{aligned}$$

where the latter optimization problem can be solved by iteratively optimizing over each row so that the total computation time is $\mathcal{O}\left(2^k n\right)$ rather than 2^{kn}.

BMF on binary fields. We again form the matrix $\widetilde{\mathbf{A}}$ by taking a strong coreset of $\mathbf{A}$, constructed using an algorithm that gives integer weight w_i to each point, and then duplicating the rows to form $\widetilde{\mathbf{A}}$. That is, if the i-th row $\mathbf{A}_i$ of $\mathbf{A}$ is sampled with weight w_i in the coreset, then $\widetilde{\mathbf{A}}$ will contain w_i repetitions of the row $\mathbf{A}_i$. We want to use the same approach for binary fields to make guesses for $\mathbf{SU}^*$ and $\mathbf{S}\widetilde{\mathbf{A}}$. However, it is no longer true that $\mathbf{S}$ will provide an affine embedding over $\mathbb{F}_2$, in part because the subspace embedding property of $\mathbf{S}$ computes leverage scores of each row of $\mathbf{U}^*$ and $\mathbf{A}$ with respect to general integers. Hence, we require a different approach for matrix operations over $\mathbb{F}_2$.

Instead, we group the rows of $\widetilde{\mathbf{A}}$ by their number of repetitions, so that group $\mathbf{G}_j$ consists of the rows of $\widetilde{\mathbf{A}}$ that are repeated $[(1+\varepsilon)^j,(1+\varepsilon)^{j+1})$ times. That is, if $\mathbf{A}_i$ appears w_i times in $\widetilde{\mathbf{A}}$, then it appears a single time in group $\mathbf{G}_j$ for $j=\lfloor \log w_i \rfloor$. We then perform entrywise L_0 low-rank approximation over $\mathbb{F}_2$ for each of the groups $\mathbf{G}_j$, which gives low-rank factors $\mathbf{U}^{(j)}$ and $\mathbf{V}^{(j)}$. We then compute $\widetilde{\mathbf{U}^{(j)}}$ by duplicating rows appropriately so that if $\mathbf{A}_i$ is in $\mathbf{G}_j$, then we place the row of $\mathbf{U}^{(j)}$ corresponding to $\mathbf{A}_i$

into the ith row of $\widetilde{\mathbf{U}^{(j)}}$, for all $i \in [n]$. Otherwise if $\mathbf{A}_i$ is not in $\mathbf{G}_j$, then we set ith row of $\widetilde{\mathbf{U}^{(j)}}$ to be the all zeros row. We compute $\mathbf{V}^{(j)}$ by padding accordingly and then collect

$$\widetilde{\mathbf{U}} = \left[\widetilde{\mathbf{U}^{(0)}} | \ldots | \widetilde{\mathbf{U}^{(\ell)}}\right], \qquad \widetilde{\mathbf{V}} \leftarrow \widetilde{\mathbf{V}^{(0)}} \circ \ldots \circ \widetilde{\mathbf{V}^{(i)}},$$

where $\left[\widetilde{\mathbf{U}^{(0)}} | \ldots | \widetilde{\mathbf{U}^{(\ell)}}\right]$ denotes horizontal concatenation of matrices and $\widetilde{\mathbf{V}^{(0)}} \circ \ldots \circ \widetilde{\mathbf{V}^{(i)}}$ denotes vertical concatenation (stacking) of matrices, to achieve bicriteria low-rank approximations $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ to $\widetilde{\mathbf{A}}$. Finally, to achieve bicriteria factors $\mathbf{U}'$ and $\mathbf{V}'$ to $\mathbf{A}$, we ensure that $\mathbf{U}'$ achieves the same block structure as $\widetilde{\mathbf{U}}$.

BMF with L_p loss. We would again like to use the same approach as our $(1+\varepsilon)$-approximation algorithm for BMF with Frobenius loss. To that end, we observe that a coreset construction for clustering under L_p metrics rather than Euclidean distance is known, which we can use to construct $\widetilde{\mathbf{A}}$. However, the challenge is that no known sampling matrix $\mathbf{S}$ guarantees an affine embedding. One might hope that recent results on active L_p regression [76, 254, 236, 228, 229] can provide such a tool. Unfortunately, adapting these techniques would still require taking a union bound over a number of columns, which would result in the sampling matrix having too many rows for our desired runtime.

Instead, we invoke the coreset construction on the rows and the columns so that $\widetilde{\mathbf{A}}$ has a small number of distinct rows and columns. We again partition the rows of $\widetilde{\mathbf{A}}$ into groups based on their frequency, but now we further partition the groups based on the frequency of the columns. Thus, it remains to solve BMF with L_p loss on the partition, each part of which has a small number of rows and columns. Since the contribution of each row toward the overall loss is small (because there is a small number of columns), we show that there exists a matrix that samples $\mathrm{poly}(k/\varepsilon)$ rows of each partition that finally achieves the desired affine embedding. Therefore, we can solve the problem on each partition, pad the factors accordingly, and build the bicriteria factors as in the binary field case.

4.1.3 Motivation and Related Work

Low-rank approximation is one of the fundamental problems of machine learning and data science. Therefore, it has received extensive attention, e.g., see the surveys [179, 218, 321]. When the underlying loss function is the Frobenius norm, the low-rank approximation problem can be optimally solved via singular value decomposition (SVD). However, when we restrict both the observed input $\mathbf{A}$ and the factors $\mathbf{U}, \mathbf{V}$ to binary matrices, the SVD no longer guarantees optimal factors. In fact, many restricted variants of low-rank approximation are NP-hard [269, 298, 202, 32, 33, 130, 217].

Motivation and background for BMF. The BMF problem has applications to graph partitioning [66], low-density parity-check codes [268], and optimizing passive organic LED (OLED) displays [202]. Observe that we can use $\mathbf{A}$ to encode the incidence matrix

of the bipartite graph with n vertices on the left side of the bipartition and d vertices on the right side so that $\mathbf{A}_{i,j} = 1$ if and only if there exists an edge connecting the ith vertex on the left side with the jth vertex on the right side. Then $\mathbf{UV}$ can be written as the sum of k rank-1 matrices, each encoding a different bipartite clique of the graph, i.e., a subset of vertices on the left and a subset of vertices on the right such that there exists an edge between every vertex on the left and every vertex on the right. It then follows that the BMF problem solves the bipartite clique partition problem [249, 128, 65, 242], in which the goal is to find the smallest integer k such that the graph can be represented as a union of k bipartite cliques.

Kumar et al. [202] also present the following motivation for the BMF problem to improve the performance of passive OLED displays, which rapidly and sequentially illuminate rows of lights to render an image in a manner so that the human eye integrates this sequence of lights into a complete image. However, Kumar et al. [202] observed that passive OLED displays could illuminate many rows simultaneously, provided the image being shown is a rank-1 matrix and that the apparent brightness of an image is inversely proportional to the rank of the decomposition. Thus Kumar et al. [202] note that BMF can be used to not only find a low-rank decomposition that illuminates pixels in a way that seems brighter to the viewer but also achieves binary restrictions on the decomposition in order to use simple and inexpensive voltage drivers on the rows and columns, rather than a more expensive bank of video-rate digital to analog-to-digital converters.

BMF with Frobenius loss. Kumar et al. [202] were the first to give a constant factor approximation algorithm for the BMF problem using runtime $2^{\tilde{\mathcal{O}}(k^2)}\operatorname{poly}(n,d)$, i.e., time singly exponential in k. Fomin et al. [130] introduced a $(1+\varepsilon)$-approximation to the BMF problem with rank-k factors, but their algorithm uses doubly exponential time, specifically runtime $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log^2\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$, which was later improved to doubly exponential runtime $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2}\log\frac{1}{\varepsilon}}\operatorname{poly}(n,d)$ by [32], who also showed that $2^{k^{\Omega(1)}}$ runtime is necessary even for constant-factor approximation, under the Small Set Expansion Hypothesis and the Exponential Time Hypothesis. By introducing sparsity constraints on the rows of $\mathbf{U}$ and $\mathbf{V}$, Chen et al. [73] provide an alternate parametrization of the runtime, though, at the cost of runtime quasipolynomial in n and d.

BMF on binary fields. Binary matrix factorization is particularly suited for datasets involving binary data. Thus, the problem is well-motivated for binary fields when performing dimensionality reduction on high-dimension datasets [201]. To this end, many heuristics have been developed for this problem [201, 288, 132, 171], due to its NP-hardness [137, 96].

For the special case of $k = 1$, Shen et al. [288] first gave a 2-approximation algorithm that uses polynomial time through a relaxation of integer linear programming. Subsequently, Jiang et al. [171] produced a simpler approach, and Bringmann et al. [55] introduced a sublinear time algorithm. For general k, Kumar et al. [202] gave a constant factor approximation algorithm using runtime $2^{\operatorname{poly}(k)}\operatorname{poly}(n,d)$, i.e., singly exponential time,

at the expense of a bicriteria solution, i.e., factors with rank $k' = \mathcal{O}(k \log n)$. Fomin et al. [130] introduced a $(1+\varepsilon)$-approximation to the BMF problem with rank-k factors, but their algorithm uses doubly exponential time, specifically runtime $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2} \log^2 \frac{1}{\varepsilon}} \operatorname{poly}(n, d)$, which was later improved to doubly exponential runtime $2^{\frac{2^{\mathcal{O}(k)}}{\varepsilon^2} \log \frac{1}{\varepsilon}} \operatorname{poly}(n, d)$ by Ban et al. [32], who also showed that doubly exponential runtime is necessary for $(1+\varepsilon)$-approximation without bicriteria relaxation under the Exponential Time Hypothesis.

BMF with L_p loss. Using more general L_p loss functions can result in drastically different behaviors of the optimal low-rank factors for the BMF problem. For example, the low-rank factors for $p > 2$ are penalized more when the corresponding entries of $\mathbf{UV}$ are large, and thus may choose to prioritize a larger number of small entries that do not match $\mathbf{A}$ rather than a single large entry. On the other hand, $p = 1$ corresponds to robust principal component analysis, which yields factors that are more robust to outliers in the data [188, 189, 204, 338, 56, 222, 298, 253, 32, 217]. The first approximation algorithm with provable guarantees for L_1 low-rank approximation on the reals was given by Song et al. [298]. They achieved $\operatorname{poly}(k) \cdot \log d$-approximation in roughly $\mathcal{O}(nd)$ time. For constant k, Song et al. [298] further achieved constant-factor approximation in polynomial time.

When we restrict the inputs and factors to be binary, Kumar et al. [202] observed that $p = 1$ corresponds to minimizing the number of edges in the symmetric difference between an unweighted bipartite graph G and its approximation H, which is the multiset union of k bicliques. Here we represent the graph G with n and d vertices on the bipartition's left- and right-hand side, respectively, through its edge incidence matrix $\mathbf{A}$. Similarly, we have $\mathbf{U}_{i,j} = 1$ if and only if the i-th vertex on the left bipartition is in the j-th biclique and $\mathbf{V}_{i,j} = 1$ if and only if the j-th vertex on the right bipartition is in the i-th biclique. Then we have $\|\mathbf{UV} - \mathbf{A}\|_1 = |E(G) \triangle E(H)|$. Chandran et al. [66] showed how to solve the exact version of the problem, i.e., to recover $\mathbf{U}, \mathbf{V}$ under the promise that $\mathbf{A} = \mathbf{UV}$, using $2^{\mathcal{O}(k^2)} \operatorname{poly}(n, d)$ time. Kumar et al. [202] recently gave the first constant-factor approximation algorithm for this problem, achieving a C-approximation using $2^{\operatorname{poly}(k)} \operatorname{poly}(n, d)$ time, for some constant $C \geq 122^{2p-2} + 2^{p-1}$.

4.1.4 Preliminaries

For an integer $n > 0$, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use $\operatorname{poly}(n)$ to represent a fixed polynomial in n and more generally, $\operatorname{poly}(n_1, \ldots, n_k)$ to represent a fixed multivariate polynomial in $n_1, \ldots, n_k$. For a function $f(n_1, \ldots, n_k)$, we use $\tilde{\mathcal{O}}(f(n_1, \ldots, n_k))$ to denote $f(n_1, \ldots, n_k) \cdot \operatorname{poly}(\log f(n_1, \ldots, n_k))$.

We generally use bold-font variables to denote matrices. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, we use $\mathbf{A}_i$ to denote the ith row of $\mathbf{A}$ and $\mathbf{A}^{(j)}$ to denote the jth column of $\mathbf{A}$. We use $A_{i,j}$ to denote the entry in the ith row and jth column of $\mathbf{A}$. For $p \geq 1$, we write the

entry-wise L_p norm of $\mathbf{A}$ as

$$\|\mathbf{A}\|_p = \left(\sum_{i \in [n]} \sum_{j \in [d]} A_{i,j}^p \right)^{1/p}.$$

The Frobenius norm of $\mathbf{A}$, denoted $\|\mathbf{A}\|_F$ is simply the entry-wise L_2 norm of $\mathbf{A}$:

$$\|\mathbf{A}\|_F = \left(\sum_{i \in [n]} \sum_{j \in [d]} A_{i,j}^2 \right)^{1/2}.$$

The entry-wise L_0 norm of $\mathbf{A}$ is

$$\|\mathbf{A}\|_0 = |\{(i,j) \mid i \in [n], j \in [d] : A_{i,j} \neq 0\}|.$$

We use $\circ$ to denote vertical stacking of matrices, so that

$$\mathbf{A}^{(1)} \circ \ldots \circ \mathbf{A}^{(m)} = \begin{bmatrix} \mathbf{A}^{(1)} \\ \vdots \\ \mathbf{A}^{(m)} \end{bmatrix}.$$

For a set X of n points in $\mathbb{R}^d$ weighted by a function w, the k-means clustering cost of X with respect to a set S of k centers is defined as

$$\operatorname{cost}(X, S, w) := \sum_{x \in X} w(x) \cdot \min_{s \in S} \|x - s\|_2^2.$$

When the weights w are uniformly unit across all points in X, we simply write $\operatorname{cost}(X, S) = \operatorname{cost}(X, S, w)$.

One of the core ingredients for avoiding the triangle inequality and achieving $(1+\varepsilon)$-approximation is our use of coresets for k-means clustering:

Definition 4.1 (Strong coreset)**.** *Given an accuracy parameter $\varepsilon > 0$ and a set X of n points in $\mathbb{R}^d$, we say that a subset C of X with weights w is a strong ε-coreset of X for the k-means clustering problem if for any set S of k points in $\mathbb{R}^d$, we have*

$$(1-\varepsilon)\operatorname{cost}(X, S) \leq \operatorname{cost}(C, S, w) \leq (1+\varepsilon)\operatorname{cost}(X, S).$$

Many coreset construction exist in the literature, and the goal is to minimize $|C|$, the size of the coreset, while preserving $(1 \pm \varepsilon)$-approximate cost for all sets of k centers. If the points lie in $\mathbb{R}^d$, we can find coresets of size $\tilde{\mathcal{O}}\left(\operatorname{poly}(k, d, \varepsilon^{-1})\right)$, i.e., the size is independent of n.

Leverage scores. Finally, we recall the notion of a leverage score sampling matrix. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, the leverage score of row $\mathbf{a}_i$ with $i \in [n]$ is defined as $\mathbf{a}_i(\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{a}_i^\top$. We can use the leverage scores to generate a random leverage score sampling matrix as follows:

Theorem 4.2 (Leverage score sampling matrix)**.** *[113, 114, 215, 321] Let $C > 1$ be a universal constant and $\alpha > 1$ be a parameter. Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, let ℓ_i be the leverage score of the i-th row of $\mathbf{A}$. Suppose, for all $i \in [n]$,*

$$p_i \in \left[\min\left(1, \frac{C\ell_i \log k}{\varepsilon^2}\right), \min\left(1, \frac{C\alpha\ell_i \log k}{\varepsilon^2}\right)\right].$$

For $m := \mathcal{O}\left(\frac{\alpha}{\varepsilon^2} d \log d\right)$, let $\mathbf{S} \in \mathbb{R}^{m \times n}$ be generated so that each row of $\mathbf{S}$ randomly selects row $j \in [n]$ with probability proportional to p_j and rescales the row by $\frac{1}{\sqrt{mp_j}}$. Then with probability at least 0.99, we have that simultaneously for all vectors $\mathbf{x} \in \mathbb{R}^d$,

$$(1-\varepsilon)\|\mathbf{A}\mathbf{x}\|_2 \leq \|\mathbf{S}\mathbf{A}\mathbf{x}\|_2 \leq (1+\varepsilon)\|\mathbf{A}\mathbf{x}\|_2.$$

The main point of Theorem 4.2 is that given constant-factor approximations p_i to the leverage scores ℓ_i, it suffices to sample $\mathcal{O}(d \log d)$ rows of $\mathbf{A}$ to achieve a constant-factor subspace embedding of $\mathbf{A}$, and similar bounds can be achieved for $(1+\varepsilon)$-approximate subspace embeddings. Finally, we remark that $\mathbf{S}$ can be decomposed as the product of matrices $\mathbf{DT}$, where $\mathbf{T} \in \mathbb{R}^{m \times n}$ is a sparse matrix with a single one per row, denoting the selection of a row for the purposes of leverage score sampling and $\mathbf{D}$ is the diagonal matrix with the corresponding scaling factor, i.e., the ith diagonal entry of $\mathbf{D}$ is set to $\frac{1}{\sqrt{mp_j}}$ if the jth row of $\mathbf{A}$ is selected for the i-th sample.

4.2 Binary Low-Rank Approximation

In this section, we present a $(1+\varepsilon)$-approximation algorithm for binary low-rank approximation with Frobenius norm loss, where to goal is to find matrices $\mathbf{U} \in \{0,1\}^{n \times k}$ and $\mathbf{V} \in \{0,1\}^{k \times d}$ to minimize $\|\mathbf{UV} - \mathbf{A}\|_F^2$. Suppose optimal low-rank factors are $\mathbf{U}^* \in \{0,1\}^{n \times k}$ and $\mathbf{V}^* \in \{0,1\}^{k \times d}$, so that

$$\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_F^2 = \min_{\mathbf{U} \in \{0,1\}^{n \times k}, \mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{UV} - \mathbf{A}\|_F^2.$$

Observe that if we knew matrices $\mathbf{SU}^*$ and $\mathbf{SA}$ so that for all $\mathbf{V} \in \{0,1\}^{k \times d}$,

$$(1-\varepsilon)\|\mathbf{U}^*\mathbf{V} - \mathbf{A}\|_F^2 \leq \|\mathbf{SU}^*\mathbf{V} - \mathbf{SA}\|_F^2 \leq (1+\varepsilon)\|\mathbf{U}^*\mathbf{V} - \mathbf{A}\|_F^2,$$

then we could find a $(1+\varepsilon)$-approximate solution for $\mathbf{V}^*$ by solving the problem

$$\operatorname*{argmin}_{\mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{SU}^*\mathbf{V} - \mathbf{SA}\|_F^2$$

instead.

We would like to make guesses for the matrices $\mathbf{SU}^*$ and $\mathbf{SA}$, but first we must ensure there are not too many possibilities for these matrices. For example, if we chose $\mathbf{S}$ to be a dense matrix with random Gaussian entries, then $\mathbf{SU}^*$ could have too many possibilities because without additional information, there are 2^{nk} possibilities for the matrix $\mathbf{U}^* \in \{0,1\}^{n\times k}$. We can instead choose $\mathbf{S}$ to be a leverage score sampling matrix, which samples rows from $\mathbf{U}^*$ and $\mathbf{A}$. Since each row of $\mathbf{U}^*$ has dimension k, then there are at most 2^k distinct possibilities for each of the rows of $\mathbf{U}^*$. On the other hand, $\mathbf{A} \in \{0,1\}^{n\times d}$, so there may be 2^d distinct possibilities for the rows of $\mathbf{A}$, which is too many to guess.

Thus we first reduce the number of unique rows in $\mathbf{A}$ by computing a strong coreset $\widetilde{\mathbf{A}}$ for $\mathbf{A}$. The strong coreset has the property that for any choices of $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$, there exists $\mathbf{X} \in \{0,1\}^{n\times k}$ such that

$$(1-\varepsilon)\|\mathbf{UV}-\mathbf{A}\|_F^2 \le \|\mathbf{XV}-\widetilde{\mathbf{A}}\|_F^2 \le (1+\varepsilon)\|\mathbf{UV}-\mathbf{A}\|_F^2.$$

Therefore, we instead first solve the low-rank approximation problem on $\widetilde{\mathbf{A}}$ first. Crucially, $\widetilde{\mathbf{A}}$ has $2^{\mathrm{poly}(k/\varepsilon)}$ unique rows so then for a matrix $\mathbf{S}$ that samples $\mathrm{poly}(k/\varepsilon)$ rows, there are $\binom{2^{\mathrm{poly}(k/\varepsilon)}}{\mathrm{poly}(k/\varepsilon)} = 2^{\mathrm{poly}(k/\varepsilon)}$ possible choices of $\mathbf{S}\widetilde{\mathbf{A}}$, so we can enumerate all of them for both $\mathbf{SU}^*$ and $\mathbf{S}\widetilde{\mathbf{A}}$. We can then solve

$$\mathbf{V}' = \operatorname*{argmin}_{\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{SU}^*\mathbf{V} - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2$$

and

$$\mathbf{U}' = \operatorname*{argmin}_{\mathbf{U}\in\{0,1\}^{n\times k}} \|\mathbf{UV}' - \mathbf{A}\|_F^2,$$

where the latter optimization problem can be solved by iteratively optimizing over each row, so that the total computation time is $\mathcal{O}\left(2^k n\right)$ rather than 2^{kn}. We give the full algorithm in Algorithm 12 and the subroutine for optimizing with respect to $\widetilde{\mathbf{A}}$ in Algorithm 11. We give the subroutines for solving for $\mathbf{V}'$ and $\mathbf{U}'$ in Algorithm 9 and Algorithm 10, respectively.

Algorithm 9: Algorithm for computing optimal $\mathbf{V}$ given $\mathbf{U}$

Input: $\widetilde{\mathbf{A}} \in \{0,1\}^{N\times d}$, $\mathbf{U} \in \{0,1\}^{N\times k}$
Output: $\mathbf{V}' = \operatorname{argmin}_{\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F$

for $i = 1, \ldots, d$ **do** `// Optimize for each column individually`
  Set $\mathbf{V}'^{(i)} = \operatorname{argmin}_{\mathbf{V}^{(i)}\in\{0,1\}^{k\times 1}} \|\mathbf{UV}^{(i)} - \widetilde{\mathbf{A}}^{(i)}\|_2$; `// Enumerate over all` 2^k `possible binary vectors`
return $\mathbf{V}' = \left[\mathbf{V}'^{(1)} | \ldots | \mathbf{V}'^{(d)}\right]$;

First, we recall that leverage score sampling matrices preserve approximate matrix multiplication.

Algorithm 10: Algorithm for computing optimal $\mathbf{U}$ given $\mathbf{V}$

Input: $\widetilde{\mathbf{A}} \in \{0,1\}^{N \times d}$, $\mathbf{V} \in \{0,1\}^{k \times d}$
Output: $\mathbf{U}' = \operatorname{argmin}_{\mathbf{U} \in \{0,1\}^{N \times k}} \|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F$

for $i = 1, \ldots, N$ **do** `// Optimize for each row individually`
  Set $\mathbf{U}'_i = \operatorname{argmin}_{\mathbf{U}_i \in \{0,1\}^{1 \times k}} \|\mathbf{U}_i\mathbf{V} - \widetilde{\mathbf{A}}_i\|_2$; `// Enumerate over all` 2^k `possible binary vectors`
return $\mathbf{U}' = \mathbf{U}'_1 \circ \ldots \circ \mathbf{U}'_N$;

Algorithm 11: Low-rank approximation for matrix $\widetilde{\mathbf{A}}$ with t distinct rows

Input: $\widetilde{\mathbf{A}} \in \{0,1\}^{N \times d}$ with at most t distinct rows, rank parameter k, accuracy parameter $\varepsilon > 0$
Output: $\mathbf{U}' \in \{0,1\}^{n \times k}, \mathbf{V}' \in \{0,1\}^{k \times d}$ satisfying the property that $\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \leq (1+\varepsilon) \min_{\mathbf{U} \in \{0,1\}^{n \times k}, \mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F^2$

$V \leftarrow \emptyset$;
for *each guess of* $\mathbf{SU}^*$ *and* $\mathbf{S}\widetilde{\mathbf{A}}$, *where* $\mathbf{S}$ *is a leverage score sampling matrix with* $m = \mathcal{O}\left(\frac{k \log k}{\varepsilon^2}\right)$ *rows with weights that are powers of two up to* $\operatorname{poly}(N)$ **do**
  $V \leftarrow V \cup \operatorname{argmin}_{\mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{SU}^*\mathbf{V} - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2$; `// Algorithm 9`
for *each* $\mathbf{V} \in V$ **do**
  Let $\mathbf{U}_\mathbf{V} = \operatorname{argmin}_{\mathbf{U} \in \{0,1\}^{N \times k}} \|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F^2$; `// Algorithm 10`
$\mathbf{V}' \leftarrow \operatorname{argmin}_{\mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{SU}_\mathbf{V}\mathbf{V} - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2$;
$\mathbf{U}' \leftarrow \mathbf{U}_{\mathbf{V}'}$;
return $(\mathbf{U}', \mathbf{V}')$;

Lemma 4.3 (Lemma 32 in [82]). *Let* $\mathbf{U} \in \mathbb{R}^{N \times k}$ *have orthonormal columns,* $\widetilde{\mathbf{A}} \in \{0,1\}^{N \times d}$, *and* $\mathbf{S} \in \mathbb{R}^{m \times N}$ *be a leverage score sampling matrix for* $\mathbf{U}$ *with* $m = \mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ *rows. Then,*

$$\mathbf{Pr}\left[\|\mathbf{U}^\top\mathbf{S}^\top\mathbf{S}\widetilde{\mathbf{A}} - \mathbf{U}^\top\widetilde{\mathbf{A}}\|_F^2 < \varepsilon^2\|\mathbf{U}\|_F^2\|\widetilde{\mathbf{A}}\|_F^2\right] \geq 0.99.$$

Next, we recall that leverage score sampling matrices give subspace embeddings.

Theorem 4.4 (Theorem 42 in [82]). *For* $\mathbf{U} \in \mathbb{R}^{N \times k}$, *let* $\mathbf{S} \in \mathbb{R}^{m \times N}$ *be a leverage score sampling matrix for* $\mathbf{U} \in \{0,1\}^{N \times k}$ *with* $m = \mathcal{O}\left(\frac{k \log k}{\varepsilon^2}\right)$ *rows. Then with probability at least* 0.99, *we have for all* $\mathbf{V} \in \mathbb{R}^{k \times d}$,

$$(1-\varepsilon)\|\mathbf{UV}\|_F^2 \leq \|\mathbf{SUV}\|_F^2 \leq (1+\varepsilon)\|\mathbf{UV}\|_F^2.$$

Finally, we recall that approximate matrix multiplication and leverage score sampling suffices to achieve an affine embedding.

Theorem 4.5 (Theorem 39 in [82]). *Let* $\mathbf{U} \in \mathbb{R}^{N\times k}$ *have orthonormal columns. Let* $\mathbf{S}$ *be a sampling matrix that satisfies Lemma 4.3 with error parameter* $\frac{\varepsilon}{\sqrt{k}}$ *and also let* $\mathbf{S}$ *be a subspace embedding for* $\mathbf{U}$ *with error parameter* ε*. Let* $\mathbf{V}^* = \operatorname{argmin}_{\mathbf{V}} \|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F$ *and* $\mathbf{X} = \mathbf{UV}^* - \widetilde{\mathbf{A}}$*. Then for all* $\mathbf{V} \in \mathbb{R}^{k\times d}$*,*

$$(1-2\varepsilon)\|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{X}\|_F^2 \leq \|\mathbf{SUV} - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{SX}\|_F^2 \leq (1+2\varepsilon)\|\mathbf{UV} - \widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{X}\|_F^2.$$

We first show that Algorithm 11 achieves a good approximation to the optimal low-rank factors for the coreset $\widetilde{\mathbf{A}}$.

Lemma 4.6. *Suppose* $\varepsilon < \frac{1}{10}$*. Then with probability at least* 0.97*, the output of Algorithm 11 satisfies*

$$\|\mathbf{U}'\mathbf{V}' - \widetilde{\mathbf{A}}\|_F^2 \leq (1+6\varepsilon)\|\mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}\|_F^2.$$

Proof. Let $\mathbf{V}'' = \operatorname{argmin}_{\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{SU}^*\mathbf{V} - \widetilde{\mathbf{A}}\|_F^2$ and let $\mathbf{U}'' = \operatorname{argmin}_{\mathbf{U}\in\{0,1\}^{N\times k}} \|\mathbf{SUV}'' - \widetilde{\mathbf{A}}\|_F^2$ Since the algorithm chooses $\mathbf{U}'$ and $\mathbf{V}'$ over $\mathbf{U}''$ and $\mathbf{V}''$, then

$$\|\mathbf{U}'\mathbf{V}' - \widetilde{\mathbf{A}}\|_F^2 \leq \|\mathbf{U}''\mathbf{V}'' - \widetilde{\mathbf{A}}\|_F^2.$$

Due to the optimality of $\mathbf{U}''$,

$$\|\mathbf{U}''\mathbf{V}'' - \widetilde{\mathbf{A}}\|_F^2 \leq \|\mathbf{U}^*\mathbf{V}'' - \widetilde{\mathbf{A}}\|_F^2.$$

Let $\mathbf{X} = \mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}$. Note that since $\mathbf{U}^*$ has orthonormal columns, then by Lemma 4.3, the leverage score sampling matrix $\mathbf{S}$ achieves approximate matrix multiplication with probability at least 0.99. By Theorem 4.4, the matrix $\mathbf{S}$ also is a subspace embedding for $\mathbf{U}$. Thus, $\mathbf{S}$ meets the criteria for applying Theorem 4.5. Then for the correct guess $\mathbf{DT}$ of matrix $\mathbf{S}$ corresponding to $\mathbf{U}^*$ and conditioning on the correctness of $\mathbf{S}$ in Theorem 4.5,

$$\|\mathbf{U}^*\mathbf{V}'' - \widetilde{\mathbf{A}}\|_F^2 \leq \frac{1}{1-2\varepsilon}[\|\mathbf{SU}^*\mathbf{V}'' - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{SX}\|_F^2 + \|\mathbf{X}\|_F^2.]$$

Due to the optimality of $\mathbf{V}''$,

$$\begin{aligned}\frac{1}{1-2\varepsilon}[\|\mathbf{SU}^*\mathbf{V}'' - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{SX}\|_F^2 + \|\mathbf{X}\|_F^2] \\ \leq \frac{1}{1-2\varepsilon}[\|\mathbf{SU}^*\mathbf{V}^* - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{SX}\|_F^2 + \|\mathbf{X}\|_F^2].\end{aligned}$$

Then again conditioning on the correctness of $\mathbf{S}$,

$$\begin{aligned}\frac{1}{1-2\varepsilon}[\|\mathbf{SU}^*\mathbf{V}^* - \mathbf{S}\widetilde{\mathbf{A}}\|_F^2 - \|\mathbf{SX}\|_F^2 + \|\mathbf{X}\|_F^2] \\ \leq \frac{1}{1-2\varepsilon}[(1+2\varepsilon)\|\mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}\|_F^2 + \|\mathbf{SX}\|_F^2 - \|\mathbf{X}\|_F^2 - \|\mathbf{SX}\|_F^2 + \|\mathbf{X}\|_F^2] \\ \leq (1+6\varepsilon)\|\mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}\|_F^2,\end{aligned}$$

for sufficiently small ε, e.g., $\varepsilon < \frac{1}{10}$. Thus, putting things together, we have that conditioned on the correctness of $\mathbf{S}$ in Theorem 4.5,

$$\|\mathbf{U}'\mathbf{V}' - \widetilde{\mathbf{A}}\|_F^2 \leq (1+6\varepsilon)\|\mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}\|_F^2.$$

Since the approximate matrix multiplication property of Lemma 4.3, the subspace embedding property of Theorem 4.4, and the affine embedding property of Theorem 4.5 all fail with probability at most 0.01, then by a union bound, $\mathbf{S}$ succeeds with probability at least 0.97. □

We now analyze the runtime of the subroutine Algorithm 11.

Lemma 4.7. *Algorithm 11 uses* $2^{\mathcal{O}(m^2+m\log t)}\operatorname{poly}(N,d)$ *runtime for* $m = \mathcal{O}\left(\frac{k\log k}{\varepsilon^2}\right)$.

Proof. We analyze the number of possible guesses $\mathbf{D}$ and $\mathbf{T}$ corresponding to guesses of $\mathbf{S}\widetilde{\mathbf{A}}$ (see the remark after Theorem 4.2). There are at most $\binom{t}{m} = 2^{\mathcal{O}(m\log t)}$ distinct subsets of $m = \mathcal{O}\left(\frac{k\log k}{\varepsilon^2}\right)$ rows of $\widetilde{\mathbf{A}}$. Thus there are $2^{\mathcal{O}(m\log t)}$ possible matrices $\mathbf{T}$ that selects m rows of $\widetilde{\mathbf{A}}$, for the purposes of leverage score sampling. Assuming the leverage score sampling matrix does not sample any rows with leverage score less than $\frac{1}{\operatorname{poly}(N)}$, then there are $\mathcal{O}(\log N)^m = 2^{\mathcal{O}(m\log\log N)}$ total guesses for the matrix $\mathbf{D}$. Note that $\log n \leq 2^m$ implies that $2^{\mathcal{O}(m\log\log N)} \leq 2^{\mathcal{O}(m^2)}$ while $\log N > 2^m$ implies that $2^{\mathcal{O}(m\log\log N)} \leq 2^{\mathcal{O}(\log^2\log N)} \leq N$. Therefore, there are at most $2^{\mathcal{O}(m^2+m\log t)}N$ total guesses for all combinations of $\mathbf{T}$ and $\mathbf{D}$, corresponding to all guesses of $\mathbf{S}\widetilde{\mathbf{A}}$.

For each guess of $\mathbf{S}$ and $\mathbf{S}\widetilde{\mathbf{A}}$, we also need to guess $\mathbf{S}\mathbf{U}^*$. Since $\mathbf{U}^* \in \{0,1\}^{N\times k}$ is binary and $\mathbf{T}$ samples m rows before weighting each row with one of $\mathcal{O}(\log N)$ possible weights, the number of total guesses for $\mathbf{S}\mathbf{U}^*$ is $(2\cdot\mathcal{O}(\log N))^{mk}$.

Given guesses for $\mathbf{S}\mathbf{A}$ and $\mathbf{S}\mathbf{U}^*$, we can then compute $\operatorname{argmin}_{\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{S}\mathbf{U}^*\mathbf{V}-\mathbf{S}\mathbf{A}\|_F^2$ using $\mathcal{O}\left(2^k d\right)$ time through the subroutine Algorithm 9, which enumerates through all possible 2^k binary vectors for each column. For a fixed $\mathbf{V}$, we can then compute $\mathbf{U}_\mathbf{V} = \operatorname{argmin}_{\mathbf{U}\in\{0,1\}^{N\times k}}\|\mathbf{U}\mathbf{V}-\mathbf{A}\|_F^2$ using $\mathcal{O}\left(2^k N\right)$ time through the subroutine Algorithm 10, which enumerates through all possible 2^k binary vectors for each row of $\mathbf{U}_\mathbf{V}$. Therefore, the total runtime of Algorithm 11 is $2^{\mathcal{O}(m^2+m\log t)}\operatorname{poly}(N,d)$. □

We recall the following construction for a strong ε-coreset for k-means clustering.

Theorem 4.8 (Theorem 36 in [124])**.** *Let* $X \subset \mathbb{R}^d$ *be a subset of* n *points,* $\varepsilon \in (0,1)$ *be an accuracy parameter, and let* $t = \mathcal{O}\left(\frac{k^3\log^2 k}{\varepsilon^4}\right)$. *There exists an algorithm that uses* $\mathcal{O}\left(nd^2 + n^2 d + \frac{nkd}{\varepsilon^2} + \frac{nk^2}{\varepsilon^2}\right)$ *time and outputs a set of* t *weighted points that is a strong* ε*-coreset for* k*-means clustering with probability at least* 0.99. *Moreover, each point has an integer weight that is at most* $\operatorname{poly}(n)$.

We now justify the correctness of Algorithm 12.

Algorithm 12: Low-rank approximation for matrix $\mathbf{A}$

Input: $\mathbf{A} \in \{0,1\}^{n\times d}$, rank parameter k, accuracy parameter $\varepsilon > 0$
Output: $\mathbf{U}' \in \{0,1\}^{n\times k}, \mathbf{V}' \in \{0,1\}^{k\times d}$ satisfying the property that
$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \le (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2$

1 $t \leftarrow \mathcal{O}\left(\frac{2^{3k}k^2}{\varepsilon^4}\right)$; // Theorem 4.8 for 2^k-means clustering
2 Compute a strong coreset C for 2^k-means clustering of $\mathbf{A}$, with size t and total weight $N = \mathrm{poly}(n)$;
3 Let $\widetilde{\mathbf{A}} \in \{0,1\}^{N\times d}$ be the matrix representation of C, where weighted points are duplicated appropriately;
4 Let $(\widetilde{\mathbf{U}}, \widetilde{\mathbf{V}})$ be the output of Algorithm 11 on input $\widetilde{\mathbf{A}}$;
5 $\mathbf{U}' \leftarrow \mathrm{argmin}_{\mathbf{U}\in\{0,1\}^{n\times k}} \|\mathbf{U}\widetilde{\mathbf{V}} - \mathbf{A}\|_F^2$, $\mathbf{V}' \leftarrow \widetilde{\mathbf{V}}$; // Algorithm 10
6 **return** $(\mathbf{U}', \mathbf{V}')$;

Lemma 4.9. *With probability at least* 0.95*, Algorithm 12 returns* $\mathbf{U}', \mathbf{V}'$ *such that*

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \le (1+\varepsilon) \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2.$$

Proof. Let $\widetilde{\mathbf{M}}$ be the indicator matrix that selects a row of $\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} = \widetilde{\mathbf{U}}\mathbf{V}'$ to match to each row of $\mathbf{A}$, so that by the optimality of $\mathbf{U}'$,

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \le \|\widetilde{\mathbf{M}}\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \mathbf{A}\|_F^2.$$

Note that any $\mathbf{V}$ is a set of k points in $\{0,1\}^d$ and so each row $\mathbf{U}_i$ of $\mathbf{U}$ induces one of at most 2^k possible points $\mathbf{U}_i\mathbf{V} \in \{0,1\}^d$. Hence $\|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2$ is the objective value of a constrained 2^k-means clustering problem. Thus by the choice of t in Theorem 4.8, we have that $\widetilde{\mathbf{A}}$ is a strong coreset, so that

$$\|\widetilde{\mathbf{M}}\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \mathbf{A}\|_F^2 \le (1+\varepsilon)\|\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \widetilde{\mathbf{A}}\|_F^2.$$

Let $\mathbf{U}^* \in \{0,1\}^{n\times k}$ and $\mathbf{V}^* \in \{0,1\}^{k\times d}$ such that

$$\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_F^2 = \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2.$$

Let $\mathbf{M}^*$ be the indicator matrix that selects a row of $\mathbf{U}^*\mathbf{V}^*$ to match to each row of $\widetilde{\mathbf{A}}$, so that by Lemma 4.6,

$$(1+\varepsilon)\|\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \widetilde{\mathbf{A}}\|_F^2 \le (1+\varepsilon)^2\|\mathbf{M}^*\mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}\|_F^2.$$

Then by the choice of t in Theorem 4.8, we have that

$$(1+\varepsilon)^2\|\mathbf{M}^*\mathbf{U}^*\mathbf{V}^* - \widetilde{\mathbf{A}}\|_F^2 \le (1+\varepsilon)^3\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_F^2.$$

The desired claim then follows from rescaling ε. □

We now analyze the runtime of Algorithm 12.

Lemma 4.10. *Algorithm 12 uses* $2^{\tilde{\mathcal{O}}(k^2/\varepsilon^4)}\operatorname{poly}(n,d)$ *runtime.*

Proof. By Theorem 4.8, it follows that Algorithm 12 uses $\mathcal{O}\left(nd^2+n^2d+\frac{nkd}{\varepsilon^2}+\frac{nk^2}{\varepsilon^2}\right)$ time to compute $\widetilde{\mathbf{A}}\in\{0,1\}^{N\times d}$ with $N=\operatorname{poly}(n)$. By Lemma 4.7, it follows that Algorithm 11 on input $\widetilde{\mathbf{A}}$ thus uses runtime $2^{\mathcal{O}(m^2+m\log t)}\operatorname{poly}(N,d)$ for $m=\mathcal{O}\left(\frac{k\log k}{\varepsilon^2}\right)$ and $t=\mathcal{O}\left(\frac{2^{3k}k^2}{\varepsilon^4}\right)$. Finally, computing $\mathbf{U}'$ via Algorithm 10 takes $\mathcal{O}\left(2^k n\right)$ time after enumerating through all possible 2^k binary vectors for each row of $\mathbf{U}'$. Therefore, the total runtime of Algorithm 12 is $2^{\tilde{\mathcal{O}}\left(\frac{k^2\log^2 k}{\varepsilon^4}\right)}\operatorname{poly}(n,d)=2^{\tilde{\mathcal{O}}(k^2/\varepsilon^4)}\operatorname{poly}(n,d)$. □

Combining Lemma 4.9 and Lemma 4.10, we have:

Theorem 4.11. *There exists an algorithm that uses* $2^{\tilde{\mathcal{O}}(k^2/\varepsilon^4)}\operatorname{poly}(n,d)$ *runtime and with probability at least* $\frac{2}{3}$*, outputs* $\mathbf{U}'\in\{0,1\}^{n\times k}$ *and* $\mathbf{V}'\in\{0,1\}^{k\times d}$ *such that*

$$\|\mathbf{U}'\mathbf{V}'-\mathbf{A}\|_F^2\leq(1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{U}\mathbf{V}-\mathbf{A}\|_F^2.$$

4.3 $\mathbb{F}_2$ Low-Rank Approximation

In this section, we present a $(1+\varepsilon)$-approximation algorithm for binary low-rank approximation on $\mathbb{F}_2$, where to goal is to find matrices $\mathbf{U}\in\{0,1\}^{n\times k}$ and $\mathbf{V}\in\{0,1\}^{k\times d}$ to minimize the Frobenius norm loss $\|\mathbf{U}\mathbf{V}-\mathbf{A}\|_F^2$, but now all operations are performed in $\mathbb{F}_2$. We would like to use the same approach as in Section 4.2, i.e., to make guesses for the matrices $\mathbf{S}\mathbf{U}^*$ and $\mathbf{S}\mathbf{A}$ while ensuring there are not too many possibilities for these matrices. To do so for matrix operations over general integers, we chose $\mathbf{S}$ to be a leverage score sampling matrix that samples rows from $\mathbf{U}^*$ and $\mathbf{A}$. We then used the approximate matrix multiplication property in Lemma 4.3 and the subspace embedding property in Theorem 4.4 to show that $\mathbf{S}$ provides an affine embedding in Theorem 4.5 over general integers. However, it no longer necessarily seems true that $\mathbf{S}$ will provide an affine embedding over $\mathbb{F}_2$, in part because the subspace embedding property of $\mathbf{S}$ computes leverage scores of each row of $\mathbf{U}^*$ and $\mathbf{A}$ with respect to general integers. Thus we require an alternate approach for matrix operations over $\mathbb{F}_2$.

Instead, we form the matrix $\widetilde{\mathbf{A}}$ by taking a strong coreset of $\mathbf{A}$ and then duplicating the rows according to their weight w_i to form $\widetilde{\mathbf{A}}$. That is, if the ith row $\mathbf{A}_i$ of $\mathbf{A}$ is sampled with weight w_i in the coreset, then $\widetilde{\mathbf{A}}$ will contain w_i repetitions of the row $\mathbf{A}_i$, where we note that w_i is an integer. We then group the rows of $\widetilde{\mathbf{A}}$ by their repetitions, so that group $\mathbf{G}_j$ consists of the rows of $\widetilde{\mathbf{A}}$ that are repeated $[(1+\varepsilon)^j,(1+\varepsilon)^{j+1})$ times. Thus if $\mathbf{A}_i$ appears w_i times in $\widetilde{\mathbf{A}}$, then it appears a single time in group $\mathbf{G}_j$ for $j=\lfloor\log w_i\rfloor$.

We perform entry-wise L_0 low-rank approximation over $\mathbb{F}_2$ for each of the groups $\mathbf{G}_j$, which gives low-rank factors $\mathbf{U}^{(j)}$ and $\mathbf{V}^{(j)}$. We then compute $\widetilde{\mathbf{U}^{(j)}}\in\mathbb{R}^{n\times d}$ from $\mathbf{U}^{(j)}$ by following procedure. If $\mathbf{A}_i$ is in $\mathbf{G}_j$, then we place the row of $\mathbf{U}^{(j)}$ corresponding to $\mathbf{A}_i$

into the ith row of $\widetilde{\mathbf{U}^{(j)}}$, for all $i \in [n]$. Note that the row of $\mathbf{U}^{(j)}$ corresponding to $\mathbf{A}_i$ may not be the ith row of $\mathbf{U}^{(j)}$, e.g., since $\mathbf{A}_i$ will appear only once in $\mathbf{G}_j$ even though it appears $w_i \in [(1+\varepsilon)^j, (1+\varepsilon)^{j+1})$ times in $\mathbf{A}$. Otherwise if $\mathbf{A}_i$ is not in $\mathbf{G}_j$, then we set ith row of $\widetilde{\mathbf{U}^{(j)}}$ to be the all zeros row. We then achieve $\mathbf{V}^{(j)}$ by padding accordingly. Finally, we collect

$$\widetilde{\mathbf{U}} = \left[\widetilde{\mathbf{U}^{(0)}} | \ldots | \widetilde{\mathbf{U}^{(\ell)}}\right], \qquad \widetilde{\mathbf{V}} \leftarrow \widetilde{\mathbf{V}^{(0)}} \circ \ldots \circ \widetilde{\mathbf{V}^{(i)}}$$

to achieve bicriteria low-rank approximations $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ to $\widetilde{\mathbf{A}}$. Finally, to achieve bicriteria low-rank approximations $\mathbf{U}'$ and $\mathbf{V}'$ to $\mathbf{A}$, we require that $\mathbf{U}'$ achieves the same block structure as $\widetilde{\mathbf{U}}$. We describe this subroutine in Algorithm 13 and we give the full low-rank approximation bicriteria algorithm in Algorithm 14.

We first recall the following subroutine to achieve entry-wise L_0 low-rank approximation over $\mathbb{F}_2$. Note that for matrix operations over $\mathbb{F}_2$, we have that the entry-wise L_0 norm is the same as the entry-wise L_p norm for all p.

Lemma 4.12 (Theorem 3 in [32]). *For $\varepsilon \in (0,1)$, there exists a $(1+\varepsilon)$-approximation algorithm to entry-wise L_0 rank-k approximation over $\mathbb{F}_2$ running in $d \cdot n^{\mathrm{poly}(k/\varepsilon)}$ time.*

Algorithm 13: Algorithm for computing optimal $\mathbf{U}$ given $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(\ell)}$

Input: $\widetilde{\mathbf{A}} \in \{0,1\}^{N \times d}$, $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(\ell)} \in \{0,1\}^{k \times d}$
Output: $\mathbf{U}' = \mathrm{argmin}_{\mathbf{U} \in \{0,1\}^{N \times \ell k}} \|\mathbf{U}\mathbf{V} - \widetilde{\mathbf{A}}\|_F$, where $\mathbf{U}$ is restricted to one nonzero block of k coordinates

1 **for** $i = 1, \ldots, N$ **do**
2 Set $(\mathbf{U}'_i, j') = \mathrm{argmin}_{\mathbf{U}_i \in \{0,1\}^{1 \times k}, j \in [\ell]} \|\mathbf{U}_i \mathbf{V}^{(j)} - \widetilde{\mathbf{A}}_i\|_2$; `// Enumerate over all` 2^k `possible binary vectors, all` ℓ `indices`
3 Pad $\mathbf{U}'_i$ with length ℓk, as the j'th block of k coordinates;
4 **return** $\mathbf{U}' = \mathbf{U}'_1 \circ \ldots \circ \mathbf{U}'_N$;

We first justify the correctness of Algorithm 14.

Lemma 4.13. *With probability at least 0.95, Algorithm 14 returns $\mathbf{U}', \mathbf{V}'$ such that*

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \leq (1+\varepsilon) \min_{\mathbf{U} \in \{0,1\}^{n \times k}, \mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2,$$

where all matrix operations are performed in $\mathbb{F}_2$.

Proof. Let $\widetilde{\mathbf{U}} \leftarrow \left[\widetilde{\mathbf{U}^{(0)}} | \ldots | \widetilde{\mathbf{U}^{(\ell)}}\right]$ in Algorithm 14. Let $\widetilde{\mathbf{M}}$ be the indicator matrix that selects a row of $\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} = \widetilde{\mathbf{U}}\mathbf{V}'$ to match to each row of $\mathbf{A}$, so that by the optimality of $\mathbf{U}'$,

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \leq \|\widetilde{\mathbf{M}}\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \mathbf{A}\|_F^2.$$

Algorithm 14: Bicriteria low-rank approximation on $\mathbb{F}_2$ for matrix $\mathbf{A}$

Input: $\mathbf{A} \in \{0,1\}^{n\times d}$, rank parameter k, accuracy parameter $\varepsilon > 0$

Output: $\mathbf{U}' \in \{0,1\}^{n\times k}, \mathbf{V}' \in \{0,1\}^{k\times d}$ satisfying the property that $\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_F^2 \le (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2$, where all matrix operations are performed in $\mathbb{F}_2$

$\ell \leftarrow \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$, $t \leftarrow \mathcal{O}\left(\frac{(2^k\ell)^3k^2}{\varepsilon^4}\right)$, $k' \leftarrow \ell k$; `// Theorem 4.8 for` 2^k`-means clustering`

Compute a strong coreset C for 2^k-means clustering of $\mathbf{A}$, with size t and total weight $N = \text{poly}(n)$;

Let $\widetilde{\mathbf{A}} \in \{0,1\}^{N\times d}$ be the matrix representation of C, where weighted points are duplicated appropriately;

For $i \in [\ell]$, let $\mathbf{G}^{(i)}$ be the group of rows (removing multiplicity) of $\widetilde{\mathbf{A}}$ with frequency $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$;

Let $(\widetilde{\mathbf{U}^{(i)}}, \widetilde{\mathbf{V}^{(i)}})$ be the output of Lemma 4.12 on input $\mathbf{G}^{(i)}$, padded to $\mathbb{R}^{n\times k}$ and $\mathbb{R}^{k\times d}$, respectively;

$\widetilde{\mathbf{V}} \leftarrow \widetilde{\mathbf{V}^{(0)}} \circ \ldots \circ \widetilde{\mathbf{V}^{(\ell)}}$;

Use Algorithm 13 with $\widetilde{\mathbf{V}^{(0)}}, \ldots, \widetilde{\mathbf{V}^{(\ell)}}$ and $\mathbf{A}$ to find $\mathbf{U}'$;

return $(\mathbf{U}', \mathbf{V}')$ with $\mathbf{V}' = \widetilde{\mathbf{V}}$;

Since $\mathbf{V}$ is a set of k points in $\{0,1\}^d$ and each row $\mathbf{U}_i$ of $\mathbf{U}$ induces one of at most 2^k possible points $\mathbf{U}_i\mathbf{V} \in \{0,1\}^d$, then $\|\mathbf{U}\mathbf{V} - \mathbf{A}\|_F^2$ is the objective value of a constrained 2^k-means clustering problem, even when all operations performed are on $\mathbb{F}_2$. Similarly, $\mathbf{V}^{(j)}$ is a set of k points in $\{0,1\}^d$ for each $j \in [\ell]$. Each row $\mathbf{U}_i$ of $\mathbf{U}$ induces one of at most 2^k possible points $\mathbf{U}_i\mathbf{V}^{(j)} \in \{0,1\}^d$ for a fixed $j \in [\ell]$, so that $\|\mathbf{U}\mathbf{V}' - \mathbf{A}\|_F^2$ is the objective value of a constrained $2^k\ell$-means clustering problem, even when all operations performed are on $\mathbb{F}_2$.

Hence by the choice of t in Theorem 4.8, it follows that $\widetilde{\mathbf{A}}$ is a strong coreset, and so

$$\|\widetilde{\mathbf{M}}\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \mathbf{A}\|_F^2 \le (1+\varepsilon)\|\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \widetilde{\mathbf{A}}\|_F^2.$$

We decompose the rows of $\widetilde{\mathbf{A}}$ into $\mathbf{G}^{(0)}, \ldots, \mathbf{G}^{(\ell)}$ for $\ell = \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$. Let G_i be the corresponding indices in $[n]$ so that $j \in G_i$ if and only if $\widetilde{\mathbf{A}_j}$ is nonzero in $\mathbf{G}_i$. Then we have

$$\|\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \widetilde{\mathbf{A}}\|_F^2 = \sum_{i\in[\ell]}\sum_{j\in G_i} \|\mathbf{U}'_j\mathbf{V}' - \widetilde{\mathbf{A}_j}\|_F^2.$$

Since each row in G_i is repeated a number of times in $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$, then

$$\sum_{j\in G_i} \|\mathbf{U}'_j\mathbf{V}' - \widetilde{\mathbf{A}_j}\|_F^2 \le (1+\varepsilon)^2 \min_{\mathbf{U}^{(i)}\in\{0,1\}^{n\times k},\mathbf{V}^{(i)}\in\{0,1\}^{\times k\times d}} \|\mathbf{U}^{(i)}\mathbf{V}^{(i)} - \mathbf{G}^{(i)}\|_F^2,$$

where the first factor of $(1+\varepsilon)$ is from the $(1+\varepsilon)$-approximation guarantee of $\mathbf{U}^{(i)}$ and $\mathbf{V}^{(i)}$ by Lemma 4.12 and the second factor of $(1+\varepsilon)$ is from the number of each row in $\mathbf{G}^{(i)}$ varying by at most a $(1+\varepsilon)$ factor. Therefore,

$$\begin{aligned}\|\mathbf{U}'\mathbf{V}'-\mathbf{A}\|_F^2 &\leq (1+\varepsilon)^3 \sum_{i\in[\ell]} \min_{\mathbf{U}^{(i)}\in\{0,1\}^{n\times k},\mathbf{V}^{(i)}\in\{0,1\}^{k\times d}} \|\mathbf{U}^{(i)}\mathbf{V}^{(i)}-\mathbf{G}^{(i)}\|_F^2 \\ &\leq (1+\varepsilon)^3 \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V}-\widetilde{\mathbf{A}}\|_F^2.\end{aligned}$$

Let $\mathbf{U}^*\in\{0,1\}^{n\times k}$ and $\mathbf{V}^*\in\{0,1\}^{k\times d}$ such that

$$\|\mathbf{U}^*\mathbf{V}^*-\mathbf{A}\|_F^2 = \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V}-\mathbf{A}\|_F^2,$$

where all operations are performed in $\mathbb{F}_2$. Let $\mathbf{M}^*$ be the indicator matrix that selects a row of $\mathbf{U}^*\mathbf{V}^*$ to match to each row of $\widetilde{\mathbf{A}}$, so that by Lemma 4.6,

$$\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V}-\widetilde{\mathbf{A}}\|_F^2 \leq (1+\varepsilon)\|\mathbf{M}^*\mathbf{U}^*\mathbf{V}^*-\widetilde{\mathbf{A}}\|_F^2.$$

Then by the choice of t in Theorem 4.8 so that $\widetilde{\mathbf{A}}$ is a strong coreset of $\mathbf{A}$,

$$\|\mathbf{M}^*\mathbf{U}^*\mathbf{V}^*-\widetilde{\mathbf{A}}\|_F^2 \leq (1+\varepsilon)\|\mathbf{U}^*\mathbf{V}^*-\mathbf{A}\|_F^2.$$

Therefore, we have

$$\|\mathbf{U}'\mathbf{V}'-\mathbf{A}\|_F^2 \leq (1+\varepsilon)^5\|\mathbf{U}^*\mathbf{V}^*-\mathbf{A}\|_F^2$$

and the desired claim then follows from rescaling ε. □

It remains to analyze the runtime of Algorithm 14.

Lemma 4.14. *Algorithm 14 uses* $2^{\mathrm{poly}(k/\varepsilon)}\,\mathrm{poly}(n,d)$ *runtime.*

Proof. By Theorem 4.8, we have that Algorithm 14 uses $\mathcal{O}\left(nd^2+n^2d+\frac{nkd}{\varepsilon^2}+\frac{nk^2}{\varepsilon^2}\right)$ time to compute $\widetilde{\mathbf{A}}\in\{0,1\}^{N\times d}$ with $N=\mathrm{poly}(n)$. By Lemma 4.12, it takes $d\cdot(2^k)^{\mathrm{poly}(k/\varepsilon)}$ time to compute $\widetilde{\mathbf{U}^{(i)}},\widetilde{\mathbf{V}^{(i)}}$ for each $i\in[\ell]$ for $\ell=\mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$. Hence, it takes $2^{\mathrm{poly}(k/\varepsilon)}\,\mathrm{poly}(n,d)$ runtime to compute $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$. Finally, computing $\mathbf{U}'$ via Algorithm 13 takes $\mathcal{O}\left(2^{k'}n\right)$ time after enumerating through all possible $2^k\ell$ binary vectors for each row of $\mathbf{U}'$. Therefore, the total runtime of Algorithm 12 is $2^{\mathrm{poly}(k/\varepsilon)}\,\mathrm{poly}(n,d)$. □

By Lemma 4.13 and Lemma 4.14, we thus have:

Theorem 4.15. *There exists an algorithm that uses* $2^{\mathrm{poly}(k/\varepsilon)}\,\mathrm{poly}(n,d)$ *runtime and with probability at least* $\frac{2}{3}$*, outputs* $\mathbf{U}'\in\{0,1\}^{n\times k'}$ *and* $\mathbf{V}'\in\{0,1\}^{k'\times d}$ *such that*

$$\|\mathbf{U}'\mathbf{V}'-\mathbf{A}\|_F^2 \leq (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V}-\mathbf{A}\|_F^2,$$

where $k'=\mathcal{O}\left(\frac{k\log k}{\varepsilon}\right)$.

4.4 L_p Low-Rank Approximation

In this section, we present a $(1+\varepsilon)$-approximation algorithm for binary low-rank approximation with L_p loss, where to goal is to find matrices $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$ to minimize $\|\mathbf{UV}-\mathbf{A}\|_p^p$. We would like to use the same approach as in Section 4.2, where we first compute a weighted matrix $\widetilde{\mathbf{A}}$ from a strong coreset for $\mathbf{A}$, and then we make guesses for the matrices $\mathbf{SU}^*$ and $\mathbf{SA}$ and solve for $\min_{\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{SU}^*\mathbf{V}-\mathbf{SA}\|_F^2$ while ensuring there are not too many possibilities for the matrices $\mathbf{SU}^*$ and $\mathbf{SA}$. Thus to adapt this approach to L_p loss, we first require the following strong coreset construction for discrete metrics:

Theorem 4.16 (Theorem 1 in [91]). *Let $X \subset \mathbb{R}^d$ be a subset of n points, $\varepsilon \in (0,1)$ be an accuracy parameter, $p \geq 1$ be a constant, and let*

$$t = \mathcal{O}\left(\min(\varepsilon^{-2}+\varepsilon^{-p}, k\varepsilon^{-2})\cdot k\log n\right).$$

There exists an algorithm that uses $\mathrm{poly}(n,d,k)$ *runtime and outputs a set of t weighted points that is a strong ε-coreset for k-clustering on discrete L_p metrics with probability at least* 0.99. *Moreover, each point has an integer weight that is at most* $\mathrm{poly}(n)$.

For Frobenius error, we crucially require the affine embedding property that

$$(1-\varepsilon)\|\mathbf{U}^*\mathbf{V}-\mathbf{A}\|_F^2 \leq \|\mathbf{SU}^*\mathbf{V}-\mathbf{SA}\|_F^2 \leq (1+\varepsilon)\|\mathbf{U}^*\mathbf{V}-\mathbf{A}\|_F^2,$$

for all $\mathbf{V} \in \{0,1\}^{k\times d}$. Unfortunately, it is not known whether there exists an efficient sampling-based affine embedding for L_p loss.

We instead invoke the coreset construction of Theorem 4.16 on the rows and the columns so that $\widetilde{\mathbf{A}}$ has a small number of distinct rows and columns. We again use the idea from Section 4.3 to partition the rows of $\widetilde{\mathbf{A}}$ into groups based on their frequency, but now we further partition the groups based on the frequency of the columns. It then remains to solve BMF with L_p loss on the partition, each part of which has a small number of rows and columns. Because the contribution of each row toward the overall loss is small (because there is a small number of columns), it turns out that there exists a matrix that samples $\mathrm{poly}(k/\varepsilon)$ rows of each partition that finally achieves the desired affine embedding. Thus, we can solve the problem on each partition, pad the factors accordingly, and build the bicriteria factors as in the binary field case. The algorithm appears in full in Algorithm 17, with subroutines appearing in Algorithm 15 and Algorithm 16.

We first justify the correctness of Algorithm 16 by showing the existence of an L_0 sampling matrix $\mathbf{S}$ that achieves a subspace embedding for binary inputs.

Lemma 4.17. *Given matrices $\mathbf{A} \in \{0,1\}^{n\times k}$ and $\mathbf{B} \in \{0,1\}^{n\times r}$, there exists a matrix $\mathbf{S} \in \mathbb{R}^{m\times n}$ with $m = \mathcal{O}\left(\frac{k^{p+1}}{\varepsilon^2}\log r\right)$ such that with probability at least* 0.99, *we have that simultaneously for all $\mathbf{X} \in \{0,1\}^{k\times r}$,*

$$(1-\varepsilon)\|\mathbf{AX}-\mathbf{B}\|_p^p \leq \|\mathbf{SAX}-\mathbf{SB}\|_p^p \leq (1+\varepsilon)\|\mathbf{AX}-\mathbf{B}\|_p^p.$$

Algorithm 15: Algorithm for computing optimal $\mathbf{U}$ given $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(\ell)}$

Input: $\widetilde{\mathbf{A}} \in \{0,1\}^{N \times d}$, $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(\ell)} \in \{0,1\}^{k \times d}$
Output: $\mathbf{U}' = \operatorname{argmin}_{\mathbf{U} \in \{0,1\}^{N \times \ell k}} \|\mathbf{U}\mathbf{V} - \widetilde{\mathbf{A}}\|_p^p$, where $\mathbf{U}$ is restricted to one nonzero block of k coordinates

for $i = 1, \ldots, N$ **do**
  Set $(\mathbf{U}'_i, j') = \operatorname{argmin}_{\mathbf{U}_i \in \{0,1\}^{1 \times k}, j \in [\ell]} \|(\mathbf{U}_i \mathbf{V}^{(j)} - \widetilde{\mathbf{A}}_i\|_p^p$; `// Enumerate over all` 2^k `possible binary vectors, all` ℓ `indices`
  Pad $\mathbf{U}'_i$ with length ℓk, as the j'th block of k coordinates;
return $\mathbf{U}' = \mathbf{U}'_1 \circ \ldots \circ \mathbf{U}'_N$;

Algorithm 16: Low-rank approximation for matrix $\widetilde{\mathbf{A}}$ with t distinct rows and t' distinct columns

Input: $\widetilde{\mathbf{A}} \in \{0,1\}^{N \times D}$ with at most t distinct rows and r distinct columns
Output: $\mathbf{U}', \mathbf{V}'$ with $\|\mathbf{U}\mathbf{V} - \widetilde{\mathbf{A}}\|_p \leq (1+\varepsilon) \min_{\mathbf{U} \in \{0,1\}^{N \times k}, \mathbf{V} \in \{0,1\}^{k \times D}} \|\mathbf{U}\mathbf{V} - \widetilde{\mathbf{A}}\|_p$

$V \leftarrow \emptyset$;
for *each guess of* $\mathbf{S}\mathbf{U}^*$ *and* $\mathbf{S}\mathbf{A}$, *where* $\mathbf{S}$ *is a* L_0 *sampling matrix with* $m = \mathcal{O}\left(\frac{k^{p+1}}{\varepsilon^2} \log r\right)$ *rows with weights that are powers of two up to* $\operatorname{poly}(N)$ **do**
  $V \leftarrow V \cup \operatorname{argmin}_{\mathbf{V} \in \{0,1\}^{k \times D}} \|\mathbf{S}\mathbf{U}^*\mathbf{V} - \mathbf{S}\mathbf{A}\|_p^p$; `// Algorithm 9 with` L_p `loss`
for *each* $\mathbf{V} \in V$ **do**
  Let $\mathbf{U}_{\mathbf{V}} = \operatorname{argmin}_{\mathbf{U} \in \{0,1\}^{N \times k}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_p^p$; `// Algorithm 10 with` L_p `loss`
$\mathbf{V}' \leftarrow \operatorname{argmin}_{\mathbf{V} \in \{0,1\}^{k \times d}} \|\mathbf{S}\mathbf{U}_{\mathbf{V}}\mathbf{V} - \mathbf{S}\mathbf{A}\|_p^p$;
$\mathbf{U}' \leftarrow \mathbf{U}_{\mathbf{V}'}$;
return $(\mathbf{U}', \mathbf{V}')$;

Proof. Let $\mathbf{M} \in \{0, 1, \ldots, k\}^{n \times 1}$ be an arbitrary matrix and let S be a set that contains the nonzero rows of $\mathbf{M}$ and has cardinality that is a power of two. That is, $|S| = 2^i$ for some integer $i \geq 0$. Let $\mathbf{z}$ be a random element of S, i.e., a random non-zero row of $\mathbf{M}$, so that we have

$$\mathbb{E}\left[2^i \cdot \|\mathbf{z}\|_p^p\right] = \|\mathbf{M}\|_p^p.$$

Similarly, we have

$$\mathbb{V}\left[(\right] 2^i \cdot \|\mathbf{z}\|_p^p) \leq 2^i k^p \leq 2k^p \|\mathbf{M}\|_p^p.$$

Hence if we repeat take the mean of $\mathcal{O}\left(\frac{k^p}{\varepsilon^2}\right)$ estimators, we have that with probability at least 0.99,

$$(1-\varepsilon)\|\mathbf{M}\|_p^p \leq \|\mathbf{S}\mathbf{M}\|_p^p \leq (1+\varepsilon)\|\mathbf{M}\|_p^p.$$

We can further improve the probability of success to $1-\delta$ for $\delta \in (0,1)$ by repeating $\mathcal{O}\left(\log \frac{1}{\delta}\right)$ times. By setting $\mathbf{M} = \mathbf{A}\mathbf{x} - \mathbf{B}^{(i)}$ for fixed $\mathbf{A} \in \{0,1\}^{n \times k}$, $\mathbf{x} \in \{0,1\}^k$, and $\mathbf{B} \in \{0,1\}^{n \times r}$ with $i \in [r]$, we have that the sketch matrix gives a $(1+\varepsilon)$-approximation

Algorithm 17: Bicriteria low-rank approximation with L_p loss for matrix $\mathbf{A}$

Input: $\mathbf{A} \in \{0,1\}^{n\times d}$, rank parameter k, accuracy parameter $\varepsilon > 0$
Output: $\mathbf{U}' \in \{0,1\}^{n\times k}, \mathbf{V}' \in \{0,1\}^{k\times d}$ satisfying the property that $\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_p^p \le (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_p^p$

$t \leftarrow \mathcal{O}\left(\min(\varepsilon^{-2}+\varepsilon^{-p}, k\varepsilon^{-2})\cdot k\log n\right)$; // Theorem 4.16

$\ell \leftarrow \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$, $k' \leftarrow \ell k$;

Compute a strong coreset C for 2^k-means clustering of $\mathbf{A}$, with t rows, with weights $N = \text{poly}(n)$;

Compute a strong coreset C' for 2^k-means clustering of C, with t rows and columns, with weights $N, D = \text{poly}(n)$;

Let $\widetilde{\mathbf{A}} \in \{0,1\}^{N\times D}$ be the matrix representation of C, where weighted points are duplicated appropriately;

For $i \in [\ell]$, let $\mathbf{G}^{(i)}$ be the group of rows (removing multiplicity) of $\widetilde{\mathbf{A}}$ with frequency $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$;

For $i, j \in [\ell]$, let $\mathbf{G}^{(i,j)}$ be the group of columns (removing multiplicity) of $\mathbf{G}^{(i,j)}$ with frequency $[(1+\varepsilon)^j, (1+\varepsilon)^{j+1})$;

Compute the low-rank minimizers $(\widetilde{\mathbf{U}^{(i,j)}}, \widetilde{\mathbf{V}^{(i,j)}})$ on input $\mathbf{G}^{(i,j)}$ using Algorithm 16, padded to $\mathbb{R}^{n\times k}$ and $\mathbb{R}^{k\times D}$, respectively;

$\widetilde{\mathbf{U}} \leftarrow \left[\widetilde{\mathbf{U}^{(0,0)}}|\widetilde{\mathbf{U}^{(1,0)}}|\ldots|\widetilde{\mathbf{U}^{(\ell,\ell)}}\right]$, $\widetilde{\mathbf{V}} \leftarrow \widetilde{\mathbf{V}^{(0,0)}} \circ \widetilde{\mathbf{V}^{(1,0)}} \ldots \circ \widetilde{\mathbf{V}^{(\ell,\ell)}}$;

Use Algorithm 15 with $\widetilde{\mathbf{U}^{(0,0)}}, \widetilde{\mathbf{U}^{(1,0)}} \ldots, \widetilde{\mathbf{U}^{(\ell,\ell)}}$ and C to find $\mathbf{V}'$;

Use $\mathbf{V}'$ and $\mathbf{A}$ to find $\mathbf{U}'$, i.e., Algorithm 10 with dimension k' and L_p loss;

Return $(\mathbf{U}', \mathbf{V}')$;

to $\|\mathbf{A}\mathbf{x} - \mathbf{B}^{(i)}\|_p^p$. The result then follows from setting $\delta = \frac{1}{2^k r}$, taking a union bound over all $\mathbf{x} \in \{0,1\}^k$, and then a union bound over all $i \in [r]$. □

We then justify the correctness of Algorithm 17.

Lemma 4.18. *With probability at least* 0.95, *Algorithm 17 returns* $\mathbf{U}', \mathbf{V}'$ *such that*

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_p^p \le (1+\varepsilon)\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_p^p.$$

Proof. Let $\mathbf{M}_1$ and $\mathbf{M}_2$ be the sampling and rescaling matrices used to acquire $\widetilde{\mathbf{A}} \in \mathbb{R}^{N\times D}$, so that by the optimality of $\mathbf{U}'$,

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_p^p \le \|\mathbf{M}_1\widetilde{\mathbf{U}}\widetilde{\mathbf{V}}\mathbf{M}_2 - \mathbf{A}\|_p^p.$$

Observe that $\mathbf{V}$ is a set of k points in $\{0,1\}^d$. Thus, each row $\mathbf{U}_i$ of $\mathbf{U}$ induces one of at most 2^k possible points $\mathbf{U}_i\mathbf{V} \in \{0,1\}^d$. Hence, $\|\mathbf{U}\mathbf{V} - \mathbf{A}\|_p^p$ is the objective value of a constrained 2^k-clustering problem under the L_p metric. Similarly, since $\mathbf{V}^{(j)}$ is a

set of k points in $\{0,1\}^d$ for each $j \in [\ell]$, then each row $\mathbf{U}_i$ of $\mathbf{U}$ induces one of at most 2^k possible points $\mathbf{U}_i\mathbf{V}^{(j)} \in \{0,1\}^d$ for a fixed $j \in [\ell]$. Therefore, $\|\mathbf{UV}' - \mathbf{A}\|_p^p$ is the objective value of a constrained $2^k\ell$-clustering problem under the L_p metric.

By the choice of t in Theorem 4.16, $\widetilde{\mathbf{A}}$ is a strong coreset, and so

$$\|\mathbf{M}_1\widetilde{\mathbf{U}}\widetilde{\mathbf{V}}\mathbf{M}_2 - \mathbf{A}\|_F^2 \leq (1+\varepsilon)\|\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \widetilde{\mathbf{A}}\|_F^2.$$

We decompose the rows of $\widetilde{\mathbf{A}}$ into groups $\mathbf{G}^{(0)}, \ldots, \mathbf{G}^{(\ell)}$ for $\ell = \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$. For each group $\mathbf{G}^{(i)}$, we decompose the columns of $\mathbf{G}^{(i)}$ into groups $\mathbf{G}^{(i,0)}, \ldots, \mathbf{G}^{(i,\ell)}$ for $\ell = \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$. Let G_i be the indices in $[n]$ corresponding to the rows in $\mathbf{G}^{(i)}$ and let $G_{i,j}$ be the indices in $[n]$ corresponding to the columns in $\mathbf{G}^{(i,j)}$. Then

$$\|\widetilde{\mathbf{U}}\widetilde{\mathbf{V}} - \widetilde{\mathbf{A}}\|_p^p = \sum_{i\in[\ell]}\sum_{a\in G_i}\sum_{j\in[\ell]}\sum_{b\in G_{i,j}}\left|(\mathbf{U}'\mathbf{V}')_{a,b} - \widetilde{\mathbf{A}}_{a,b}\right|^p.$$

Since each row in G_i is repeated a number of times in $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ and each column in $G_{i,j}$ is repeated a number of times in $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$, then

$$\begin{aligned}&\sum_{a\in G_i}\sum_{b\in G_{i,j}}\left|(\mathbf{U}'\mathbf{V}')_{a,b} - \widetilde{\mathbf{A}}_{a,b}\right|^p \\ &\qquad \leq (1+\varepsilon)^3 \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{\times k\times d}}\sum_{a\in G_i}\sum_{b\in G_{i,j}}\left|(\mathbf{UV})_{a,b} - \widetilde{\mathbf{A}}_{a,b}\right|^p,\end{aligned}$$

where the first factor of $(1+\varepsilon)$ is from the $(1+\varepsilon)$-approximation guarantee of $\mathbf{U}^{(i)}$ and $\mathbf{V}^{(i)}$ by Lemma 4.12 and the second and third factors of $(1+\varepsilon)$ is from the number of each row and each column in $\mathbf{G}^{(i,j)}$ varying by at most $(1+\varepsilon)$ factor. Therefore,

$$\begin{aligned}\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_p^p &\leq (1+\varepsilon)\sum_{i\in[\ell]}\sum_{a\in G_i}\sum_{j\in[\ell]}\sum_{b\in G_{i,j}}\left|(\mathbf{U}'\mathbf{V}')_{a,b} - \widetilde{\mathbf{A}}_{a,b}\right|^p \\ &\leq (1+\varepsilon)^4 \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{UV} - \widetilde{\mathbf{A}}\|_p^p\end{aligned}$$

Let $\mathbf{U}^* \in \{0,1\}^{n\times k}$ and $\mathbf{V}^* \in \{0,1\}^{k\times d}$ be minimizers to the binary L_p low-rank approximation problem, so that

$$\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_p^p = \min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{UV} - \mathbf{A}\|_p^p.$$

Let $\mathbf{M}_3$ and $\mathbf{M}_4$ be the indicator matrices that select rows and columns of $\mathbf{U}^*\mathbf{V}^*$ to match to each row of $\widetilde{\mathbf{A}}$, so that by Lemma 4.6,

$$\min_{\mathbf{U}\in\{0,1\}^{n\times k},\mathbf{V}\in\{0,1\}^{k\times d}}\|\mathbf{UV} - \widetilde{\mathbf{A}}\|_p^p \leq (1+\varepsilon)\|\mathbf{M}_3\mathbf{U}^*\mathbf{V}^*\mathbf{M}_4 - \widetilde{\mathbf{A}}\|_p^p.$$

Then by the choice of t in Theorem 4.16 so that $\widetilde{\mathbf{A}}$ is a strong coreset of $\mathbf{A}$,

$$\|\mathbf{M}_3\mathbf{U}^*\mathbf{V}^*\mathbf{M}_4 - \widetilde{\mathbf{A}}\|_p^p \leq (1+\varepsilon)\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_p^p.$$

Therefore,

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_p^p \leq (1+\varepsilon)^6\|\mathbf{U}^*\mathbf{V}^* - \mathbf{A}\|_p^p$$

and the desired claim then follows from rescaling ε. □

We now analyze the runtime of Algorithm 17.

Lemma 4.19. *For any constant $p \geq 1$, Algorithm 17 uses $2^{\text{poly}(k/\varepsilon)}\,\text{poly}(n,d)$ runtime.*

Proof. By Theorem 4.16, we have that Algorithm 17 uses $2^{\mathcal{O}(k)} \cdot \text{poly}(n,d)$ time to compute $\widetilde{\mathbf{A}} \in \{0,1\}^{N\times D}$ with $N, D = \text{poly}(n)$. We now consider the time to compute $\widetilde{\mathbf{U}^{(i,j)}}, \widetilde{\mathbf{V}^{(i,j)}}$ for each $i,j \in [\ell]$ for $\ell = \mathcal{O}\left(\frac{\log n}{\varepsilon}\right)$. For each i,j, we make guesses for $\mathbf{SU}^*$ and $\mathbf{SA}$ in Since $\mathbf{SU}^*$ and $\mathbf{SA}$ have $m = \mathcal{O}\left(\frac{k^{p+1}\log r}{\varepsilon^2}\right)$ rows, then there are $\binom{t}{m}$ possible choices for $\mathbf{SU}^*$ and $\binom{t}{m}$ choices for $\mathbf{SA}$, where $t = \frac{2^k \log n}{\varepsilon^p}$. Hence, there are $2^{\text{poly}(k/\varepsilon)}\,\text{poly}(n,d)$ possible guesses for $\mathbf{SU}^*$ and $\mathbf{SA}$.

For each guess of $\mathbf{SU}^*$ and $\mathbf{SA}$, Algorithm 16 iterates through the columns of $\widetilde{\mathbf{V}^{(i,j)}}$, which uses $2^{\mathcal{O}(k)} \cdot \text{poly}(n,d)$ time. Similarly, the computation of $\widetilde{\mathbf{U}^{(i,j)}}$, $\mathbf{U}'$, and $\mathbf{V}'$ all take $2^{\mathcal{O}(k)} \cdot \text{poly}(n,d)$ time. Therefore, the total runtime of Algorithm 17 is $2^{\text{poly}(k/\varepsilon)}\,\text{poly}(n,d)$. □

By Lemma 4.18 and Lemma 4.19, we thus have:

Theorem 4.20. *For any constant $p \geq 1$, there exists an algorithm that uses $2^{\text{poly}(k/\varepsilon)}\,\text{poly}(n,d)$ runtime and with probability at least $\frac{2}{3}$, outputs $\mathbf{U}' \in \{0,1\}^{n\times k'}$ and $\mathbf{V}' \in \{0,1\}^{k'\times d}$ such that*

$$\|\mathbf{U}'\mathbf{V}' - \mathbf{A}\|_p^p \leq (1+\varepsilon) \min_{\mathbf{U}\in\{0,1\}^{n\times k}, \mathbf{V}\in\{0,1\}^{k\times d}} \|\mathbf{U}\mathbf{V} - \mathbf{A}\|_p^p,$$

where $k' = \mathcal{O}\left(\frac{k\log^2 k}{\varepsilon^2}\right)$.

We note here that the $\text{poly}(k/\varepsilon)$ term in the exponent hides a k^p factor, as we assume p to be a (small) constant.

4.5 Applications to Big Data Models

This section describes how we can generalize our techniques to big data models such as streaming or distributed models.

Algorithmic modularization. To adapt our algorithm to the streaming model or the distributed model, we first present a high-level modularization of our algorithm across all applications, i.e., Frobenius binary low-rank approximation, binary low-rank approximation over $\mathbb{F}_2$, and binary low-rank approximation with L_p loss. We are given the input matrix $\mathbf{A} \in \{0,1\}^{n\times d}$ in each of these settings. We first construct a weighted coreset $\widetilde{\mathbf{A}}$ for $\mathbf{A}$. We then perform a number of operations on $\widetilde{\mathbf{A}}$ to obtain low-rank factors $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ for $\widetilde{\mathbf{A}}$. Setting $\mathbf{V}' = \widetilde{\mathbf{V}}$, our algorithms finally use $\mathbf{A}$ and $\mathbf{V}'$ to construct the optimal factor $\mathbf{U}'$ to match $\mathbf{V}'$.

4.5.1 Streaming Model

We can adapt our approach to the streaming model, where either the rows or columns of the input matrix arrive sequentially. For brevity, we shall only discuss the setting where the rows of the input matrix arrive sequentially; the setting where the columns of the input matrix arrive sequentially is symmetric.

Formal streaming model definition. We consider the two-pass row-arrival variant of the streaming model. In this setting, the rank parameter k and the accuracy parameter $\varepsilon > 0$ are given to the algorithm before the data stream. The input matrix $\mathbf{A} \in \{0,1\}^{n\times d}$ is then defined through the sequence of row-arrivals, $\mathbf{A}_1, \ldots, \mathbf{A}_n \in \{0,1\}^d$, so that the ith row that arrives in the data stream is $\mathbf{A}_i$. The algorithm passes over the data twice so that in the first pass, it can store some sketch S that uses space sublinear in the input size, i.e., using $o(nd)$ space. After the first pass, the algorithm can perform some post-processing on S and then must output factors $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$ after being given another pass over the data, i.e., the rows $\mathbf{A}_1, \ldots, \mathbf{A}_n \in \{0,1\}^d$.

Two-pass streaming algorithm. To adapt our algorithm to the two-pass streaming model, recall the high-level modularization of our algorithm described at the beginning of Section 4.5. The first step is constructing a coreset $\widetilde{\mathbf{A}}$ of $\mathbf{A}$. Whereas our previous coreset constructions were offline, we now require a streaming algorithm to produce the coreset $\widetilde{\mathbf{A}}$. To that end, we use the following well-known merge-and-reduce paradigm for converting an offline coreset construction to a coreset construction in the streaming model.

Theorem 4.21. *Suppose there exists an algorithm that, with probability* $1 - \frac{1}{\text{poly}(n)}$*, produces an offline coreset construction that uses* $f(n,\varepsilon)$ *space, suppressing dependencies on other input parameters, such as* k *and* p*. Then there exists a one-pass streaming algorithm that, with probability* $1 - \frac{1}{\text{poly}(n)}$*, produces a coreset that uses* $f(n,\varepsilon') \cdot \mathcal{O}(\log n)$ *space, where* $\varepsilon' = \frac{\varepsilon}{\log n}$*.*

In the first pass of the stream, we can use Theorem 4.21 to construct a strong coreset C of $\mathbf{A}$ with accuracy $\mathcal{O}(\varepsilon)$. However, C will have $2^{\text{poly}(k)} \cdot \text{poly}\left(\frac{1}{\varepsilon}, \log n\right)$ rows, and thus, we cannot immediately duplicate the rows of C to form $\widetilde{\mathbf{A}}$ because we cannot have $\log n$ dependencies in the number of rows of $\widetilde{\mathbf{A}}$.

After the first pass of the stream, we further apply the respective offline coreset construction, i.e., Theorem 4.8 or Theorem 4.16 to C to obtain a coreset C' with accuracy ε and a number of rows independent of $\log n$. We then use C' to form $\widetilde{\mathbf{A}}$ and perform a number of operations on $\widetilde{\mathbf{A}}$ to obtain low-rank factors $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ for $\widetilde{\mathbf{A}}$. Setting $\mathbf{V}' = \widetilde{\mathbf{V}}$, we can finally use the second pass of the data stream over $\mathbf{A}$, along with $\mathbf{V}'$, to construct the optimal factor $\mathbf{U}'$ to match $\mathbf{V}'$. Thus the two-pass streaming algorithm uses $2^{\mathrm{poly}(k)} \cdot d \cdot \mathrm{poly}\left(\frac{1}{\varepsilon}, \log n\right)$ total space in the row-arrival model. For the column-arrival model, the two-pass streaming algorithm uses $2^{\mathrm{poly}(k)} \cdot n \cdot \mathrm{poly}\left(\frac{1}{\varepsilon}, \log d\right)$ total space.

4.5.2 Two-round distributed algorithm.

Our approach can also be adapted to the distributed model, where the rows or columns of the input matrix are partitioned across multiple users. For brevity, we again discuss the setting where the rows of the input matrix are partitioned; the setting where the columns of the input matrix are partitioned is symmetric.

Formal distributed model definition. We consider the two-round distributed model, where the rank parameter k and the accuracy parameter $\varepsilon > 0$ are known in advance to all users. The input matrix $\mathbf{A} \in \{0,1\}^{n \times d}$ is then defined arbitrarily through the union of rows, $\mathbf{A}_1, \ldots, \mathbf{A}_n \in \{0,1\}^d$, where each row $\mathbf{A}_i$ may be given to any of γ users. An additional central coordinator sends and receives messages from the users. The protocol is then permitted to use two rounds of communication so that in the first round, the protocol can send $o(nd)$ bits of communication. The coordinator can process the communication to form some sketch S, perform some post-processing on S, and then request additional information from each user, possibly using $o(nd)$ communication to specify the information demanded from each user. After the users again use $o(nd)$ bits of communication in the second round of the protocol, the central coordinator must output factors $\mathbf{U} \in \{0,1\}^{n \times k}$ and $\mathbf{V} \in \{0,1\}^{k \times d}$.

Two-round distributed algorithm. To adapt our algorithm to the two-round distributed model, again recall the high-level modularization of our algorithm described at the beginning of Section 4.5. The first step is constructing a coreset $\widetilde{\mathbf{A}}$ of $\mathbf{A}$. Whereas our previous coreset constructions were offline, we now require a distributed algorithm to produce the coreset $\widetilde{\mathbf{A}}$. To that end, we request that each of the t users send a coreset with accuracy $\mathcal{O}(\varepsilon)$ of their respective rows. Note that each user can construct the coreset locally without requiring any communication since the coreset is only a summary of the rows held by the user. Thus the total communication in the first round is just the offline coreset size times the number of players, i.e., $\gamma \cdot 2^{\mathrm{poly}(k)} \cdot \mathrm{poly}\left(\frac{1}{\varepsilon}, \log n\right)$ rows.

Given the union C of the coresets sent by all users, the central coordinator then constructs a coreset C' of $\mathbf{A}$ with accuracy ε, again using an offline coreset construction. The coordinator then uses C' to form $\widetilde{\mathbf{A}}$ and performs the required operations on $\widetilde{\mathbf{A}}$ to obtain low-rank factors $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{V}}$ for $\widetilde{\mathbf{A}}$.

The coordinator can then send $\mathbf{V}'$ to all players, using $\mathbf{V}'$ and their local subset rows of $\mathbf{A}$ to construct $\mathbf{U}'$ collectively. The users then send the rows of $\mathbf{U}'$ corresponding to the rows of $\mathbf{A}$ local to the user back to the central coordinator, who can then construct $\mathbf{U}'$. Thus the second round of the protocol uses $\tilde{\mathcal{O}}\left(nk+kd\right)\cdot\mathrm{poly}\left(\frac{1}{\varepsilon}\right)$ bits of communication. Hence, the total communication of the protocol is $d\gamma\cdot 2^{\mathrm{poly}(k)}\cdot\mathrm{poly}\left(\frac{1}{\varepsilon},\log n\right)+\tilde{\mathcal{O}}\left(nk+kd\right)\cdot\mathrm{poly}\left(\frac{1}{\varepsilon}\right)$ in the two-round row-partitioned distributed model. For the two-round column-partitioned distributed model, the total communication of the protocol is $n\gamma\cdot 2^{\mathrm{poly}(k)}\cdot\mathrm{poly}\left(\frac{1}{\varepsilon},\log d\right)+\tilde{\mathcal{O}}\left(nk+kd\right)\cdot\mathrm{poly}\left(\frac{1}{\varepsilon}\right)$.

4.6 Experiments

In this section, we aim to evaluate the feasibility of the algorithmic ideas of our algorithm against existing algorithms for binary matrix factorization from previous literature. The running time of our full algorithms for BMF is prohibitively expensive, even for small k, so our algorithm will be based on the ideas of Kumar et al. [202], who only run their algorithms in part, obtaining weaker theoretical guarantees. Indeed, by simply performing k-means clustering, they obtained a simple algorithm that outperformed more sophisticated heuristics in practice.

We perform two main types of experiments, first comparing the algorithm presented in the next section against existing baselines and then showing the feasibility of using coresets in the BMF setting.

Baseline and algorithm. We compare several algorithms for binary matrix factorization that have implementations available online, namely the algorithm by Zhang et al. [335], which has been implemented in the `NIMFA` library [339], the message passing algorithm of Ravanbakhsh et al. [268], as well as our implementation of the algorithm used in the experiments of Kumar et al. [202]. We refer to these algorithms as Zh, MP, and kBMF, respectively. We choose the default parameters provided by the implementations. We chose the maximum number of rounds for the iterative methods so that the runtime does not exceed 20 seconds, as all algorithms besides kBMF [202] are iterative. However, in our experiments, the algorithms usually converged to a solution below the maximum number of rounds. We let every algorithm use the matrix operations over the preferred semiring, i.e., boolean, integer, or and-or matrix multiplication, in order to achieve the best approximation. We additionally found a binary matrix factorization algorithm for sparse matrices based on subgradient descent and random sampling[1] that is not covered in the literature. This algorithm was excluded from our experiments as it did not produce binary factors in our experiments. Specifically, we found that it produces real-valued $\mathbf{U}$ and $\mathbf{V}$, and requires binarizing the product $\mathbf{UV}$ after multiplication, therefore not guaranteeing that the binary matrix is of rank k.

Motivated by the idea of partially executing a more complicated algorithm with strong theoretical guarantees, we build upon the idea of finding a k-means clustering solution

[1] `https://github.com/david-cortes/binmf`

as a first approximation and mapping the Steiner points to their closest neighbors in $\mathbf{A}$, giving us a matrix $\mathbf{V}$ of k binary points, and a matrix $\mathbf{U}$ of assignments of the points of $\mathbf{A}$ to their nearest neighbors. This solution restricts $\mathbf{U}$ to have a single non-zero entry per row. Instead of outputting this $\mathbf{U}$ as Kumar et al. [202] did, we solve the minimization problem $\min_{\mathbf{U}\in\{0,1\}^{n\times k}} \|\mathbf{UV}-\mathbf{A}\|_F^2$ exactly at a cost of 2^k per row, which is affordable for small k. For a qualitative example of how this step improves the solution quality, see Fig. 4.1. We call this algorithm kBMF+.

Using k-means as the first step in a binary matrix factorization algorithm is well-motivated by the theoretical and experimental results of Kumar et al. [202], but does not guarantee a $(1+\varepsilon)$-approximation. However, as we do not run the full algorithm, we are not guaranteed a $(1+\varepsilon)$-approximation either way, as unfortunately, guessing the optimal matrix $\mathbf{V}$ is very time-consuming. We would first have to solve the sketched problem $\|\mathbf{S}\widetilde{\mathbf{A}}-\mathbf{SUV}\|_F^2$ for all guesses of $\mathbf{SA}$ and $\mathbf{SU}$.

We implement our algorithm and the one of Kumar et al. [202] in Python 3.10 and numpy. For solving k-means, we use the implementation of Lloyd's algorithm with k-means++ seeding provided by the `scikit-learn` library [256]. All experiments were performed on a Linux notebook with a 3.9 GHz 12th generation Intel Core i7 six-core processor with 32 gigabytes of RAM.

Datasets. We use both real and synthetic data for our experiments. We choose two datasets from the UCI Machine Learning Repository [116], namely the voting record of the 98th Congress, consisting of 435 rows of 16 binary features representing each congressperson's vote on one of 16 bills, and the Thyroid dataset[2], of 9371 patient data comprising 31 features. We restricted ourselves to only binary features, leaving us with 21 columns. Finally, we use the ORL dataset of faces, which we binarize using a threshold of 0.33, following the methodology of Kumar et al. [202].

For our synthetic data, we generate random matrices, where each entry is set to be 1 independently with probability p, at two different sparsity levels of $p \in \{0.1, 0.5\}$. Additionally, we generate low-rank matrices by generating $\mathbf{U} \in \{0,1\}^{n\times k}$ and $\mathbf{V} \in \{0,1\}^{k\times d}$ and multiplying them together in $\mathbb{F}_2$. We generate $\mathbf{U}$ and $\mathbf{V}$ at different sparsity levels of 0.5 and 0.1, for $k \in \{5, 10, 15\}$. Finally, we also use these matrices with added noise, where after multiplying, each bit is flipped with probability $p_e \in \{0.01, 0.001\}$.

We generate 25 matrices of size 250×50 for each configuration. These classes are named, in order of introduction: full, lr, and noisy.

Limitations. We opted to use only binary datasets, thus limiting the available datasets for our experiments. Because of this, our largest dataset's size is less than 10000. Our algorithms are practical for these sizes and the parameters k we have chosen. Investigating the feasibility of algorithms for binary matrix factorization for large datasets may be an interesting direction for future research.

[2] `https://www.kaggle.com/datasets/emmanuelfwerr/thyroid-disease-data`

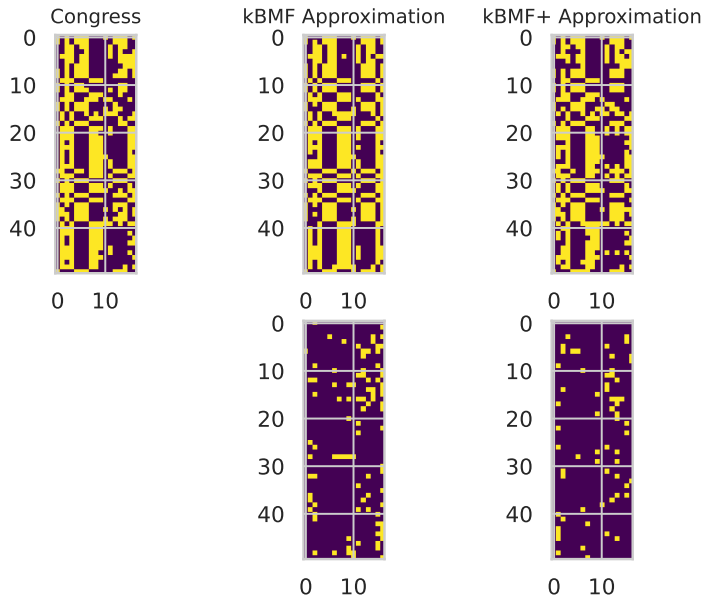

Figure 4.1: A demonstration of the improved approximation of our algorithm over the algorithm used in the experiments of Kumar et al. [202]. In the first column, we show the first 50 rows of the congress dataset, where purple indicates 0 and yellow indicates 1. The next columns show the approximation of Kumar et al. [202], and our algorithm's approximation, both with $k = 10$. The second row indicates the entries in which the respective approximations differ from the original dataset in yellow. Our experiments found that the number of wrongly reconstructed entries almost halved from the kBMF to the kBMF+ algorithm on this dataset for $k = 10$.

4.6.1 Comparing Algorithms for BMF

Synthetic data. For each algorithm, Table 4.2 shows the mean Frobenius norm error (i.e. $\mathrm{err}_{\mathbf{A}}(\mathbf{U}, \mathbf{V}) = \|\mathbf{UV} - \mathbf{A}\|_F$) across 10 runs of each algorithm and the mean runtime in milliseconds for the synthetic datasets described above. For our choices of parameters, we find that all algorithms terminate in under a second, with Zhang's algorithm and BMF being the fastest and the message-passing algorithm generally being the slowest. This is, of course, also influenced by the fact that the algorithms' implementations use different technologies, which limits the conclusions we can draw from the data. We find that the kBMF+ algorithm slows down by a factor of 1.5 for small $k \in \{2, 3, 5\}$, and 15 when $k = 15$, compared to the kBMF algorithm.

This is offset by the improved error, where our algorithm kBMF+ generally achieves the best approximation for dense matrices, being able to sometimes find a perfect factorization, for example, in the case of a rank 5 matrix, when using $k \in \{10, 15\}$. Even when the perfect factorization is not found, we see that the Frobenius norm error is 2-10 times lower. On spare matrices, we find that Zhang's and the message-passing algorithms outperform kBMF+, yielding solutions that are about 2 times better in the worst case (matrix of rank 5, with sparsity 0.1 and $k = 5$). The kBMF algorithm generally performs the worst across datasets, which is surprising considering the results of Kumar et al. [202]. Another point of note is that Zhang's algorithm is tuned for sparse matrices, sometimes converging to factors that yield real-valued matrices. If so, we attempted to round the matrix as best we could.

Real data. As before, Table 4.3 shows the algorithms' average Frobenius norm error and average running time. We observe that all algorithms are fairly close in Frobenius norm error, with the best and worst factorizations' error differing by about up to a factor of 3 across parameters and datasets. Zhang's algorithm performs best on the Congress dataset, while the message-passing algorithm performs best on the ORL and Thyroid datasets. The kBMF algorithm generally does worst, but the additional processing we do in kBMF+ can improve the solution considerably, putting it on par with the other heuristics. On the Congress dataset, kBMF+ is about 1.1-2 times worse than Zhang's, while on the ORL dataset, it is about 10-30% worse than the message-passing algorithm. Finally, the Thyroid dataset's error is about 10-20% worse than competing heuristics.

We note that on the Thyroid datasets, which has almost 10000 rows, Zhang's algorithm slows considerably, about 10 times slower than kBMF and even slower than kBMF+ for $k = 15$. This suggests that for large matrices and small to moderate k, the kBMF+ algorithm may actually run faster than other heuristics while providing comparable results. The message-passing algorithm slows tremendously, being almost three orders of magnitude slower than kBMF, but we believe this could be improved with another implementation.

Discussion. In our experiments, we found that on dense synthetic data, the algorithm kBMF+ outperforms other algorithms for the BMF problem. Additionally, we found that is competitive for sparse synthetic data and real datasets. Curiously, we could

		Error [Frobenius norm]				Time [ms]			
	Alg	kBMF	kBMF+	MP	Zh	kBMF	kBMF+	MP	Zh
Dataset	k								
Random	2	75.8	72.3	**71.3**	**71.3**	11.2	8.6	280.7	11.6
$p = 0.5$	3	74.3	69.9	69.4	**68.7**	14.9	12.5	309.8	11.7
	5	72.2	65.8	66.6	**64.9**	10.9	11.5	347.7	13.3
	10	68.7	**57.4**	61.5	58.5	15.4	53.4	486.6	17.2
	15	66.4	**50.4**	57.9	53.7	16.2	272.1	667.3	21.7
Random	2	36.0	35.0	**34.9**	35.2	10.8	11.3	277.3	9.9
$p = 0.1$	3	35.9	**34.9**	**34.9**	35.0	7.5	13.9	302.1	10.6
	5	35.6	34.6	35.5	**34.2**	12.7	18.5	336.9	12.6
	10	35.0	33.9	35.8	**31.7**	17.0	64.5	459.6	15.9
	15	34.3	33.0	38.5	**29.0**	20.9	269.5	628.4	19.6
Low-Rank	2	72.5	67.1	**66.0**	67.8	4.1	7.9	274.9	11.9
$r = 5$	3	69.2	**60.0**	62.3	64.0	12.8	12.0	301.5	13.5
$p = 0.5$	5	64.0	**26.9**	55.2	56.7	10.4	11.9	339.8	15.4
	10	52.9	**0.7**	41.0	42.5	14.7	72.7	472.5	19.5
	15	43.3	**0.0**	32.8	31.1	18.0	296.0	658.0	23.8
Low-Rank	2	20.5	20.4	16.5	**15.8**	9.4	6.3	185.6	4.8
$r = 5$	3	17.0	16.6	13.1	**12.0**	5.0	5.8	209.1	12.3
$p = 0.1$	5	11.1	8.4	**4.6**	5.1	7.0	8.0	275.9	14.8
	10	5.1	**0.0**	0.7	2.3	19.3	75.0	460.5	18.1
	15	1.5	**0.0**	0.4	1.4	20.2	297.0	630.9	22.1
Low-Rank	2	75.8	72.2	**71.1**	71.7	13.4	15.5	281.2	11.5
$r = 10$	3	74.3	69.6	69.1	**69.0**	15.8	20.0	308.0	11.7
$p = 0.5$	5	72.0	**64.7**	66.1	64.8	20.9	19.7	345.5	13.6
	10	68.2	**28.4**	60.2	57.9	16.2	51.4	477.8	17.3
	15	65.6	**0.8**	56.0	52.9	19.3	245.2	659.6	21.3
Low-Rank	2	30.8	30.5	**27.6**	28.5	10.0	14.3	213.4	5.7
$r = 10$	3	28.5	28.1	**25.2**	25.5	11.1	13.3	248.5	11.5
$p = 0.1$	5	24.7	23.2	20.4	**19.9**	13.1	18.7	292.0	13.4
	10	18.3	10.2	**7.6**	8.8	16.4	76.2	434.6	16.9
	15	15.2	**2.5**	4.7	5.4	14.8	261.3	638.8	22.1
Low-Rank	2	75.7	72.3	**71.2**	71.3	14.5	18.6	277.6	11.3
$r = 15$	3	74.2	69.9	69.3	**68.7**	12.7	11.1	306.5	11.7
$p = 0.5$	5	72.1	65.7	66.6	**64.8**	15.0	19.0	339.7	13.0
	10	68.6	**56.5**	61.5	58.4	18.7	51.4	478.3	17.2
	15	66.4	**29.2**	57.7	53.6	13.0	239.9	652.8	21.1
Low-Rank	2	38.7	38.2	**35.6**	36.5	12.1	10.4	242.2	9.7
$r = 15$	3	37.1	36.2	**33.7**	34.2	10.0	13.0	274.1	12.8
$p = 0.1$	5	33.7	32.2	29.8	**29.5**	13.2	17.9	313.2	14.6
	10	28.1	22.3	20.3	**19.8**	20.2	56.3	457.3	17.9
	15	25.3	14.2	**11.6**	13.4	21.2	247.9	643.8	21.2
Noisy	2	75.8	72.3	**71.2**	71.6	13.9	12.8	290.4	11.3
$r = 10$	3	74.3	69.6	69.3	**69.0**	13.8	15.6	309.3	11.6
$p = 0.5$	5	72.1	**64.7**	66.2	65.0	17.6	23.8	345.8	13.6
$p_{noise} = 0.001$	10	68.2	**33.8**	60.3	58.1	16.8	54.0	481.1	17.6
	15	65.6	**4.8**	56.2	53.2	18.4	247.1	661.8	21.6
Noisy	2	32.5	32.1	**29.3**	30.0	6.3	9.6	223.6	7.6
$r = 10$	3	30.0	29.5	**26.9**	27.1	6.4	10.1	255.4	11.6
$p = 0.1$	5	26.2	24.6	22.0	**21.3**	6.6	9.7	291.9	13.5
$p_{noise} = 0.001$	10	19.8	12.0	**9.3**	10.4	16.4	67.4	441.2	18.2
	15	16.7	**4.9**	6.8	7.2	13.9	255.0	641.8	22.4
Noisy	2	75.8	72.1	**71.0**	71.7	9.7	11.4	276.1	11.4
$r = 10$	3	74.3	69.5	**69.0**	69.1	12.1	13.3	302.4	12.0
$p = 0.5$	5	72.0	**64.7**	66.0	64.8	12.4	12.5	338.9	13.4
$p_{noise} = 0.01$	10	68.3	**38.2**	60.2	57.9	15.0	50.7	475.0	17.2
	15	65.7	**16.7**	56.1	52.8	18.0	254.0	672.9	21.3
Noisy	2	33.3	33.0	**30.3**	30.9	9.9	11.5	225.3	9.2
$r = 10$	3	31.3	30.8	28.2	**28.0**	10.8	10.5	257.5	12.5
$p = 0.1$	5	27.8	26.2	23.6	**23.4**	9.4	18.3	292.1	14.3
$p_{noise} = 0.01$	10	22.3	16.3	**14.0**	15.1	21.0	58.5	448.5	17.4
	15	19.9	12.5	12.5	**12.0**	20.5	260.3	645.4	21.7

Table 4.2: The average running time and error for different Binary Matrix Factorization algorithms on synthetic datasets. The minimum Frobenius norm error is marked in bold.

Dataset	Alg / k	Error [Frobenius norm] kBMF	kBMF+	MP	Zh	Time [ms] kBMF	kBMF+	MP	Zh
Congress	2	40.0	38.8	38.8	**36.4**	2.0	3.3	280.7	6.9
	3	38.4	36.6	35.9	**32.7**	2.3	4.1	311.2	13.6
	5	35.7	32.7	31.1	**27.7**	4.6	5.2	332.9	16.2
	10	32.7	23.9	22.5	**18.4**	3.2	16.9	407.1	22.6
	15	30.9	14.8	15.5	**9.6**	7.4	246.7	480.5	27.5
ORL	2	39.4	37.8	35.9	**33.5**	2.0	2.9	203.7	11.6
	3	35.7	34.6	32.2	**29.7**	2.9	4.7	241.6	13.1
	5	31.7	30.7	27.7	**25.6**	3.8	5.8	289.4	15.4
	10	26.4	25.7	21.6	**21.4**	4.3	22.3	415.7	19.1
	15	23.4	22.8	**17.8**	19.7	6.1	318.0	575.5	22.2
Thyroid	2	106.6	98.6	**90.5**	91.6	12.6	14.2	7063.6	44.3
	3	94.5	90.5	75.5	**73.9**	14.4	18.7	7822.0	92.9
	5	82.7	80.4	78.5	**61.8**	31.8	25.2	8860.2	132.1
	10	66.0	55.4	54.0	**52.9**	28.9	59.6	12686.3	241.4
	15	57.6	**38.9**	39.2	46.7	26.7	313.4	16237.7	432.7

Table 4.3: The average running time and error for different Binary Matrix Factorization algorithms on real datasets, minimum frobenius norm error highlighted in bold.

not replicate the results of [202], which show that kBMF outperforms the algorithm of Zhang et al. [335]. We believe this appears to be due to the fact that the open source version of Zhang's algorithm used in the experiments of [202] is implemented incorrectly. One inherent benefit of the kBMF and kBMF+ algorithms is that they are very easily adapted to different norms and matrix products, as the clustering step, nearest neighbor search, and enumeration steps are all easily adapted to the setting we want. A benefit is that the factors are guaranteed to be either 0 or 1, which is not true for Zhang's heuristic, which does not always converge. None of the existing heuristics consider minimization of L_p norms, so we omitted experimental data for this setting, but we note here that the results are qualitatively similar, with our algorithm performing best on dense matrices, and the heuristics performing well on sparse data.

4.6.2 Using Coresets with our Algorithm

Motivated by our theoretical use of strong coresets for k-means clustering, we perform experiments to evaluate the increase in error using them. To this end, we run the BMF+ algorithm on either the entire dataset, a coreset constructed via importance sampling [28, 52], or a lightweight coreset [29]. Both of these algorithms were implemented in Python. The datasets in this experiment are a synthetic low-rank dataset with additional noise (size 5000×50, rank 5 and 0.0005 probability of flipping a bit), the congress, and thyroid datasets.

We construct coresets of size rn for each $r \in \{0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, \ldots, 0.9\}$.

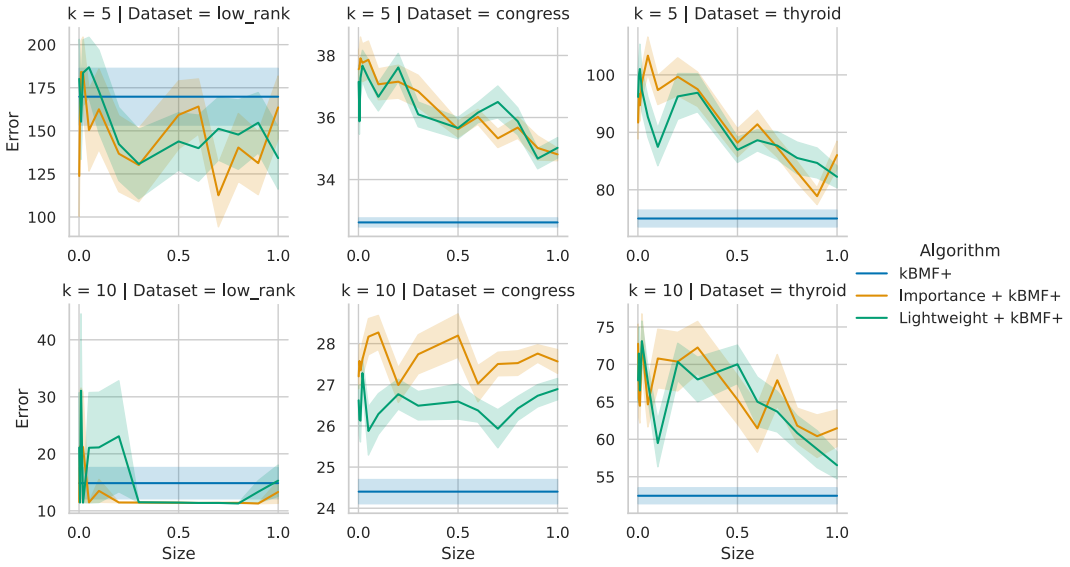

Figure 4.2: A plot of the effect of different relative coreset sizes on the results of our algorithm.

We sample 10 coresets at every size and use them when finding $\mathbf{V}$ in our BMF+ algorithm. Theory suggests that the quality of the coreset depends only on k and the dimension of the points d, which is why in Fig. 4.2, we observe a worse approximation for a given size of coreset for larger k. We find that the BMF+ algorithm performs just as well on lightweight coresets as the one utilizing the sensitivity sampling framework. This is expected in the binary setting, as the additive error in the weaker guarantee provided by lightweight coresets depends on the dataset's diameter. Thus, the faster, lightweight coreset construction appears superior in this setting.

We observe that using coreset increases the Frobenius norm error we observe by about 35%, but curiously, on the low-rank dataset, the average error decreased after using coresets. This may be due to coreset constructions not sampling the noisy outliers that are not in the low-dimensional subspace spanned by the non-noisy low-rank matrix, letting the algorithm better reconstruct the original factors instead.

Our datasets are comparatively small, none exceeding 1000 points, which is why, in combination with the fact that the coreset constructions are not optimized, we observe no speedup compared to the algorithm without coresets. However, even though constructing the coreset takes additional time, the running time between variants remained comparable. We expect to observe significant speedups for large datasets using an optimized implementation of the coreset algorithms. Using *off the shelf* coresets provides a large advantage to this algorithm's feasibility compared to the iterative methods when handling large datasets.

4.7 Conclusion

In this paper, we introduced the first $(1+\varepsilon)$-approximation algorithms for binary matrix factorization with a singly exponential dependence on the low-rank factor k, which is often a small parameter. We consider optimization with respect to the Frobenius loss, finite fields, and L_p loss. Our algorithms extend naturally to big data models and perform well in practice. Indeed, we conduct empirical evaluations demonstrating the practical effectiveness of our algorithms. For future research, we leave open the question for $(1+\varepsilon)$-approximation algorithms for L_p loss without bicriteria requirements.

5 Simple, Scalable and Effective Clustering via One-Dimensional Projections

Clustering is a fundamental problem in unsupervised machine learning with many applications in data analysis. Popular clustering algorithms such as Lloyd's algorithm and k-means++ can take $\Omega(ndk)$ time when clustering n points in a d-dimensional space (represented by an $n \times d$ matrix $\mathbf{X}$) into k clusters. In applications with moderate to large k, the multiplicative k factor can become very expensive. We introduce a simple randomized clustering algorithm that provably runs in expected time $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log n)$ for arbitrary k. Here $\mathsf{nnz}(\mathbf{X})$ is the total number of non-zero entries in the input dataset $\mathbf{X}$, which is upper bounded by nd and can be significantly smaller for sparse datasets. We prove that our algorithm achieves approximation ratio $\tilde{\mathcal{O}}(k^4)$ on any input dataset for the k-means objective. We also believe that our theoretical analysis is of independent interest, as we show that the approximation ratio of a k-means algorithm is approximately preserved under a class of projections and that k-means++ seeding can be implemented in expected $\mathcal{O}(n \log n)$ time in one dimension. Finally, we show experimentally that our clustering algorithm gives a new trade-off between running time and cluster quality compared to previous state-of-the-art methods for these tasks.

5.1 Introduction

Clustering is an essential and powerful tool for data analysis with broad applications in computer vision and computational biology, and it is one of the fundamental problems in unsupervised machine learning. In large-scale applications, datasets often contain billions of high-dimensional points. Grouping similar data points into clusters is crucial for understanding and organizing datasets. Because of its practical importance, the problem of designing efficient and effective clustering algorithms has attracted the attention of numerous researchers for many decades.

One of the most popular algorithms for the k-means clustering problem is Lloyd's algorithm [212], which seeks to locate k centers in the space that minimize the sum of squared distances from the points of the dataset to their closest center (we call this the "k-means cost"). While finding the centers minimizing the objective is NP-hard [15], in practice we can find high-quality sets of centers using Lloyd's iterative algorithm. Lloyd's algorithm maintains a set of k centers. It iteratively updates them by assigning points to one of k clusters (according to their closest center), then redefining the center

as the points' center of mass. It needs a good initial set of centers to obtain a high-quality clustering and fast convergence. In practice, the k-means++ algorithm [20], a randomized *seeding* procedure, is used to choose the initial k centers. k-means++ achieves an $\mathcal{O}(\log k)$-approximation ratio in expectation, upon which each iteration of Lloyd's algorithm improves.[1] Beyond their effectiveness, these algorithms are simple to describe and implement, contributing to their popularity.

The downside of these algorithms is that they do not scale well to massive datasets. A standard implementation of an iteration of Lloyd's algorithm needs to calculate the distance to each center for each point in the dataset, leading to a $\Theta(ndk)$ running time. Similarly, the standard implementation of the k-means++ seeding procedure produces k samples from the so-called D^2 distribution (see Section 5.3 for details). Maintaining the distribution requires making a pass over the entire dataset after choosing each sample. Generating k centers leads to a $\Theta(ndk)$ running time. Even for moderate values of k, making k passes over the entire dataset can be prohibitively expensive.

One particularly relevant application of large-scale k-means clustering is in approximate nearest neighbor search [292] (for example, in product quantization [169] and building inverted file indices [54]). There, k-means clustering is used to compress entire datasets by mapping vectors to their nearest centers, leading to billion-scale clustering problems with large k (on the order of hundreds or thousands). Other applications on large datasets requiring a large number of centers may be spam filtering [264, 286], near-duplicate detection [154], and compression or reconciliation tasks [271]. New algorithmic ideas are needed for these massive scales, and this motivates the following challenge:

> Can we design a simple, practical algorithm for k-means that runs in time roughly $\mathcal{O}(nd)$, independent of k, and produces high-quality clusters?

Given its importance in theory and practice, a significant amount of effort has been devoted to algorithms for fast k-means clustering. We summarize a few of the approaches below with the pros and cons of each so that we may highlight our work's position within the literature:

A. **Standard k-means++**: This is our standard benchmark. **Plus:** Guaranteed to be an $\mathcal{O}(\log k)$-approximation [20]; outputs centers, as well as the assignments of dataset points to centers. **Minus:** The running time is $\mathcal{O}(ndk)$, which is prohibitively expensive in large-scale applications.

B. **Using Approximate Nearest Neighbor Search**: One may implement k-means++ faster using techniques from approximate nearest neighbor search (instead of a brute force search each iteration). **Plus:** The algorithms with provable guarantees, like [90], obtain an $\mathcal{O}_\varepsilon(\log k)$-approximation. **Minus:** The running time is $\tilde{\mathcal{O}}\left(nd + (n\log(\Delta))^{1+\varepsilon}\right)$, depending on a dataset dependent parameter Δ, the ratio between the maximum and minimum distances between input points. The

[1] Approximation is with respect to the k-means cost. A c-approximation has k-means cost, which is at most c times larger than the optimal k-means cost.

techniques are algorithmically sophisticated and incur extra poly-logarithmic factors (hidden in $\tilde{\mathcal{O}}(\cdot)$), making the implementation significantly more complicated.

C. **Approximating the D^2-Distribution**: Algorithms that speed up the seeding procedure for Lloyd's algorithm or generate fast coresets (we expand on this below) have been proposed in [27, 26, 29]. **Plus:** These algorithms are fast, making only one pass over the dataset in time $\mathcal{O}(nd)$. (For [27, 26], there is an additional additive $\mathcal{O}(k^2 d)$ term in the running time). **Minus:** The approximation guarantees are qualitatively weaker than the approximation of k-means clustering. They incur an additional additive approximation error that grows with the entire dataset's variance (which can lead to an arbitrarily large error; see Section 5.7). These algorithms output a set of k centers but not the cluster assignments. Naively producing the assignments would take time $\mathcal{O}(ndk)$.[2]

Coresets. At a high level, coresets are a dataset-reduction mechanism. A large dataset $\mathbf{X}$ of n points in $\mathbb{R}^d$ is distilled into a significantly smaller (weighted) dataset $\mathbf{Y}$ of m points in $\mathbb{R}^d$, called a "coreset" which serves as a good proxy for $\mathbf{X}$, i.e., the clustering cost of any k centers on $\mathbf{Y}$ is approximately the cost of the same centers on $\mathbf{X}$. We point the reader to [28, 122] for a recent survey on coresets. Importantly, coreset constructions (with provable multiplicative-approximation guarantees) require an initial approximate clustering of the original dataset $\mathbf{X}$. Therefore, any fast algorithm for k-means clustering automatically speeds up any algorithmic pipeline that uses coresets for clustering — looking forward, we will show how our algorithm can significantly speed up coreset constructions without sacrificing approximation.

Beyond those mentioned above, many works seek to speed up k-means++ or Lloyd iterations by maintaining some nearest neighbor search data structures [258, 234, 181, 180, 119, 262, 147, 261, 317, 109, 108, 49, 243, 95, 61], or by running some first-order methods [284]. These techniques do not give provable guarantees on the quality of the k-means clustering or on the running time of their algorithms.

Theoretical Results. We give a simple randomized clustering algorithm with provable guarantees on its running time and approximation ratio without making any assumptions about the data. It has the benefit of being fast (like the algorithms in Category C above) while achieving a multiplicative error guarantee without additional additive error (like the algorithms in Category B above).

- The algorithm runs in time $\mathcal{O}(nd + n \log n)$ irrespective of k. It passes over the dataset once to perform data reduction, which gives the nd factor plus an additive $\mathcal{O}(n \log n)$ term to solve k-means on the reduced data, producing k centers and cluster assignments. On sparse input datasets, the nd term becomes $\mathsf{nnz}(\mathbf{X})$, where

[2] One may use approximate nearest neighbor search techniques to improve on the $\mathcal{O}(ndk)$ running time. However, as discussed above, approximate nearest neighbor search adds a significant layer of complexity (and approximation).

$\mathsf{nnz}(\mathbf{X})$ is the number of non-zero entries in the dataset. Thus, our algorithm runs in $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log n)$ time on sparse matrices.

- The algorithm is as simple as the k-means++ algorithm while significantly more efficient. The approximation ratio we prove is $\text{poly}(k)$, which is worse than the $\mathcal{O}(\log k)$-approximation achieved by k-means++ but multiplicative (see the remark below on improving this to $\mathcal{O}(\log k)$). It does not incur the additional additive errors from the fast algorithms in [27, 26, 29].

Our algorithm projects the input points to a random one-dimensional space and runs an efficient k-means++ seeding after the projection. For the approximation guarantee, we analyze how the approximation ratio achieved after the projection can be transferred to the original points (Lemma 5.5). We bound the running time of our algorithm by efficiently implementing the k-means++ seeding in one dimension and analyzing the running time via a potential function argument (Lemma 5.4). Our algorithm applies beyond k-means to other clustering objectives that sum up the z-th power of the distances for general $z \geq 1$, and our guarantees on its running time and approximation ratio extend smoothly to these settings.

Improving the Approximation from $\text{poly}(k)$ to $\mathcal{O}(\log k)$. The approximation ratio of $\text{poly}(k)$ may seem significantly worse than the $\mathcal{O}(\log k)$ approximations achievable with k-means++. However, we can improve this to $\mathcal{O}(\log k)$ with an additional, additive $\mathcal{O}(\text{poly}(kd) \cdot \log n)$ term in the running time. Using previous results discussed in Section 5.3.2 (specifically Theorem 5.11), a multiplicative $\text{poly}(k)$-approximation suffices to construct a coreset of size $\text{poly}(kd)$ and run k-means++ on the coreset. Constructing the coreset is simple and takes time $\text{poly}(kd) \cdot \log n$ (by sampling from an appropriate distribution); running k-means++ on the coreset takes $\text{poly}(kd)$ time (with no dependence on n). Combining our algorithm with coresets, we get a $\mathcal{O}(\log k)$-approximation in $\mathcal{O}(\mathsf{nnz}(\mathbf{X})) + \mathcal{O}(n \log n) + \text{poly}(kd) \cdot \log n$ time. Notably, these guarantees cannot be achieved with the additive approximations of [27, 26, 29].

Experimental Results. We implemented our algorithm, as well as the lightweight coreset of Bachem et al. [29] and k-means++ with sensitivity sampling [51]. We ran two types of experiments, highlighting various aspects of our algorithm. Our code is published on GitHub[3]. The two types of experiments are:

- **Coreset Construction Comparison**: First, we evaluate the performance of our clustering algorithm when we use it to construct coresets. We compare the performance of our algorithm to k-means++ with sensitivity sampling [28] and lightweight coresets [29]. In real-world, high-dimensional data, the cost of the resulting clusters from the three algorithms is roughly the same. However, ours and the lightweight coresets can be significantly faster (ours is up to **190x** faster than k-means++, see Fig. 5.4 and Table 5.1). The lightweight coresets can be faster

[3] `PRONE` GitHub repository: `https://github.com/boredoms/prone`

than our algorithm (between 3-5x); however, our algorithm is "robust" (achieving multiplicative approximation guarantees).[4] Additionally, we show that the clustering from lightweight coresets can have an arbitrarily high cost for a synthetic dataset. On the other hand, our algorithm achieves provable (multiplicative) approximation guarantees irrespective of the dataset (this is demonstrated in the right-most column of Fig. 5.4).

- **Direct k-means++ comparison**: Second, we compare the speed and cost of our algorithm to k-means++[20] as a stand-alone clustering algorithm (we also compare two other natural variants of our algorithm). Our algorithm can be up to **800x** faster than k-means++ for $k = 5000$ and our slowest variant up to **100x** faster (Table 5.1). The cost of the cluster assignments can be significantly worse than that of k-means++ (see Fig. 5.5). Such a result is expected since our theoretical results show a poly(k)-approximation. The other (similarly) fast algorithms (based on approximating the D^2-distribution) which run in time $\mathcal{O}(nd)$ [27, 26] do not produce the cluster assignments (they only output k centers). These algorithms would take $\mathcal{O}(ndk)$ time to find the cluster assignments — this is precisely the computational cost our algorithm avoids.

We do not compare our algorithm with [90] nor implement approximate nearest neighbor search to speed up k-means++ for the following reasons. The algorithm in [90] is significantly more complicated, and there is no publicly available implementation. In addition, both [90] and approximate nearest neighbor search incur additional poly-logarithmic (or even $n^{o(1)}$-factors for nearest neighbor search over ℓ_2 [17]) which add significant layers of complexity to the implementation and make a thorough evaluation of the algorithm significantly more complicated. Instead, our current implementation demonstrates that a simple, one-dimensional projection and k-means++ on the line enables dramatic speedups to coreset constructions without sacrificing approximation quality.

Related Work. Efficient algorithms for clustering problems with provable approximation guarantees have been studied extensively, with a few approaches in the literature. There are polynomial-time (constant) approximation algorithms (an exponential dependence on k is not allowed) (see [209, 57, 10, 141] for some of the most recent and strongest results), nearly linear time $(1 \pm \varepsilon)$-approximations with running time exponential in k which proceed via coresets (see [151, 71, 123, 124, 51, 28, 92, 89] and references therein, as well as the surveys [7, 122]), and nearly-linear time $(1 \pm \varepsilon)$-approximations in fixed / low-dimensional spaces [19, 197, 305, 131, 88, 85, 86]. Our $\mathcal{O}(n \log n)$-expected-time implementation of k-means++ seeding achieves an $\mathcal{O}(\log k)$ expected approximation ratio for k-median and k-means in one dimension. We are unaware of previous work on clustering algorithms running in time $\mathcal{O}(n \log n)$.

Another line of research has been on dimensionality reduction techniques for k-means clustering. Dimensionality reduction can be achieved via PCA based methods [111, 124,

[4] Recall that the lightweight coresets incur an additional additive error which can be arbitrarily large.

[84, 296], or random projection [84, 39, 219]. For random projection methods, it has been shown that the k-means objective is preserved up to small multiplicative factors when projecting onto $\mathcal{O}_\varepsilon(\log(k))$ dimensional space. Additional work has shown that dimensionality reduction can be performed in $\mathcal{O}(\mathsf{nnz}(A))$ time [211]. To the best of our knowledge, we are the first to show that clustering objectives such as k-median and k-means are preserved up to a poly(k) factor by one-dimensional projections.

Some works show that the $\mathcal{O}(\log k)$ expected approximation ratio for k-means++ can be improved by adding local search steps after the seeding procedure [205, 80]. In particular, Choo et al. [80] showed that adding εk local search steps achieves an $\mathcal{O}\left(1/\varepsilon^3\right)$ approximation ratio with high probability.

Several other algorithmic approaches exist for fast clustering of points in metric spaces. These include density-based methods like DBSCAN [120] and DBSCAN++ [168] and the line of heuristics based on the Partitioning Around Medoids (PAM) approach, such as FastPAM [282], Clarans [246], and BanditPAM [309]. While these algorithms can produce high-quality clustering, their running time is at least linear in the number of clusters (DBSCAN++ and BanditPAM) or superlinear in the number of points (DBSCAN, FastPAM, Clarans).

5.2 Overview of Our Algorithm and Proof Techniques

Our algorithm, which we call `PRONE` (PRojected ONE-dimensional clustering), takes a random projection onto a one-dimensional space, sorts the projected (scalar) numbers, and runs the k-means++ seeding strategy on the projected numbers. By virtue of its simplicity, the algorithm is scalable and effective at clustering massive datasets. More formally, `PRONE` receives as input a dataset of n points in $\mathbb{R}^d$, a parameter $k \in \mathbb{N}$ (the number of desired clusters), and proceeds as follows:

1. Sample a random vector $\mathbf{v} \in \mathbb{R}^d$ from the standard Gaussian distribution and project the data points to one dimension along the direction of $\mathbf{v}$. That is, we compute $\mathbf{x}'_i = \langle \mathbf{x}_i, \mathbf{v} \rangle \in \mathbb{R}$ in time $\mathcal{O}(\mathsf{nnz}(\mathbf{X}))$ by making a single pass over the data, effectively reducing our dataset to the collection of one-dimensional points $\mathbf{x}'_1, \ldots, \mathbf{x}'_n \in \mathbb{R}$.

2. Run k-means++ seeding on $\mathbf{x}'_1, \ldots, \mathbf{x}'_n$ to obtain k indices $j_1, \ldots, j_k \in [n]$ indicating the chosen centers $\mathbf{x}'_{j_1}, \ldots, \mathbf{x}'_{j_k}$ and an assignment $\sigma : [n] \to [k]$ assigning point $\mathbf{x}'_i$ to center $\mathbf{x}'_{j_{\sigma(i)}}$. Even though k-means++ seeding generally takes $\mathcal{O}(nk)$ time in one dimension, we give an efficient implementation, leveraging the fact that points are one-dimensional, which runs in $\mathcal{O}(n \log n)$ expected time, independent of k. A detailed algorithm description is in Section 5.5.

3. The one-dimensional k-means++ algorithm produces a collection of k centers $\mathbf{x}_{j_1}, \ldots, \mathbf{x}_{j_k}$, as well as the assignment σ mapping each point $\mathbf{x}_i$ to the center $\mathbf{x}_{j_{\sigma(i)}}$. For each $\ell \in [k]$, we update the cluster center for cluster ℓ to be the center of mass of all points assigned to $\mathbf{x}_{j_\ell}$.

While the algorithm is straightforward, the main technical difficulty lies in the analysis. In particular, our analysis

1. bounds the approximation loss incurred from the one-dimensional projection in Step 1, and

2. shows that we can implement Step 2 in $\mathcal{O}(n \log n)$ expected time, as opposed to $\mathcal{O}(nk)$ time.

We summarize the theoretical contributions in the following theorems.

Theorem 5.1. *The algorithm* PRONE *has expected running time* $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log n)$ *on any dataset* $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$. *Moreover, for any* $\delta \in (0, 1/2)$ *and any dataset* $\mathbf{X}$, *with probability at least* $1 - \delta$, *the algorithm runs in time* $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log(n/\delta))$.

Theorem 5.2. *The algorithm* PRONE *achieves an* $\tilde{\mathcal{O}}(k^4)$ *approximation ratio for the* k*-means objective with probability at least* 0.9.

To our knowledge, PRONE is the first algorithm for k-means running in time $\mathcal{O}(nd + n \log n)$ for arbitrary k. As mentioned in the paragraph on improving the competitive ratio, we obtain the following corollary of Theorems 5.1 and 5.2 using a two-stage approach with a coreset:

Corollary 5.3. *By using* PRONE *as the* α*-approximation algorithm in Theorem 5.11 and running* k*-means++ on the resulting coreset, we obtain an algorithm with an approximation ratio of* $\mathcal{O}(\log k)$ *that runs in time* $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log n + \mathrm{poly}(kd) \log n)$, *with constant success probability.*

The proofs of Theorems 5.1 and 5.2 can be found in Sections 5.4 and 5.5, where we also generalize them beyond k-means to clustering objectives that sum up the z-th power of Euclidean distances for general $z \geq 1$. The following subsections give a high-level overview of the main techniques we develop to prove our main theorems above.

5.2.1 Efficient Seeding in One Dimension

The k-means++ seeding procedure has k iterations, where a new center is sampled in each iteration. Since a new center may need to update $\Omega(n)$ distances to maintain the D^2 distribution, which samples each point with probability proportional to its distance to its closest center, a naive analysis leads to a running time of $\mathcal{O}(nk)$. A key ingredient in the proof of Theorem 5.1 is showing that, for one-dimensional datasets, k-means++ only needs to make $\mathcal{O}(n \log n)$ updates, irrespective of k.

Lemma 5.4. *The* k*-means++ seeding procedure can be implemented in expected time* $\mathcal{O}(n \log n)$ *in one dimension. Moreover, for any* $\delta \in (0, 1/2)$, *with probability at least* $1 - \delta$, *the implementation runs in time* $\mathcal{O}(n \log(n/\delta))$.

The intuition of the proof is as follows: Since points are one-dimensional, we always maintain them in sorted order. In addition, each data point $\mathbf{x}'_i$ will maintain its center assignment and distance p_i to the closest center. By building a binary tree over the sorted points (where internal nodes maintain sums of p_i^2's), it is easy to sample a new center from the D^2 distribution in $\mathcal{O}(\log n)$ time. The difficulty is that adding a new center may result in changes to p_i's of multiple points $\mathbf{x}'_i$, so the challenge is to bound the number of times these values are updated (see Fig. 5.1 below).

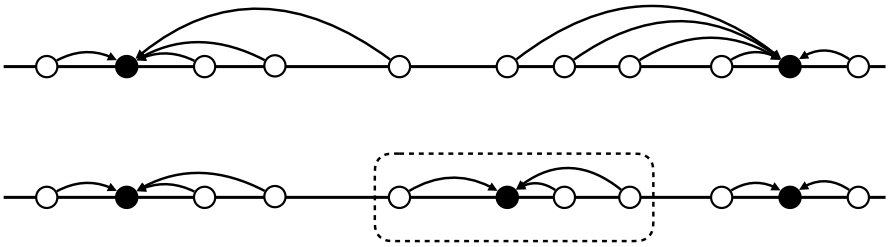

Figure 5.1: From the top to the bottom, a new center (black circle) is chosen. Every point has an arrow pointing to its closest center. The points in the dashed box are the ones that require updates.

To bound the total running time, we leverage the one-dimensional structure. Observe that, for a new center, the updated points lie in a contiguous interval around the newly chosen center. Once a center is chosen, the algorithm scans the points (to the left and the right) until we reach a point that does not need to be updated. This point identifies that points to the other side of it need not be updated, so we can get away without necessarily checking all n points (see Fig. 5.1). Somewhat surprisingly, when sampling centers from the D^2-distribution, the expected number of times that each point will be updated is only $\mathcal{O}(\log n)$, which implies a bound of $\mathcal{O}(n \log n)$ on the total number of updates in expectation. The analysis of the fact that each point is updated $\mathcal{O}(\log n)$ times is non-trivial and uses a carefully designed potential function (Lemma 5.24).

5.2.2 Approximation Guarantees from One-Dimensional Projections

Our proof of Theorem 5.2 builds on a line of work studying randomized dimension reduction for clustering problems [50, 84, 39, 219]. Prior work studied randomized dimension reduction for accurate $(1 \pm \varepsilon)$-approximations. Our perspective is slightly different; we restrict ourselves to one-dimensional projections and give an upper bound on the distortion.

For any dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, a projection to a random lower-dimensional space affects the pairwise distance between the projected points in a predictable manner — the Johnson-Lindenstrauss lemma which projects to $\mathcal{O}(\log n)$ dimensions being a prime example of this fact. When projecting to just one dimension, however, pairwise distances will be significantly affected (by up to poly(n)-factors). Thus, a naive analysis will give a poly(n)-approximation for k-means. To improve a c-approximation to a $\mathcal{O}(\log k)$-approximation, one needs a coreset of size roughly poly$(c/\log k)$. This bound becomes

vacuous when c is polynomial in n since there are at most n dataset points.

However, although many pairwise distances are significantly distorted, we show that the k-means cost is only affected by a poly(k)-factor. At a high level, this occurs because the k-means cost optimizes a sum of pairwise distances (according to a chosen clustering). The individual summands, given by pairwise distances, will change significantly, but the overall sum does not. Our proof follows the approach of Cohen et al. [84], which showed that (roughly speaking) pairwise distortion of the k optimal centers suffices to argue about the k-means cost. The k optimal centers will incur maximal pairwise distortion poly(k) when projected to one dimension (because there are only $\mathcal{O}\left(k^2\right)$ pairwise distances among the k centers). This allows us to lift an r-approximate solution after the projection to an $\mathcal{O}\left(k^4 r\right)$-approximate solution for the original points.

Lemma 5.5 (Informal)**.** *For any set $\mathbf{X}$ of points in $\mathbb{R}^d$, the following occurs with probability at least 0.9 over the choice of a standard Gaussian vector $\mathbf{v} \in \mathbb{R}^d$. Letting $\mathbf{X}' \subset \mathbb{R}$ be the one-dimensional projection of $\mathbf{X}$ onto $\mathbf{v}$, any r-approximate k-means clustering of $\mathbf{X}'$ gives an $\mathcal{O}\left(k^4 r\right)$-approximate clustering of $\mathbf{X}$ with the same clustering partition.*

5.3 Preliminaries

In this work, we always consider datasets $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$ of high-dimensional vectors, and we will measure their distance using the Euclidean (L_2) distance. Below, we define (k, z)-clustering. This problem introduces a parameter $z \geq 1$, which measures the *sensivity to outliers* (as z grows, the clusterings become more sensitive to points furthest from the cluster center). The case of k-means corresponds to $z = 2$, but other values of z capture other well-known clustering objectives, like k-median (the case of $z = 1$).

Definition 5.6 ((k, z)-Clustering)**.** *Consider a dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$, a desired number of clusters $k \in \mathbb{N}$, and a parameter $z \geq 1$. For a set of k centers $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$, let $\operatorname{cost}_z(\mathbf{X}, C)$ denote the cost of using the center set C to cluster $\mathbf{X}$, i.e.,*

$$\operatorname*{cost}_z(\mathbf{X}, C) = \sum_{i=1}^{n} \min_{j \in [k]} \|\mathbf{x}_i - \mathbf{c}_j\|_2^z.$$

We let $\mathsf{Opt}_{k,z}(X)$ denote the optimal cost over all choices of $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$:

$$\mathsf{Opt}_{k,z}(\mathbf{X}) = \inf_{\substack{C \subset \mathbb{R}^d \\ |C| \leq k}} \operatorname*{cost}_z(\mathbf{X}, C).$$

A (k, z)-clustering algorithm has the following specifications. The algorithm receives as input a dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$, as well as two parameters $k \in \mathbb{N}$ and $z \geq 1$. After it executes, the algorithm should output a set of k centers $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$ as well as an assignment $\sigma : [n] \to [k]$ mapping each point $\mathbf{x}_i$ to a center $\mathbf{c}_{\sigma(i)}$.

We measure the quality of the solution (C, σ) using the ratio between its (k, z)-clustering cost $\mathrm{cost}_z(\mathbf{X}, C, \sigma)$ and the optimal cost $\mathsf{Opt}_{k,z}(\mathbf{X})$, where

$$\underset{z}{\mathrm{cost}}(\mathbf{X}, C, \sigma) := \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{c}_{\sigma(x_i)}\|_2^z. \tag{5.1}$$

For any $\mathsf{D} > 1$, an algorithm that produces a D-approximation to (k, z)-clustering should guarantee that $\mathrm{cost}_z(\mathbf{X}, C, \sigma)$ is at most $\mathsf{D} \cdot \mathsf{Opt}_{k,z}(\mathbf{X})$. For a randomized algorithm, the guarantee should hold with large probability (referred to as the success probability) for any input dataset.

5.3.1 k-Means++ Seeding

The k-means++ seeding algorithm is a well-studied algorithm introduced by Arthur and Vassilvitskii [20], and it is an important component of our algorithm. Below, we describe it for general $z \geq 1$, not necessarily $z = 2$.

Definition 5.7 (k-means++ seeding, for arbitrary $z \geq 1$)**.** *Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, the k-means++ seeding algorithm produces a set of k centers, $\mathbf{x}_{\ell_1}, \ldots, \mathbf{x}_{\ell_k}$ with the following procedure:*

1. *Choose ℓ_1 uniformly at random from $[n]$.*
2. *For $t = 2, \ldots, k$, sample ℓ_t as follows. For every $i \in [n]$, let p_i denote the Euclidean distance from $\mathbf{x}_i$ to its closest point among $\mathbf{x}_{\ell_1}, \ldots, \mathbf{x}_{\ell_{t-1}}$. Sample ℓ_t from $[n]$ so that the probability $\mathbf{Pr}[\ell_t = i]$ is proportional to p_i^z for every $i \in [n]$. That is,*

$$\mathbf{Pr}[\ell_t = i] = \frac{p_i^z}{\sum_{i' \in [n]} p_i^z}.$$

   In the context of k-means (i.e., when $z = 2$), the distribution is known as the D^2 distribution.

3. *Output $\mathbf{x}_{\ell_1}, \ldots, \mathbf{x}_{\ell_k}$.*

In the above description, Step 2 of k-means++ needs to maintain, for each dataset point $\mathbf{x}_i$, the Euclidean distance p_i to its closest center among the centers selected before the current iteration. This step is implemented by making an entire pass over the dataset for each of the $k - 1$ iterations of Step 2, leading to an $\mathcal{O}(ndk)$ running time.

Theorem 5.8 ([20], Theorem 3 in [313])**.** *For $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ be the input to the k-means++ seeding algorithm. For the output $\mathbf{x}_{\ell_1}, \ldots, \mathbf{x}_{\ell_k}$ of the k-means++ seeding algorithm, define $C = \{\mathbf{x}_{\ell_1}, \ldots, \mathbf{x}_{\ell_k}\}$. Then*

$$\mathbb{E}\left[\underset{z}{\mathrm{cost}}(\mathbf{X}, C)\right] = \mathcal{O}\left(2^{2z} \log k\right) \cdot \mathsf{Opt}_{k,z}(\mathbf{X}).$$

5.3.2 Coresets via Sensitivity Sampling

One of our algorithm's applications is constructing coresets for (k, z)-clustering. We give a formal definition and describe the primary technique for building coresets.

Definition 5.9. *Given a dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$, as well as parameters $k \in \mathbb{N}$, $z \geq 1$ and $\varepsilon > 0$, a (strong) ε-coreset for (k, z)-clustering is specified by a set of points $\mathbf{Y} \subset \mathbb{R}^d$ and a weight function $w \colon \mathbf{Y} \to \mathbb{R}_{\geq 0}$, such that, for every set $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$,*

$$(1-\varepsilon) \cdot \operatorname*{cost}_z(\mathbf{X}, C) \leq \sum_{\mathbf{y} \in \mathbf{Y}} w(\mathbf{y}) \cdot \min_{j \in [k]} \|\mathbf{y} - \mathbf{c}_j\|_2^z \leq (1+\varepsilon) \cdot \operatorname*{cost}_z(\mathbf{X}, C).$$

Coresets are constructed via "sensitivity sampling," a technique that, given an approximate clustering of a dataset $\mathbf{X}$, produces a probability distribution such that sampling enough points from this distribution results in a coreset.

Definition 5.10 (Sensitivity Sampling)**.** *Consider a dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$, as well as parameters $k \in \mathbb{N}$, $z \geq 1$. For a center set $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$ and assignment $\sigma \colon [n] \to [k]$, let $\mathbf{X}_j = \{\mathbf{x}_i : \sigma(i) = j\}$. We let $\mathcal{D}$ be a distribution supported on $\mathbf{X}$ where*

$$\Pr_{\mathbf{x} \sim \mathcal{D}}[\mathbf{x} = \mathbf{x}_i] \propto \frac{\|\mathbf{x}_i - \mathbf{c}_{\sigma(i)}\|_2^z}{\sum_{j=1}^n \|\mathbf{x}_j - \mathbf{c}_{\sigma(j)}\|_2^z} + \frac{1}{|\mathbf{X}_{\sigma(i)}|}.$$

The main theorem that we will use is given below, which shows that given a center set and an assignment that gives an α-approximation to (k, z)-clustering, one may sample from the distribution $\mathcal{D}$ defined about in order to generate a coreset with high probability.

Theorem 5.11 ([51])**.** *For any dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$ and any parameters $k \in \mathbb{N}$ and $z \geq 1$, suppose that $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$ and $\sigma \colon [n] \to [k]$ is a α-approximation to (k, z)-clustering, i.e.,*

$$\sum_{i=1}^n \|\mathbf{x}_i - \mathbf{c}_{\sigma(i)}\|_2^z \leq \alpha \, \mathsf{Opt}_{k,z}(\mathbf{X}).$$

Letting $\mathcal{D}$ denote the distribution specified in Definition 5.10, the following occurs with high probability.

- *We let $\mathbf{y}_1, \ldots, \mathbf{y}_s$ denote independent samples from $\mathcal{D}$, and $w(\mathbf{y}_i)$ be the inverse of the probability that $\mathbf{y}_i$ is sampled according to $\mathcal{D}$. We set $s \geq \operatorname{poly}(kd \cdot \alpha \cdot 2^z/\varepsilon)$.*
- *The set $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_s\}$ with weights w is an ε-coreset for (k, z)-clustering.*

5.3.3 A Simple Lemma

We will repeatedly use the following simple lemma.

Lemma 5.12. *Let $a, b \in \mathbb{R}_{\geq 0}$ be any two numbers and $z \geq 1$. Then, $(a+b)^z \leq 2^{z-1}a^z + 2^{z-1}b^z$.*

Proof. The function $\phi(t) = t^z$ is convex for $z \geq 1$, so Jensen's inequality implies $\phi((a+b)/2) \leq (1/2)\phi(a) + (1/2)\phi(b)$. □

5.4 Approximation Guarantees from One-Dimensional Projections

In this section, we prove Theorem 5.2 (rather, the generalization of Theorem 5.2 to any $z \geq 1$) by analyzing the random one-dimensional projection step in our algorithm. In order to introduce some notation, let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$ be a set of points, and for a partition of $\mathbf{X}$ into k sets, $(Y_1, \ldots, Y_k)$, we let the (k, z)-*clustering cost of* $\mathbf{X}$ *with the partition* $(Y_1, \ldots, Y_k)$ be

$$\underset{z}{\text{cost}}(Y_1, \ldots, Y_k) = \sum_{i=1}^{k} \min_{\mathbf{c} \in \mathbb{R}^d} \sum_{\mathbf{x} \in Y_i} \|\mathbf{x} - \mathbf{c}\|_2^z \tag{5.2}$$

and call the k points c selected as minima a set of centers realizing the (k, z)-clustering cost of $(Y_1, \ldots, Y_k)$. We note that (Eq. (5.2)) is a cost function for (k, z)-clustering, but it is different from Definition 5.6. In Definition 5.6, the emphasis is on the set of k centers $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$, and the induced set of clustering of $\mathbf{X}$, i.e., the partition $(Y_1, \ldots, Y_k)$ given by assigning points to the closest center, is only implicitly specified by the set of centers. On the other hand, Eq. (5.2) emphasizes the clustering $(Y_1, \ldots, Y_k)$, and the set of k centers implicitly specified by $(Y_1, \ldots, Y_k)$. The optimal set of centers and the optimal clustering will achieve the same cost; however, our proof will mostly consider the clustering $(Y_1, \ldots, Y_k)$ as the object to optimize. Shortly, we will sample a (random) dimensionality reduction map $\mathbf{\Pi} \colon \mathbb{R}^d \to \mathbb{R}^t$ and seek bounds for $t = 1$. We will write $\text{cost}_z(\mathbf{\Pi}(Y_1), \ldots, \mathbf{\Pi}(Y_k))$ for the cost of clustering the points after applying the dimensionality reduction map $\mathbf{\Pi}$ to the partition $Y_1, \ldots, Y_k$. Namely, we write

$$\underset{z}{\text{cost}}(\mathbf{\Pi}(Y_1), \ldots, \mathbf{\Pi}(Y_k)) = \sum_{i=1}^{k} \min_{\mathbf{c} \in \mathbb{R}^t} \sum_{\mathbf{x} \in Y_i} \|\mathbf{\Pi}(\mathbf{x}) - \mathbf{c}\|_2^z.$$

Definition 5.13. *For a set of points* $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$*, we use* $X_1^*, \ldots, X_k^*$ *of* $\mathbf{X}$ *to denote the partition of* $\mathbf{X}$ *with minimum* (k, z)*-clustering cost and we use* $C^* = \{\mathbf{c}_1^*, \ldots, \mathbf{c}_k^*\} \subset \mathbb{R}^d$ *to denote a set of* k *centers which realizes the* (k, z)*-clustering cost of* $X_1^*, \ldots, X_k^*$*, i.e., the set of centers which satisfies*

$$\underset{z}{\text{cost}}(X_1^*, \ldots, X_k^*) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in X_i^*} \|\mathbf{x} - \mathbf{c}_i^*\|_2^z.$$

By slight abuse of notation, we also let $c^* \colon \mathbf{X} \to C^*$ *be the map which sends every point of* $\mathbf{X}$ *to its corresponding center (i.e., if* $\mathbf{x} \in X_i^*$*, then* $c^*(\mathbf{x})$ *is the point* $\mathbf{c}_i^*$*).*

We prove the following lemma, which generalizes Lemma 5.5 from k-means to (k, z)-clustering (recall that k-means corresponds to the case of $z = 2$).

Lemma 5.14 (Effect of One-Dimensional Projection on (k, z)-Clustering)**.** *For $n, d, k \in \mathbb{N}$ and $z \geq 1$, let $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n) \subset \mathbb{R}^d$ be an arbitrary dataset. We consider the (random) linear map $\mathbf{\Pi}\colon \mathbb{R}^d \to \mathbb{R}$ given by sampling $\mathbf{g} \sim \mathcal{N}(0, I_d)$ and setting*

$$\mathbf{\Pi}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{g} \rangle .$$

With probability at least 0.9 over $\mathbf{g}$, the following occurs:

- *We consider the projected dataset $\mathbf{X}' = (\mathbf{x}'_1, \dots, \mathbf{x}'_n) \subset \mathbb{R}$ be given by $\mathbf{x}'_i = \mathbf{\Pi}(\mathbf{x}_i)$, and*
- *For any $r \geq 1$, we let $(Y_1, \dots Y_k)$ denote any partition of $\mathbf{X}$ satisfying*

$$\operatorname*{cost}_z(\mathbf{\Pi}(Y_1), \dots, \mathbf{\Pi}(Y_k)) \leq r \cdot \min_{\mathbf{c}_1, \dots, \mathbf{c}_k \in \mathbb{R}} \sum_{i=1}^{n} \min_{j \in [k]} |\mathbf{x}'_i - \mathbf{c}_j|^z.$$

Then,

$$\operatorname*{cost}_z(Y_1, \dots, Y_k) \leq 2^{\mathcal{O}(z)} \cdot k^{2z} \cdot r \cdot \mathsf{Opt}_{k,z}(\mathbf{X}).$$

By setting $z = 2$, we obtain the desired bound from Lemma 5.5. We can immediately see that, from Lemma 5.14, and the approximation guarantees of k-means++ (or rather, its generalization to $z \geq 1$) in Theorem 5.8, we obtain our desired approximation guarantees. Below, we state the generalization of Theorem 5.2 to all $z \geq 1$ and, assuming Lemma 5.14, its proof.

Theorem 5.15 (Generalization of Theorem 5.2 to $z \geq 1$)**.** *For $n, d, k \in \mathbb{N}$ and $z \geq 1$, let $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n) \subset \mathbb{R}^d$ be an arbitrary dataset. We consider the following generalization of our algorithm* `PRONE`*:*

1. *Sample a random Gaussian vector $\mathbf{g} \sim \mathcal{N}(0, I_d)$ and consider the projection $\mathbf{X}' = (\mathbf{x}'_1, \dots, \mathbf{x}'_n)$ given by $\mathbf{x}'_i = \mathbf{\Pi}(\mathbf{x}_i)$, for $\mathbf{\Pi}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{g} \rangle \in \mathbb{R}$.*
2. *Execute the (generalization of the) k-means++ seeding strategy for $z \geq 1$ of Definition 5.7 with the dataset $\mathbf{X}' \subset \mathbb{R}$, and let $\mathbf{x}'_{j_1}, \dots, \mathbf{x}'_{j_k} \in \mathbb{R}$ denote the centers and $(Y_1, \dots, Y_k)$ denote the partition of $\mathbf{X}$ specifying the k clusters found.*
3. *Output the clustering $(Y_1, \dots, Y_k)$, and the set of centers $\mathbf{c}_1, \dots, \mathbf{c}_k \in \mathbb{R}^d$ where*

$$\mathbf{c}_\ell = \mathop{\mathbb{E}}_{\mathbf{x} \sim Y_\ell} [\mathbf{x}] \in \mathbb{R}^d.$$

Then, with probability at least 0.8 over the execution of the algorithm,

$$\operatorname*{cost}_z(Y_1, \dots, Y_k) \leq \sum_{\ell=1}^{k} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \mathbf{c}_\ell\|_2^z \leq 2^{\mathcal{O}(z)} \cdot k^{2z} \cdot \log k \cdot \mathsf{Opt}_{k,z}(\mathbf{X}).$$

Proof of Theorem 5.15 assuming Lemma 5.14. We consider the case (over the randomness in the execution of the algorithm) that:

1. The conclusions of Lemma 5.14 hold for the projected dataset $\mathbf{X}'$ (which happens with probability at least 0.9) by Lemma 5.14.
2. The execution of the generalization k-means++ seeding strategy on $\mathbf{X}'$ (from Definition 5.7) produces a set of centers $\{\mathbf{x}'_{j_1}, \dots, \mathbf{x}'_{j_k}\} \subset \mathbb{R}$ which cluster $\mathbf{X}'$ with cost at most $\mathcal{O}\left(2^{2z} \log k\right) \cdot \mathsf{Opt}_{k,z}(\mathbf{X}')$ (which also happens with probability 0.9 by Markov's inequality).

By a union bound, both hold with probability at least 0.8. We now use Lemma 5.14 to upper bound the cost of the clustering $(Y_1, \dots, Y_k)$. The first inequality is trivial; suppose we let $\hat{\mathbf{c}}^*_1, \dots, \hat{\mathbf{c}}^*_\ell \in \mathbb{R}^d$ be the centers which minimize for each $\ell \in [k]$

$$\min_{\hat{\mathbf{c}}_\ell \in \mathbb{R}^d} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \hat{\mathbf{c}}_\ell\|_2^z = \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \hat{\mathbf{c}}^*_\ell\|_2^z.$$

Then, we trivially have

$$\operatorname*{cost}_z(Y_1, \dots, Y_k) = \sum_{\ell=1}^{k} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \hat{\mathbf{c}}^*_\ell\|_2^z \le \sum_{\ell=1}^{k} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \mathbf{c}_\ell\|_2^z.$$

Furthermore, we can also show a corresponding upper bound. For each $\ell \in [k]$, recall that $\mathbf{c}_\ell \in \mathbb{R}^d$ is the center of mass of Y_k, so we can apply the triangle inequality and Lemma 5.12

$$\begin{aligned}\sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \mathbf{c}_\ell\|_2^z &\le 2^{z-1} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \hat{\mathbf{c}}^*_\ell\|_2^z + 2^{z-1} |Y_\ell| \cdot \|\hat{\mathbf{c}}^*_\ell - \mathop{\mathbb{E}}_{\mathbf{x} \sim Y_\ell}[\mathbf{x}]\|_2^z \\ &\le 2^{z-1} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \hat{\mathbf{c}}^*_\ell\|_2^z + 2^{z-1} |Y_\ell| \cdot \mathop{\mathbb{E}}_{\mathbf{x} \sim Y_\ell}\left[\|\mathbf{x} - \hat{\mathbf{c}}^*_\ell\|_2^z\right],\end{aligned}$$

where the second inequality is Jensen's inequality, since $\phi(\mathbf{x}) = \|\hat{\mathbf{c}}^*_\ell - \mathbf{x}\|_2^z$ is convex for $z \ge 1$. Thus, we have upper-bounded

$$\sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \mathbf{c}_\ell\|_2^z \le 2^z \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \hat{\mathbf{c}}^*_\ell\|_2^z,$$

and therefore

$$\sum_{\ell=1}^{k} \sum_{\mathbf{x} \in Y_\ell} \|\mathbf{x} - \mathbf{c}_\ell\|_2^z \le 2^z \cdot \operatorname*{cost}_z(Y_1, \dots, Y_k). \tag{5.3}$$

The final step involves relating $\operatorname{cost}_z(Y_1, \dots, Y_k)$ using the conclusions of Lemma 5.14. Notice that our algorithm produces the clustering $(Y_1, \dots, Y_k)$ of $\mathbf{X}'$ which is specified by letting

$$Y_\ell = \left\{\mathbf{x}_i \in \mathbf{X} : \forall j' \in [k], |\mathbf{x}'_i - \mathbf{x}'_{j_\ell}| \le |\mathbf{x}'_i - \mathbf{x}_{j'}|^z\right\},$$

and by the event (2), we have $\operatorname{cost}_z(\mathbf{\Pi}(Y_1), \dots, \mathbf{\Pi}(Y_k)) \le \mathcal{O}\left(2^{2z} \log k\right) \cdot \mathsf{Opt}_{k,z}(\mathbf{X}')$. By event (1), Lemma 5.14 implies that $\operatorname{cost}_z(Y_1, \dots, Y_k) \le 2^{\mathcal{O}(z)} \cdot k^{2z} \cdot \mathcal{O}\left(2^{2z} \log k\right) \cdot \mathsf{Opt}_{k,z}(\mathbf{X})$. Combined with Eq. (5.3), we obtain our desired bound. □

5.4.1 Proof of Lemma 5.14

We now turn to the proof of Lemma 5.14, where our analysis will proceed in two steps. First, we assume a fixed dimensionality reduction map $\Pi\colon \mathbb{R}^d \to \mathbb{R}^t$, which satisfies two geometrical conditions on Π. Under these conditions, we show how to "lift" an approximate clustering of the mapped points in $\mathbb{R}^t$ to an approximate clustering of the original dataset in $\mathbb{R}^d$ at the cost of weakening the approximation ratio. Then, we show that a simple one-dimensional projection $\mathbf{\Pi}\colon \mathbb{R}^d \to \mathbb{R}$ given by $\mathbf{\Pi}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{g}\rangle$, for $\mathbf{g}$ being sampled from a d-dimensional standard Gaussian, satisfies the geometrical conditions of our lemma.

Lemma 5.16. *Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$ and $\Pi\colon \mathbb{R}^d \to \mathbb{R}^t$ be a linear map. Let $C = \{\mathbf{c}_1^*, \ldots, \mathbf{c}_k^*\} \subset \mathbb{R}^d$ denote the set of centers minimizing $\operatorname{cost}_z(\mathbf{X}, C)$, and $(X_1^*, \ldots, X_k^*)$ denote the optimal (k,z)-clustering, and suppose that for the parameters $\mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3 \geq 1$, the following conditions hold:*

- **Centers Don't Contract**: *Every $i, j \in [k]$ satisfies*
$$\|\mathbf{c}_i^* - \mathbf{c}_j^*\|_2 \leq \mathsf{D}_1 \cdot \|\Pi(\mathbf{c}_i^*) - \Pi(\mathbf{c}_j^*)\|_2.$$
- **Cost of $(X_1^*, \ldots, X_k^*)$ does not Increase**: *We have that*
$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in X_i^*} \|\Pi(\mathbf{x}) - \Pi(\mathbf{c}_i^*)\|_2^z \leq \mathsf{D}_2 \cdot \operatorname*{cost}_z(X_1^*, \ldots, X_k^*).$$
- **Approximately Optimal $(\Pi(Y_1), \ldots, \Pi(Y_k))$**: *The partition $(Y_1, \ldots, Y_k)$ of $\mathbf{X}$ is D_3 -approximately optimal for $\Pi(\mathbf{X})$, i.e.,*
$$\operatorname*{cost}_z(\Pi(Y_1), \ldots, \Pi(Y_k)) \leq \mathsf{D}_3 \cdot \min_{\mathbf{c}_1, \ldots, \mathbf{c}_k \in \mathbb{R}^t} \sum_{\mathbf{x} \in \mathbf{X}} \min_{j \in [k]} \|\Pi(\mathbf{x}) - \mathbf{c}_j\|_2^z.$$

Then,
$$\operatorname*{cost}_z(Y_1, \ldots, Y_k) \leq \left(2^{z-1} + 2^{3z-2}\mathsf{D}_1^z \mathsf{D}_2(1 + \mathsf{D}_3)\right) \cdot \operatorname*{cost}_z(X_1^*, \ldots, X_k^*).$$

Before starting the proof of Lemma 5.16, we show that projecting points onto a random Gaussian vector gives the first two desired guarantees of the above lemma with $\mathsf{D}_1 := (k^2/\delta)$ and $\mathsf{D}_2 := 2^{\mathcal{O}(z)}/\delta$ with probability at least $1 - \delta$. The first lemma that we state below shows that the first condition of Lemma 5.16 is satisfied with high probability, and the second lemma that the second condition of Lemma 5.16 is satisfied with high probability.

Lemma 5.17 (Centers Don't Contract)**.** *Let $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\} \subset \mathbb{R}^d$ denote any collection of k points and let $\mathbf{\Pi}\colon \mathbb{R}^d \to \mathbb{R}$ be a random map given by*
$$\mathbf{\Pi}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{g}\rangle$$
for a randomly chosen vector $\mathbf{g} \sim \mathcal{N}(0, I_d)$. Then, with probability at least $1 - \delta$ over $\mathbf{g}$, every $i, j \in [k]$ satisfies
$$\left(\frac{\delta}{k^2}\right) \cdot \|\mathbf{c}_i - \mathbf{c}_j\|_2 \leq \|\mathbf{\Pi}(\mathbf{c}_i) - \mathbf{\Pi}(\mathbf{c}_j)\|_2.$$

Lemma 5.18 (Cost of $(X_1^*, \ldots, X_k^*)$ does not Increase)**.** *Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^d$ and let $X_1^*, \ldots, X_k^*$ be the partition of $\mathbf{X}$, and $\mathbf{c}_1^*, \ldots, \mathbf{c}_k^* \in \mathbb{R}^d$ be the centers which minimize the (k, z)-clustering cost of $\mathbf{X}$. Then, with probability a least $1 - \delta$,*

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathbf{X}_i^*} \|\mathbf{\Pi}(\mathbf{x}) - \mathbf{\Pi}(\mathbf{c}_i^*)\|_2^z \leq \left(2^{\mathcal{O}(z)}/\delta\right) \sum_{i=1}^{k} \sum_{\mathbf{x} \in X_i^*} \|\mathbf{x} - \mathbf{c}_i^*\|_2^z.$$

Proof of Lemma 5.14 assuming Lemma 5.16, Lemma 5.17 and Lemma 5.18. We will apply Lemma 5.16 by letting δ be a small enough constant (say, $\delta = 0.01$) to take a union bound. Lemma 5.17 implies the first condition with $\mathsf{D}_1 = \mathcal{O}\left(k^2\right)$ and Lemma 5.18 implies the second condition with $\mathsf{D}_2 = 2^{\mathcal{O}(z)}$. Finally, the second assumption of Lemma 5.14 sets $r = \mathsf{D}_3$, from which we derive the conclusion. □

We now prove Lemma 5.17, Lemma 5.18. Lemma 5.16 is proved in Section 5.4.2.

Proof of Lemma 5.17. The proof relies on the 2-stability property of the Gaussian distribution. Namely, if we let $\mathbf{z} \in \mathbb{R}^d$ be an arbitrary vector and we sample a standard Gaussian vector $\mathbf{g} \sim \mathcal{N}(0, I_d)$, the (scalar) random variable $\langle \mathbf{z}, \mathbf{g} \rangle$ is distributed like $\|\mathbf{z}\|_2 \cdot g'$, where $g' \sim \mathcal{N}(0, 1)$. Using the 2-stability of the Gaussian distribution for every $i, j \in [k]$, we have that $\|\mathbf{\Pi}(\mathbf{c}_i) - \mathbf{\Pi}(\mathbf{c}_j)\|_2^2$ is distributed as $(g')^2 \|\mathbf{c}_i - \mathbf{c}_j\|_2^2$, where g' is distributed as a (one-dimensional) Gaussian $\mathcal{N}(0, 1)$. Thus, by a union bound, the probability that there exists a pair $i, j \in [k]$, which satisfies $\|\mathbf{\Pi}(\mathbf{c}_i) - \mathbf{\Pi}(\mathbf{c}_j)\|_2^2 < \alpha^2 \cdot \|\mathbf{c}_i - \mathbf{c}_j\|_2^2$ is at most k^2 times the probability that a Gaussian random variable lies in $[-\alpha, \alpha]$, and this probability is easily seen to be less than α. Setting $\alpha = \delta/k^2$ gives the desired lemma. □

Proof of Lemma 5.18. Similarly to the proof of Lemma 5.17, we have that $\|\mathbf{\Pi}(\mathbf{x}) - \mathbf{\Pi}(\mathbf{c}_i^*)\|_2^z$ is distributed as $|g'|^z \cdot \|\mathbf{x} - \mathbf{c}_i^*\|_2^z$, where g' is distributed as a (one-dimensional) Gaussian $\mathcal{N}(0, 1)$. By linearity of expectation,

$$\mathop{\mathbb{E}}_{\mathbf{\Pi} \sim \mathcal{J}_d} \left[\sum_{i=1}^{k} \sum_{\mathbf{x} \in X_i^*} \|\mathbf{\Pi}(\mathbf{x}) - \mathbf{\Pi}(\mathbf{c}_i^*)\|_2^z \right] = \sum_{i=1}^{k} \sum_{\mathbf{x} \in X_i^*} \mathop{\mathbb{E}}_{g' \sim \mathcal{N}(0,1)} \left[|g_i'|^z \right] \cdot \|\mathbf{x} - \mathbf{c}_i^*\|_2^z.$$

To conclude, note that for $z \geq 1$, there is some $\alpha > 1$ such that αz is an even integer and $\alpha \leq 2$. Thus, by Jensen's inequality and the fact that $f(\mathbf{x}) = \mathbf{x}^{1/\alpha}$ is concave we can write

$$\mathop{\mathbb{E}}_{g' \sim \mathcal{N}(0,1)} \left[|g'|^z \right] \leq \left(\mathbb{E} \left[(g')^{\alpha z} \right] \right)^{1/\alpha}$$

Now note that all odd moments of the Gaussian distribution are zero by symmetry. Thus, for the moment generating function $\mathbb{E}\left[e^{g'}\right]$ it holds that

$$\mathbb{E}\left[e^{g'}\right] = \sum_{k=0}^{\infty} \frac{1}{(2k)!} \cdot \mathbb{E}\left[(g')^{2k}\right].$$

As $\mathbb{E}\left[e^{g'}\right] \leq e^{1/2}$ it follows that

$$\left(\mathbb{E}\left[(g')^{\alpha z}\right]\right)^{1/\alpha} \leq \left((\alpha z)!\mathbb{E}\left[e^{g'}\right]\right)^{1/\alpha} \leq \left((\alpha z)!e^{1/2}\right)^{1/\alpha} \leq 2^{\mathcal{O}(z)}.$$

Applying Markov's inequality now completes the proof. □

5.4.2 Proof of Lemma 5.16

Let $\{\hat{\mathbf{c}}_i\}_{i\in[k]}$ be an optimal set of centers for the partition $(Y_1, \ldots, Y_k)$ for the (k, z)-clustering problem on $\Pi(\mathbf{X})$, where $\hat{\mathbf{c}}_i \in Y_i$. Specifically, the points $\hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_k \in \mathbb{R}^t$ are those which minimize

$$\text{cost}(\Pi(Y_1), \ldots, \Pi(Y_k)) \stackrel{\text{def}}{=} \sum_{i=1}^{k} \sum_{\mathbf{x}\in Y_i} \|\Pi(\mathbf{x}) - \hat{\mathbf{c}}_i\|_2^z.$$

To quantize the cost difference between the centers $\mathbf{c}^*$ and the centers $\hat{\mathbf{c}}$ we analyze the following value. We assume we mapped every point of $\mathbf{X}$ to $\Pi(c^*(\mathbf{X}))$, and we then compute the cost of the partition $Y_1, \ldots, Y_k$ on this set. Formally, we let

$$\text{val}(\Pi, c^*, Y_1, \ldots, Y_k) = \sum_{i=1}^{k} \sum_{j=1}^{k} |Y_i \cap X_j^*| \cdot \|\Pi(\mathbf{c}_j^*) - \hat{\mathbf{c}}_i\|_2^z.$$

First, we prove the following simple claim.

Claim 5.19. *There exists a set of centers $\mathbf{c}'_1, \ldots, \mathbf{c}'_k$ (with possible repetitions) which are chosen among the points $\{\mathbf{c}_1^*, \ldots, \mathbf{c}_k^*\}$ such that*

$$\sum_{i=1}^{k} \sum_{j=1}^{k} |Y_i \cap X_j^*| \cdot \|\Pi(\mathbf{c}_j^*) - \Pi(\mathbf{c}'_i)\|_2^z \leq 2^z \cdot \text{val}(\Pi, c^*, Y_1, \ldots, Y_k).$$

Proof. We will prove the claim using the probabilistic method. For every $i \in [k]$, consider the distribution over center $\{\mathbf{c}_1^*, \ldots, \mathbf{c}_k^*\}$ which samples a center $\mathbf{c}'_i$ as

$$\Pr_{\mathbf{c}'_i}\left[\mathbf{c}'_i = \mathbf{c}_j^*\right] = \frac{|Y_i \cap X_j^*|}{\sum_{\ell=1}^{d} |Y_i \cap X_j^*|}.$$

Then, we upper bound the expected cost of using the centers $\mathbf{c}'_i$. Using Lemma 5.12,

$$\begin{aligned}
&\mathbb{E}\left[\sum_{i=1}^{k}\sum_{j=1}^{k}|Y_i \cap X_j^*| \cdot \|\Pi(\mathbf{c}_j^*) - \Pi(\mathbf{c}'_i)\|_2^z\right] \\
&\leq \sum_{i=1}^{k}\sum_{j=1}^{k}|Y_i \cap X_j^*| \cdot \mathbb{E}\left[\left(\|\Pi(\mathbf{c}_j^*) - \hat{\mathbf{c}}_i\|_2 + \|\Pi(\mathbf{c}'_i) - \hat{\mathbf{c}}_i\|_2\right)^z\right] \\
&\leq 2^{z-1}\sum_{i=1}^{k}\sum_{j=1}^{k}|Y_i \cap X_j^*| \cdot \|\Pi(\mathbf{c}_j^*) - \hat{\mathbf{c}}_i\|_2^z + 2^{z-1}\sum_{i=1}^{k}\left(\sum_{j=1}^{k}|Y_i \cap X_j^*|\right)\mathbb{E}\left[\|\Pi(\mathbf{c}'_i) - \hat{\mathbf{c}}_i\|_2^z\right] \\
&= 2^{z-1} \cdot \mathrm{val}(\Pi, c^*, Y_1, \ldots, Y_k) + 2^{z-1}\sum_{i=1}^{k}\left(\sum_{j=1}^{k}|Y_i \cap X_j^*|\right)\sum_{\ell=1}^{k}\frac{|Y_i \cap X_\ell^*|}{\sum_{j=1}^{k}|Y_i \cap X_j^*|} \cdot \|\Pi(\mathbf{c}_\ell^*) - \hat{\mathbf{c}}_i\|_2^z \\
&= 2^z \cdot \mathrm{val}(\Pi, c^*, Y_1, \ldots, Y_k). \qquad \square
\end{aligned}$$

We now upper bound $\mathrm{cost}_z(Y_1, \ldots, Y_k)$ in terms of $\mathrm{cost}_z(X_1^*, \ldots, X_k^*)$. We do this by going through the centers chosen according to Claim 5.19. This will allow us to upper bound the cost of clustering with $(Y_1, \ldots, Y_k)$ in terms of the $\mathrm{cost}_z(X_1^*, \ldots, X_k^*)$ as well as clustering cost involving only pairwise distances from $\{\mathbf{c}_1^*, \ldots, \mathbf{c}_k^*\}$. Then, we relate to distances after applying the map Π. Specifically, first notice that if we consider the set of centers $\{\mathbf{c}'_1, \ldots, \mathbf{c}'_k\}$ chosen from Claim 5.19

$$\begin{aligned}
\underset{z}{\mathrm{cost}}(Y_1, \ldots, Y_n) &\leq \sum_{i=1}^{k}\sum_{\mathbf{x}\in Y_i}\|\mathbf{x} - \mathbf{c}'_i\|_2^z \\
&\leq \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{\mathbf{x}\in Y_i\cap X_j^*}\left(\|\mathbf{x} - \mathbf{c}_j^*\|_2 + \|\mathbf{c}_j^* - \mathbf{c}'_i\|_2\right)^z \\
&\leq 2^{z-1} \cdot \underset{z}{\mathrm{cost}}(X_1^*, \ldots, X_k^*) + 2^{z-1}\sum_{i=1}^{k}\sum_{j=1}^{k}|Y_i \cap X_j^*| \cdot \|\mathbf{c}_j^* - \mathbf{c}'_i\|_2^z, \qquad (5.4)
\end{aligned}$$

where the third inequality uses Lemma 5.12 once more. Note that the right-most summation of Eq. (5.4) involves distances which are only among $\mathbf{c}_1^*, \ldots, \mathbf{c}_k^*$, so by the first assumption of the map Π and Claim 5.19, we may upper bound

$$\begin{aligned}
\sum_{i=1}^{k}\sum_{j=1}^{k}|Y_i \cap X_j^*| \cdot \|\mathbf{c}_j^* - \mathbf{c}'_i\|_2^z &\leq \mathsf{D}_1^z\sum_{i=1}^{k}\sum_{j=1}^{k}|Y_i \cap X_j^*| \cdot \|\Pi(\mathbf{c}_j^*) - \Pi(\mathbf{c}'_i)\|_2^z \\
&\leq \mathsf{D}_1^z \cdot 2^z \cdot \mathrm{val}(\Pi, c^*, Y_1, \ldots, Y_k). \qquad (5.5)
\end{aligned}$$

Combining Eq. (5.4)) and Eq. (5.5), we may upper bound

$$\begin{aligned}\operatorname*{cost}_z(Y_1,\dots,Y_n) &\le 2^{z-1}\cdot\operatorname*{cost}_z(X_1^*,\dots,X_k^*)+\mathsf{D}_1^z\cdot 2^{2z-1}\cdot\operatorname{val}(\Pi,c^*,Y_1,\dots,Y_k)\\ &= 2^{z-1}\cdot\operatorname*{cost}_z(X_1^*,\dots,X_k^*)+\mathsf{D}_1^z\cdot 2^{2z-1}\sum_{i=1}^k\sum_{j=1}^k|Y_i\cap X_j^*|\cdot\|\Pi(\mathbf{c}_j^*)-\hat{\mathbf{c}}_i\|_2^z.\end{aligned} \tag{5.6}$$

We continue upper bounding the right-most expression in Eq. (5.6) by applying the triangle inequality:

$$\begin{aligned}&\sum_{i=1}^k\sum_{j=1}^k|Y_i\cap X_j^*|\cdot\|\Pi(\mathbf{c}_j^*)-\hat{\mathbf{c}}_i\|_2^z\\ &\quad\le 2^{z-1}\sum_{i=1}^k\sum_{j=1}^k\sum_{x\in Y_i\cap X_j^*}\|\Pi(\mathbf{x})-\Pi(\mathbf{c}_j^*)\|_2^z+2^{z-1}\sum_{i=1}^k\sum_{j=1}^k\sum_{\mathbf{x}\in Y_i\cap X_j^*}\|\Pi(\mathbf{x})-\hat{\mathbf{c}}_i\|_2^z\\ &\quad\le 2^{z-1}\mathsf{D}_2\cdot\operatorname*{cost}_z(X_1^*,\dots,X_k^*)+2^{z-1}\cdot\operatorname*{cost}_z(\Pi(Y_1),\dots,\Pi(Y_k)).\end{aligned} \tag{5.7}$$

By the third assumption of the lemma, we note that

$$\begin{aligned}\operatorname*{cost}_z(\Pi(Y_1),\dots,\Pi(Y_k)) &\le \mathsf{D}_3\cdot\min_{\mathbf{c}_1,\dots,\mathbf{c}_k\in\mathbb{R}^t}\sum_{\mathbf{x}\in X}\min_{j\in[k]}\|\Pi(\mathbf{x})-\mathbf{c}_j\|_2^z\\ &\le \mathsf{D}_3\cdot\sum_{j=1}^k\sum_{\mathbf{x}\in X_j}\|\Pi(\mathbf{x})-\Pi(\mathbf{c}_j^*)\|_2^z\le\mathsf{D}_3\mathsf{D}_2\cdot\operatorname*{cost}_z(X_1^*,\dots,X_k^*).\end{aligned} \tag{5.8}$$

Summarizing by plugging Eq. (5.7) and Eq. (5.8) into Eq. (5.6), we can upper bound

$$\operatorname*{cost}_z(Y_1,\dots,Y_n)\le\left(2^{z-1}+2^{3z-2}\cdot\mathsf{D}_1^z\mathsf{D}_2(1+\mathsf{D}_3)\right)\cdot\operatorname*{cost}_z(X_1^*,\dots,X_k^*).$$

5.5 Efficient Seeding in One Dimension

In this section, we prove Theorem 5.1, which shows an upper bound for the running time of our algorithm PRONE. As in Theorem 5.15, we consider a generalized version of PRONE where we run k-means++ seeding for general $z\ge 1$ (Definition 5.7) in Step 2. We prove the following generalized version of Theorem 5.1:

Theorem 5.20 (Theorem 5.1 for general $z\ge 1$)**.** *Let $\mathbf{X}=(\mathbf{x}_1,\dots,\mathbf{x}_n)\subset\mathbb{R}^d$ be a dataset consisting of n points in d dimensions. Assume that $d\le\mathsf{nnz}(\mathbf{X})$, which can be ensured after removing redundant dimensions $j\in[d]$ where the j-th coordinate of every $\mathbf{x}_i$ is zero. For any $z\ge 1$, the algorithm* PRONE *(for general z as in Theorem 5.15) has expected running time $\mathcal{O}\left(\mathsf{nnz}(\mathbf{X})+2^{z/2}n\log n\right)$ on $\mathbf{X}$. For any $\delta\in(0,1/2)$, with probability at least $1-\delta$, the algorithm runs in time $\mathcal{O}\left(\mathsf{nnz}(\mathbf{X})+2^{z/2}n\log(n/\delta)\right)$. Moreover, the algorithm always runs in time $\mathcal{O}\left(\mathsf{nnz}(\mathbf{X})+n\log n+nk\right)$.*

To prove Theorem 5.1, we show an efficient implementation (Algorithm 18) of the k-means++ seeding procedure that runs in expected time $\mathcal{O}\left(2^{z/2}n\log n\right)$ for one-dimensional points (Lemma 5.22). A naive implementation of the seeding procedure would take $\Theta\left(nk\right)$ time in one dimension because we need $\Theta\left(n\right)$ time to update p_i and sample from the D^2 distribution to add each of the k centers. To obtain an improved and provable running time, we use a basic binary tree data structure to sample from the D^2 distribution more efficiently, and we use a potential argument to bound the number of updates to p_i.

The data structure S we use in Algorithm 18 can be implemented as a basic binary tree, as described in more detail in Section 5.6. The data structure S keeps track of n nonnegative numbers $s_1,\ldots,s_n$ corresponding to $p_1^z,\ldots,p_n^z$ and it supports the following operations:

1. $\texttt{Initialize}(a)$. Given an array $a=(a_1,\ldots,a_n)\in\mathbb{R}^n_{\geq 0}$, the operation $\texttt{Initialize}(a)$ creates a data structure S that keeps track of the numbers $s_1,\ldots,s_n$ initialized so that $(s_1,\ldots,s_n)=(a_1,\ldots,a_n)$. This operation runs in $\mathcal{O}\left(n\right)$ time.

2. $\texttt{Sum}(S)$. The operation $\texttt{Sum}(S)$ returns the sum $s_1+\cdots+s_n$. This operation runs in $\mathcal{O}\left(1\right)$ time, as the value will be maintained as the data structure is updated.

3. $\texttt{Find}(S,r)$. Given a number $r\in[0,\sum_{i=1}^n s_i)$, the operation $\texttt{Find}(S,r)$ returns the unique index $\ell\in\{1,\ldots,n\}$ such that

$$\sum_{i=1}^{\ell-1} s_i \leq r < \sum_{i=1}^{\ell} s_i.$$

   This operation runs in $\mathcal{O}\left(\log n\right)$ time.

4. $\texttt{Update}(S,a,i_1,i_2)$. Given an array $a=(a_1,\ldots,a_n)\in\mathbb{R}^n_{\geq 0}$ and indices i_1,i_2 satisfying $1\leq i_1\leq i_2\leq n$, the operation $\texttt{Update}(S,a,i_1,i_2)$ performs the updates $s_i\leftarrow a_i$ for every $i=i_1,i_1+1,\ldots,i_2$. This operation runs in $\mathcal{O}\left((i_2-i_1+1)+\log n\right)$ time.

The following claim shows that Algorithm 18 correctly implements the k-means++ seeding procedure in one dimension.

Claim 5.21. *Consider the values of $t,a_1,\ldots,a_n$ and the data structure S at the beginning of each iteration of the for-loop (i.e., right before Line 6). Let $s_1,\ldots,s_n$ be the numbers the data structure S keeps track of. For every $i=1,\ldots,n$, define $p_i:=\min_{t'=1,\ldots,t-1}|x_i-x_{\ell_{t'}}|$. Then $s_i=a_i=p_i^z$ for every $i=1,\ldots,n$. Consequently, the distribution of ℓ_t at Line 7 conditioned on the execution history so far satisfies $\mathbf{Pr}\left[\ell_t=i\right]=p_i^z/\sum_{i'=1}^n p_{i'}^z$ for every $i=1,\ldots,n$.*

The claim follows immediately by induction over the iterations of the for-loop based on the description of the data structure S and its operations above. The following lemma bounds the running time of Algorithm 18:

Algorithm 18: Efficient k-means++ seeding in one dimension

Input: Points $x_1, \ldots, x_n \in \mathbb{R}$; $k \in \mathbb{Z}$ satisfying $1 \leq k \leq n$; real number $z \geq 1$.
Output: Centers $x_{\ell_1}, \ldots, x_{\ell_k} \in \mathbb{R}$; assignment $\sigma : [n] \to [k]$.

1 Sort and re-order the points so that $x_1 \leq \cdots \leq x_n$;
2 Choose ℓ_1 uniformly at random from $\{1, \ldots, n\}$;
3 Initialize $a = (a_1, \ldots, a_n)$ by setting $a_i \leftarrow |x_i - x_{\ell_1}|^z$ for every $i = 1, \ldots, n$;
4 $S \leftarrow \texttt{Initialize}(a)$;
5 **for** $t = 2, \ldots, k$ **do**
6 Choose r uniformly at random from $[0, \texttt{Sum}(S))$;
7 $\ell_t \leftarrow \texttt{Find}(S, r)$; $a_{\ell_t} \leftarrow 0$; $i \leftarrow \ell_t - 1$; $j \leftarrow \ell_t + 1$;
8 **while** $i \geq 0$ **and** $|x_i - x_{\ell_t}|^z < a_i$ **do**
9 $a_i \leftarrow |x_i - x_{\ell_t}|^z$;
10 $i \leftarrow i - 1$;
11 **while** $j \leq n$ **and** $|x_j - x_{\ell_t}|^z < a_j$ **do**
12 $a_j \leftarrow |x_j - x_{\ell_t}|^z$;
13 $j \leftarrow j + 1$;
14 `Update` $(S, a, i + 1, j - 1)$;
15 Sort and re-order $\ell_1, \ldots, \ell_k$ so that $\ell_1 \leq \cdots \leq \ell_k$;
16 $i \leftarrow 1$; $j \leftarrow 1$;
17 **while** $i \leq n$ **do** `/* Assign` x_i `to the closest center` $x_{\ell_{\sigma(i)}}$ `among` $x_{\ell_1}, \ldots, x_{\ell_k}$. `*/`
18 **if** $j < k$ **and** $|x_i - x_{\ell_j}| \geq |x_i - x_{\ell_{j+1}}|$ **then**
19 $j \leftarrow j + 1$;
20 **else**
21 $\sigma(i) \leftarrow j$;
22 $i \leftarrow i + 1$;
23 **return** $x_{\ell_1}, \ldots, x_{\ell_k}$ and σ (converted to the old ordering of $x_1, \ldots, x_n$ before Line 1);

Lemma 5.22 (Lemma 5.4 for general $z \geq 1$)**.** *The expected running time of Algorithm 18 is $\mathcal{O}\left(2^{z/2} n \log n\right)$. For any $\delta \in (0, 1/2)$, with probability at least $1 - \delta$, Algorithm 18 runs in time $\mathcal{O}\left(2^{z/2} n \log(n/\delta)\right)$. Moreover, Algorithm 18 always runs in time $\mathcal{O}\left(n \log n + nk\right)$.*

Before proving Lemma 5.22, we first use it to prove Theorem 5.20.

Proof of Theorem 5.20. Lemma 5.22 bounds the running time of Step 2 of our algorithm PRONE defined in Section 5.2. Now we show that Step 1 (random one-dimensional projection) can be performed in time $\mathcal{O}\left(\mathsf{nnz}(\mathbf{X}) + n\right)$. Indeed, $\mathbf{x}'_i$ can be computed as $\mathbf{x}'_i = \sum_j \mathbf{x}_{ij}\mathbf{v}_j$ where the sum is over all the non-zero coordinates $\mathbf{x}_{ij}$ of $\mathbf{x}_i$, and each $\mathbf{v}_j$ is drawn independently from the one-dimensional standard Gaussian (the value of $\mathbf{v}_j$ should be shared for all i). The time needed to compute $\mathbf{x}'_1, \ldots, \mathbf{x}'_n$ in this way is $\mathcal{O}\left(\mathsf{nnz}(\mathbf{X}) + n\right)$. In Step 3 of PRONE, we compute the center of mass for every cluster. This can be done in time $\mathcal{O}\left(\mathsf{nnz}(\mathbf{X}) + n\right)$ by summing up the points in each cluster and dividing each sum by the number of points in that cluster. □

The key step towards proving Lemma 5.22 is to bound the number of updates to a at Lines 9 and 12. As the algorithm starts by sorting the n input points, which can be done in time $\mathcal{O}\left(n \log n\right)$, we can assume that the points are sorted such that $x_1 \leq \cdots \leq x_n$. For $i = 1, \ldots, n$ and $t = 2, \ldots, k$, we define $\xi(i, t) = 1$ if a_i is updated at Line 9 in iteration t and define $\xi(i, t) = 0$ otherwise. Here, we denote each iteration of the for-loop beginning at Line 5 by the value of the iterate t. We define $u_i := \sum_{t=2}^{k} \xi(i, t)$ to be the number of times a_i gets updated at Line 9. The following lemma gives upper bounds on u_i both in expectation and with high probability:

Lemma 5.23. *For every $i = 1, \ldots, n$, it holds that*

$$\mathbb{E}\left[u_i\right] \leq \mathcal{O}\left(2^{z/2} \log n\right).$$

Moreover, for some absolute constant $B > 0$ and for every $\delta \in (0, 1/2)$, it holds that

$$\mathbf{Pr}\left[u_i \leq B 2^{z/2} \log(n/\delta)\right] \geq 1 - \delta.$$

Before proving Lemma 5.23, we first use it to prove Lemma 5.22.

Proof of Lemma 5.22. Recall that for $i = 1, \ldots, n$ and $t = 2, \ldots, k$, we define $\xi(i, t) = 1$ if a_i is updated at Line 9 in iteration t and define $\xi(i, t) = 0$ otherwise. Similarly, for $j = 1, \ldots, n$ and $t = 2, \ldots, k$, we define $\xi'(j, t) = 1$ if a_j is updated at Line 12 in iteration t and define $\xi'(j, t) = 0$ otherwise.

The computation at Lines 1-4 takes $\mathcal{O}\left(n \log n\right)$ time. The computation at Lines 15-23 takes $\mathcal{O}\left(k \log k + n\right) = \mathcal{O}\left(n \log n\right)$ time. For $t = 2, \ldots, k$, iteration t of the for-loop takes time $\mathcal{O}\left(\log n + \sum_{i=1}^{n} \xi(i, t) + \sum_{i=1}^{n} \xi'(i, t)\right)$. Summing them up, the total running time of Algorithm 18 is

$$\mathcal{O}\left(n \log n + \sum_{i=1}^{n} \sum_{t=2}^{k} \xi(i, t) + \sum_{i=1}^{n} \sum_{t=2}^{k} \xi'(i, t)\right). \tag{5.9}$$

By Lemma 5.23,

$$\mathbb{E}\left[\sum_{i=1}^{n}\sum_{t=2}^{k}\xi(i,t)\right] = \mathbb{E}\left[\sum_{i=1}^{n}u_i\right] = \mathcal{O}\left(2^{z/2}n\log n\right). \tag{5.10}$$

Also, for any $\delta' \in (0,1/2)$, setting $\delta = \delta'/n$ in Lemma 5.23, by the union bound we have

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i=1}^{n}\sum_{t=2}^{k}\xi(i,t) \le 2B2^{z/2}n\log(n/\delta')\right] &\ge \mathbf{Pr}\left[\sum_{i=1}^{n}\sum_{t=2}^{k}\xi(i,t) \le B2^{z/2}n\log(n/\delta)\right]\\ &\ge 1-n\delta\\ &= 1-\delta'. \end{aligned}\tag{5.11}$$

Similarly to (5.10) and (5.11) we have

$$\mathbb{E}\left[\sum_{i=1}^{n}\sum_{t=2}^{k}\xi'(i,t)\right] = \mathcal{O}\left(2^{z/2}n\log n\right), \quad \text{and} \tag{5.12}$$

$$\mathbf{Pr}\left[\sum_{i=1}^{n}\sum_{t=2}^{k}\xi'(i,t) \le 2B2^{z/2}n\log(n/\delta')\right] \ge 1-\delta'. \tag{5.13}$$

Plugging (5.10) and (5.11) into (5.9) proves that the expected running time of Algorithm 18 is $\mathcal{O}\left(2^{z/2}n\log n\right)$. Choosing $\delta' = \delta/2$ for the δ in Lemma 5.22 and plugging (5.11) and (5.13) into (5.9), we can use the union bound to conclude that with probability at least $1-\delta$ Algorithm 18 runs in time $\mathcal{O}\left(2^{z/2}n\log(n/\delta)\right)$. Finally, plugging $\xi(i,t) \le 1$ and $\xi'(i,t) \le 1$ into (5.9), we get that Algorithm 18 always runs in time $\mathcal{O}\left(n\log n + nk\right)$. □

To prove Lemma 5.23, for $i = 0,\ldots,n$ and $u = 0,\ldots,u_i$, we define $t(i,u)$ to be the smallest $t \in \{1,\ldots,k\}$ such that $\sum_{t'=2}^{t}\xi(i,t') = u$. That is, a_i gets updated at Line 9 for the u-th time in iteration $t(i,u)$. Our definition implies that $t(i,0) = 1$ and $t(i,u) \in \{2,\ldots,k\}$ for $u = 1,\ldots,u_i$. We define a nonnegative potential function $\eta(i,u)$ as follows and show that it decreases exponentially in expectation as u increases (Lemma 5.24).

Potential Function $\eta(i,u)$. For $t = 2,\ldots,k$, we consider the value of ℓ_t after Line 7 in iteration t. For $u = 1,\ldots,u_i$, we define $\eta(i,u)$ to be $\ell_{t(i,u)} - i$, which is guaranteed to be a positive integer by the definition of $t(i,u)$. Indeed, in the while-loop containing Line 9, i starts from $\ell_t - 1$ and keeps decreasing, so whenever Line 9 is executed, i is smaller than ℓ_t. In particular, a_i is updated at Line 9 in iteration $t(i,u)$ of the for-loop, so we have $i < \ell_{t(i,u)}$. We define $\eta(i,0) = n$, and for $u = u_i+1, u_i+2,\ldots,$ we define $\eta(i,u) = 0$. See Figure 5.2 for an example illustrating the definition of $\eta(i,u)$.

Lemma 5.24 (Potential function decrease)**.** *For any $i \in \{1,\ldots,n\}$ and $u \in \mathbb{Z}_{\ge 0}$,*

$$\mathbb{E}\left[\eta(i,u+1)|\eta(i,0),\ldots,\eta(i,u)\right] \le \max\left\{0, \frac{2^{z/2}}{2^{z/2}+1}\eta(i,u) - \frac{1}{2}\right\}.$$

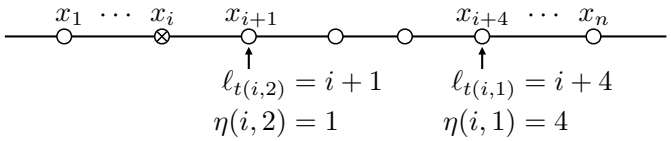

Figure 5.2: An example illustrating the definition of the potential function η. Here, a_i is updated at Line 9 for the first time in iteration $t(i,1)$ when $\ell_{t(i,1)} = i+4$. Then a_i gets updated at Line 9 for the second time in iteration $t(i,2)$ when $\ell_{t(i,2)} = i+1$. We always have $\ell_t > i$ whenever a_i is updated at Line 9, and η is the difference between ℓ_t and i. Thus in this example $\eta(i,1) = 4$ and $\eta(i,2) = 1$. If a_i is never updated at Line 9 after iteration $t(i,2)$, we define $\eta(i,u) = 0$ for $u = 3, 4, \ldots$.

Before proving Lemma 5.24, we first use it to prove Lemma 5.23. Intuitively, Lemma 5.24 says that $\eta(i,u)$ decreases exponentially (in expectation) as a function of u. Since $\eta(i,u)$ is always a nonnegative integer, we should expect $\eta(i,u)$ to become zero as soon as u exceeds a small threshold. Moreover, our definition ensures $\eta(i,u_i) > 0$, so u_i must be smaller than the threshold. This allows us to show upper bounds for u_i and prove Lemma 5.23.

Proof of Lemma 5.23. Our definition of η ensures $\eta(i,u_i) \geq 1$. By Lemma 5.24 and Lemma 5.26,

$$\mathbb{E}\left[u_i + 1\right] \leq \frac{\ln n}{\ln \frac{2^{z/2}+1}{2^{z/2}}} + \frac{1}{1 - \frac{2^{z/2}}{2^{z/2}+1}} = \mathcal{O}\left(2^{z/2} \ln n\right) + 2^{z/2} + 1.$$

This implies $\mathbb{E}\left[u_i\right] = \mathcal{O}\left(2^{z/2} \log n\right)$. Moreover, by Lemma 5.24,

$$\mathbb{E}\left[\eta(i,u)\right] \leq \eta(i,0) \left(\frac{2^{z/2}}{2^{z/2}+1}\right)^u = n \left(\frac{2^{z/2}}{2^{z/2}+1}\right)^u,$$

and thus, by Markov's inequality,

$$\mathbf{Pr}\left[u_i \geq u\right] = \mathbf{Pr}\left[\eta(i,u) \geq 1\right] \leq n \left(\frac{2^{z/2}}{2^{z/2}+1}\right)^u.$$

For any $\delta \in (0, 1/2)$, choosing $u = \ln(n/\delta) / \ln(\frac{2^{z/2}+1}{2^{z/2}}) = \mathcal{O}\left(2^{z/2} \log(n/\delta)\right)$ in the inequality above gives $\mathbf{Pr}\left[u_i \geq u\right] \leq \delta$. □

We need the following helper lemma to prove Lemma 5.24.

Lemma 5.25. *The following holds at the beginning of each iteration of the for-loop in Algorithm 18, i.e., right before Line 6 is executed. Choose an arbitrary $i = 1, \ldots, n$ and define*

$$L := \{i\} \cup \{\ell \in \mathbb{Z} : i < \ell \leq n, |x_i - x_\ell|^z < a_i\}. \tag{5.14}$$

Then for $\ell, \ell' \in L$ satisfying $\ell < \ell'$, it holds that $a_{\ell'} \leq 2^z a_\ell$.

Proof. For $t = 2, \dots, k$, at the beginning of iteration t, the values $\ell_1, \dots, \ell_{t-1}$ have been determined. For every $x \in \mathbb{R}$, let $\rho(x)$ denote the value among $x_{\ell_1}, \dots, x_{\ell_{t-1}}$ closest to x. By Claim 5.21, $a_i = |x_i - \rho(x_i)|^z$ for every $i \in [n]$. Now for a fixed $i \in [n]$, define L as in (5.14) and consider $\ell, \ell' \in L$ satisfying $\ell < \ell'$. It is easy to see that $x_i \le \rho(x_\ell) \le x_{\ell'}$ cannot hold because otherwise $a_i \le |x_i - \rho(x_\ell)|^z \le |x_i - x_{\ell'}|^z < a_i$, a contradiction. For the same reason, the inequality $x_i \le \rho(x_{\ell'}) \le x_{\ell'}$ cannot hold. Thus there are only three possible orderings of $x_i, x_\ell, x_{\ell'}, \rho(x_\ell), \rho(x_{\ell'})$:

1. $\rho(x_\ell) = \rho(x_{\ell'}) < x_i \le x_\ell \le x_{\ell'}$;
2. $\rho(x_\ell) < x_i \le x_\ell \le x_{\ell'} < \rho(x_{\ell'})$;
3. $x_i \le x_\ell \le x_{\ell'} < \rho(x_\ell) = \rho(x_{\ell'})$.

In scenario 3, it is clear that $a_{\ell'} = |x_{\ell'} - \rho(x_{\ell'})|^z \le |x_\ell - \rho(x_\ell)|^z = a_\ell$. In the first two scenarios, for any $t' = 0, \dots, t-1$,

$$|x_i - x_{\ell_{t'}}| \ge |x_\ell - x_{\ell_{t'}}| - |x_\ell - x_i| \ge |x_\ell - \rho(x_\ell)| - |x_\ell - x_i| = |x_i - \rho(x_\ell)|.$$

This implies that $\rho(x_\ell)$ is the closest point to x_i among $x_{\ell_1}, \dots, x_{\ell_{t-1}}$. Therefore, $a_i = |x_i - \rho(x_\ell)|^z$. Jensen's inequality ensures $((g+h)/2)^z \le (g^z + h^z)/2$ for any $g, h \ge 0$, which implies $(g+h)^z \le 2^{z-1} g^z + 2^{z-1} h^z$. Therefore,

$$a_{\ell'} \le |x_{\ell'} - \rho(x_\ell)|^z \le 2^{z-1} |x_{\ell'} - x_i|^z + 2^{z-1} |x_i - \rho(x_\ell)|^z < 2^z a_i,$$

whereas

$$a_\ell = |x_\ell - \rho(x_\ell)|^z \ge |x_i - \rho(x_\ell)|^z = a_i.$$

Thus, we have $a_{\ell'} \le 2^z a_\ell$ in all three scenarios. □

Proof of Lemma 5.24. Throughout the proof, we fix $i \in \{1, \dots, n\}$ and $u \in \mathbb{Z}_{\ge 0}$ so that they are deterministic numbers. Algorithm 18 is a randomized algorithm, and when we run it, exactly one of the following four events happens, and we define a random variable t^* accordingly:

1. Event E_1: $u_i < u$. That is, a_i gets updated at Line 9 for less than u times. In this case we have $\eta(i, u+1) = 0$ by our definition of η, and we define $t^* = +\infty$.
2. Event E_2: $u_i = u$ and i is never chosen as ℓ_t at Line 7. In this case we also have $\eta(i, u+1) = 0$, and we also define $t^* = +\infty$.
3. Event E_3: $u_i = u$ and there exists $t \in \{2, 3, \dots, k\}$ such that i is chosen as ℓ_t at Line 7 in iteration t. This t must satisfy $t > t(i, u)$, as all updates to a_i in Line 9 must happen before x_i is chosen as a center. We define $t^* = t$ in this case. Again, we have $\eta(i, u+1) = 0$ in this case.
4. Event E_4: $u_i > u$. We define $t^* := t(i, u+1) > t(i, u)$ in this case.

Define $E^* := E_3 \cup E_4$. Since $\eta(i, u+1) = 0$ under E_1 and E_2, it suffices to prove that[5]

$$\mathbb{E}\left[\eta(i,u+1) | \eta(i,0), \ldots, \eta(i,u), E^*\right] \leq \frac{2^{z/2}}{2^{z/2}+1}\eta(i,u) - \frac{1}{2}. \tag{5.15}$$

By our definition, the random variable t^* takes its value in $\{2, 3, \ldots, k\} \cup \{+\infty\}$. Moreover, $t^* = +\infty$ if and only if E^* does not happen. Therefore, to prove (5.15), it suffices to prove the following for every $t_0 = 2, 3, \ldots, k$:

$$\mathbb{E}\left[\eta(i,u+1) | \eta(i,0), \ldots, \eta(i,u), t^* = t_0\right] \leq \frac{2^{z/2}}{2^{z/2}+1}\eta(i,u) - \frac{1}{2}. \tag{5.16}$$

Consider a fixed $t_0 \in \{2, 3, \ldots, k\}$. For $t^* = t_0$ to happen, the following must hold during the execution of Algorithm 18 before iteration t_0: a_i has been updated at Line 9 for exactly u times, and i has not been chosen as ℓ_t at Line 7. Thus, the rest of the proof assumes that the execution history H of Algorithm 18 before iteration t_0 satisfies this property. Now we know that the values $\eta(i,0), \ldots, \eta(i,u)$ are determined by the execution history H. Lines 6-7 guarantee that the distribution of ℓ_{t_0} satisfies

$$\mathbf{Pr}\left[\ell_{t_0} = \ell | H\right] = \frac{a_\ell}{\sum_{j=1}^n a_j} \quad \text{for every } \ell = 1, \ldots, n,$$

where we use the values $a_1, \ldots, a_n$ right before iteration t_0 is executed. Moreover, conditioned on H, we have $t^* = t_0$ if and only if $\ell_{t_0} \in L$, where

$$L = \{i\} \cup \{\ell \in \mathbb{Z} : i < \ell \leq n, |x_\ell - x_i|^z < a_i\}.$$

Therefore, if we further condition on $t^* = t_0$, we have $\ell_{t_0} \in L$ and

$$\mathbf{Pr}\left[\ell_{t_0} = \ell | H, t^* = t_0\right] = \frac{a_\ell}{\sum_{j \in L} a_j} \quad \text{for every } \ell \in L. \tag{5.17}$$

When $t^* = t_0$, we have $\eta(i, u+1) = \ell_{t_0} - i$. Therefore, to prove (5.16), it suffices to show that

$$\mathbb{E}\left[\ell_{t_0} - i | H, t^* = t_0\right] \leq \frac{2^{z/2}}{2^{z/2}+1}\eta(i,u) - \frac{1}{2}. \tag{5.18}$$

It is clear that we can write L as $L = \{i, i+1, \ldots, \ell^*\}$ for some integer $\ell^* \geq i$. If $u > 0$, Claim 5.21 implies $a_i \leq |x_i - x_{\ell_{t(i,u)}}|^z$, and thus $\ell^* < \ell_{t(i,u)}$ and $\ell^* - i \leq \ell_{t(i,u)} - i = \eta(i,u)$. If $u = 0$, we have $\eta(i,u) = n$, so it also holds that $\ell^* - i \leq \eta(i,u)$. By (5.17) and Lemma 5.25, we can set $\gamma = 2^z$ in Lemma 5.27 to get

$$\mathbb{E}\left[\ell_{t_0} - i | H, t^* = t_0\right] \leq \frac{2^{z/2}}{2^{z/2}+1}(\ell^* - i) - \frac{1}{2} \leq \frac{2^{z/2}}{2^{z/2}+1}\eta(i,u) - \frac{1}{2}.$$

This proves (5.18) and thus proves the lemma. □

[5] We have $\eta(i, u+1) = 0$ also for E_3, so one can also simply choose $E^* = E_4$. Choosing $E^* = E_3 \cup E_4$ helps us get improved constants in our bound.

5.5.1 Helper Lemmas

Lemma 5.26. *Let $M \geq 1$ and $\lambda \in (0,1)$ be parameters. Let $\alpha_0, \alpha_1, \ldots \in [0, +\infty)$ be random variables satisfying $\alpha_0 = M$ and $\mathbb{E}[\alpha_{i+1} | \alpha_1, \ldots, \alpha_i] \leq \lambda \alpha_i$ for every $i = 0, 1, \ldots$. Let $t \geq 0$ be the smallest integer satisfying $\alpha_t < 1$. Then*

$$\mathbb{E}[t] \leq \frac{\ln M}{\ln(1/\lambda)} + \frac{1}{1-\lambda}.$$

Proof. For every $j = 0, 1, \ldots$, we define a random variable $t_j := \min\{t, j\}$. By the monotone convergence theorem, it suffices to show that

$$\mathbb{E}[t_j] \leq \frac{\ln M}{\ln(1/\lambda)} + \frac{1}{1-\lambda} \quad \text{for every } j = 0, 1, \ldots. \tag{5.19}$$

We prove (5.19) by induction on j. When $j = 0$, we have $t_j = 0$, and the inequality above holds trivially. We assume that (5.19) holds for an arbitrary $j \in \mathbb{Z}_{\geq 0}$ and show that it also holds with j replaced by $j+1$. We have

$$\mathbb{E}[t_{j+1}] = 1 + \mathbb{E}[t_{j+1} - 1] = 1 + \mathbb{E}[\mathbb{E}[(t_{j+1} - 1) | \alpha_1]]. \tag{5.20}$$

By our definition of t_{j+1}, we have $t_{j+1} - 1 = \min\{t - 1, j\}$. Applying our induction hypothesis on the sequence $\alpha_1, \alpha_2, \ldots$, we have

$$\mathbb{E}[(t_{j+1} - 1) | \alpha_1] = \mathbb{E}[\min\{t - 1, j\} | \alpha_1] \leq f(\alpha_1), \tag{5.21}$$

where

$$f(\alpha) = \begin{cases} \frac{\alpha}{1-\lambda}, & \text{if } \alpha \in [0, 1); \\ \frac{\ln \alpha}{\ln(1/\lambda)} + \frac{1}{1-\lambda}, & \text{if } \alpha \geq 1. \end{cases}$$

It is easy to check that f is an increasing concave function of $\alpha \in [0, +\infty)$ and $1 + f(\lambda\alpha) \leq f(\alpha)$ holds for every $\alpha \geq 1$. Plugging (5.21) into (5.20), we have

$$\mathbb{E}[t_{j+1}] \leq 1 + \mathbb{E}[f(\alpha_1)] \leq 1 + f(\mathbb{E}[\alpha_1]) \leq 1 + f(\lambda M) \leq f(M) = \frac{\ln M}{\ln(1/\lambda)} + \frac{1}{1-\lambda}. \quad \square$$

Lemma 5.27. *Let $\gamma \geq 1$ be a real number. Let $\beta_0, \ldots, \beta_{m-1}$ be non-negative real numbers such that for every $i, j \in \{0, \ldots, m-1\}$ satisfying $i \leq j$, it holds that $\beta_j \leq \gamma \beta_i$. Then,*

$$\sum_{i=0}^{m-1} i\beta_i \leq \left(\frac{\sqrt{\gamma} \cdot m}{\sqrt{\gamma} + 1} - \frac{1}{2} \right) \sum_{i=0}^{m-1} \beta_i.$$

Proof. The lemma holds trivially if $\beta_0 = 0$ because in this case $\beta_j \leq \gamma \beta_0 = 0$ for every $j = 0, \ldots, m-1$. We thus assume w.l.o.g. that $\beta_0 > 0$. Define τ to be the unique real number satisfying

$$\tau \sum_{i=0}^{m-1} \beta_i - \sum_{i=0}^{m-1} i\beta_i = 0.$$

It is clear that $\tau \in [0, m-1]$. Our goal is to prove that

$$\tau \leq \frac{\sqrt{\gamma} \cdot m}{\sqrt{\gamma}+1} - \frac{1}{2}. \tag{5.22}$$

Define $\beta_* := \min_{0 \leq i \leq \tau} \beta_i$. For every $i = 0, \ldots, m-1$, we have $\beta_i \geq \beta_*$ if $i \leq \tau$, and $\beta_i \leq \gamma\beta_*$ if $i > \tau$. Therefore, defining $s := \lfloor \tau \rfloor$, we have

$$\begin{aligned}
0 &= \tau \sum_{i=0}^{m-1} \beta_i - \sum_{i=0}^{m-1} i\beta_i \\
&= \sum_{i=0}^{m-1} (\tau - i)\beta_i \\
&\geq \sum_{i \leq \tau} (\tau - i)\beta_* + \sum_{i > \tau} (\tau - i)\gamma\beta_* && (5.23) \\
&= \frac{(s+1)(2\tau - s)}{2} \cdot \beta_* + \frac{(m-s-1)(2\tau - m - s)}{2} \cdot \gamma\beta_*. && (5.24)
\end{aligned}$$

Now, we show that $\beta_* > 0$. For the sake of contradiction, assume $\beta_* = 0$. We already assumed that $\beta_0 > 0$, so $\beta_* \neq \beta_0$. By the definition of β_*, this means that $\tau > 0$ and inequality (5.23) is strict, leading to the false claim of

$$0 > \sum_{i \leq \tau} (\tau - i)\beta_* + \sum_{i > \tau} (\tau - i)\gamma\beta_* = 0.$$

Therefore, $\beta_* > 0$ must hold. Now we know that (5.24) implies

$$(s+1)(2\tau - s) + (m-s-1)(2\tau - m - s)\gamma \leq 0.$$

Treating s as a real-valued variable, the left-hand side is minimized when $s = \tau - 1/2$, giving us

$$(\tau + 1/2)^2 - (m - \tau - 1/2)^2\gamma \leq 0.$$

The inequality above implies

$$(\tau + 1/2)^2 \leq (m - \tau - 1/2)^2\gamma.$$

Taking square root for both sides and solving for τ gives (5.22). □

5.6 Data Structure for Fast Sampling in Seeding

In Section 5.5, our Algorithm 18 uses a binary tree data structure S that keeps track of n nonnegative numbers $s_1, \ldots, s_n$ and supports several operations. Here, we describe the implementation of this data structure. We assume that $n = 2^q$ for some nonnegative integer q. This is without loss of generality because we can choose n' to be the number that satisfy $n \leq n' < 2n$ and $n' = 2^q$ for some $q \in \mathbb{Z}_{\geq 0}$ and consider $s_1, \ldots, s_n, s_{n+1}, \ldots, s_{n'}$

with $s_{n+1} = \cdots = s_{n'} = 0$. Under this assumption, the data structure S is a complete binary tree with $q+1$ layers indexed by $0, \ldots, q$. In each layer $\zeta = 0, \ldots, q$ there are 2^ζ nodes each corresponding to a set of indices from $\{1, \ldots, n\}$. The root, denoted by $v_1^{(0)}$, is the unique node in layer 0 and it corresponds to the entire set $V_1^{(0)} := \{1, \ldots, n\}$. For $\zeta = 0, \ldots, q-1$, each node $v_j^{(\zeta)}$ in the ζth layer has two children $v_{2j-1}^{(\zeta+1)}, v_{2j}^{(\zeta+1)}$ in the $(\zeta+1)$th layer corresponding to the sets $V_{2j-1}^{(\zeta+1)}, V_{2j}^{(\zeta+1)}$, respectively, where $V_{2j-1}^{(\zeta+1)}$ is the smaller half of $V_j^{(\zeta)}$ and $V_{2j}^{(\zeta+1)}$ is the larger half. Thus,

$$V_j^{(\zeta)} = \{i \in \mathbb{Z} : (j-1)2^{q-\zeta} < i \leq j2^{q-\zeta}\}.$$

Each node $v_j^{(\zeta)}$ in the tree stores a sum $s_j^{(\zeta)} := \sum_{i \in V_j^{(\zeta)}} s_i$.

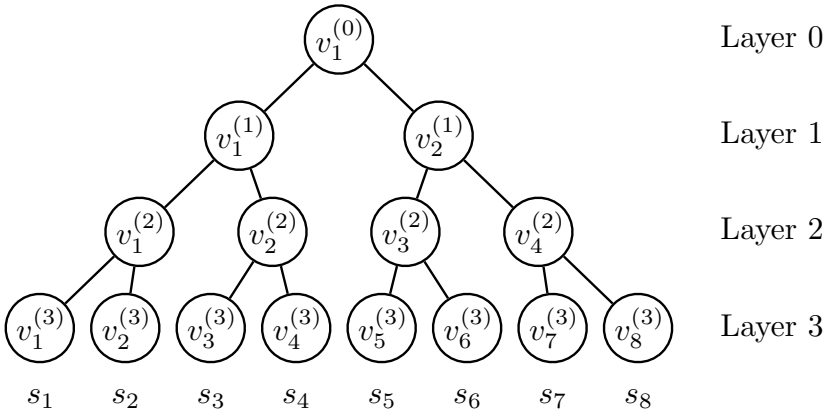

Figure 5.3: An example of a basic binary tree data structure with $q = 3$.

The data structure supports the four types of operations needed in Section 5.5 as follows:

1. `Initialize`(a). Recursively run `Initialize` on the first half $(a_1, \ldots, a_{n/2})$ and the second half $(a_{n/2+1}, \ldots, a_n)$ to obtain the two subtrees rooted at $v_1^{(1)}$ and $v_2^{(1)}$. Then add a root $v_1^{(0)}$ that stores $s_1^{(0)} \leftarrow s_1^{(1)} + s_2^{(1)}$.

2. `Sum`(S). Simply output $s_1^{(0)}$.

3. `Find`(S, r). If $r < s_1^{(1)}$, recursively call `Find` on the left subtree rooted at $v_1^{(1)}$. Otherwise, recursively call `Find` on the right subtree rooted at $v_2^{(1)}$ with r replaced by $r - s_1^{(1)}$. Once we reach a leaf $v_\ell^{(q)}$, return ℓ.

4. `Update`(S, a, i_1, i_2). If $i_2 \leq n/2$, recursively call `Update` on the left subtree rooted at $v_1^{(1)}$. If $i_1 > n/2$, recursively call `Update` on the right subtree rooted at $v_2^{(1)}$. Otherwise, we have $i_1 \leq n/2 < i_2$ and we call `Update` on the left subtree with

indices $i_1, n/2$ and call `Update` on the right subtree with indices $n/2+1, i_2$. In all cases, we update $s_1^{(0)} \leftarrow s_1^{(1)} + s_2^{(1)}$ as the final step. The running time is proportional to the number of nodes we update. We need to update $s_j^{(\zeta)}$ stored at $v_j^{(\zeta)}$ only if $V_j^{(\zeta)} \cap \{i_1, \ldots, i_2\} \neq \emptyset$. For each ζ, the number of such j is at most $(i_2 - i_1 + 1)/2^{q-\zeta} + 2$. Summing up over $\zeta = 0, \ldots, q$, the total number of nodes we need to update is $\mathcal{O}((i_2 - i_1 + 1) + q) = \mathcal{O}((i_2 - i_1 + 1) + \log n)$.

5.7 Experimental Results

In this section, we outline the experimental evaluation of our algorithm. The experiments evaluate the algorithms in two different ways. For each, we measure the running time and the k-means cost of the resulting solution (the sum of squares of point-to-center-assigned distances).

1. First, we evaluate our algorithm as part of a pipeline incorporating a coreset construction – the expected use case for our algorithm.

2. Second, we evaluate our algorithm by itself for approximate k-means clustering and compare it to k-means++ [20]. As per Theorems 5.1 and 5.2, we expect our algorithm to be much faster but output an assignment of higher cost. Our goal is to quantify these differences empirically.

3. Finally, we evaluate our the algorithm in the configuration outlined by Corollary 5.3, where we use the sped up coreset construction in conjunction with k-means++ to improve the approximation ratio.

All experiments were run on Linux using a notebook with a 3.9 GHz 12th generation Intel Core i7 six-core processor and 32 GiB of RAM. All algorithms were implemented in C++, using the `blaze` library for matrix and vector operations performed on the dataset unless specified differently below. The code is publicly available on GitHub[6].

For our experiments, we use the following four datasets:

1. **KDD** [187]: Training data for the 2004 KDD challenge on protein homology. The dataset consists of 145751 observations with 77 real-valued features.

2. **Song** [44]: Timbre information for 515345 songs with 90 features each, used for year prediction.

3. **Census** [116]: 1990 US census data with 2458285 observations, each with 68 categorical features.

4. **Gaussian**: A synthetic dataset consisting of 240005 points of dimension 4. The points are generated by placing a standard normal distribution at a large positive distance from the origin on each axis and sampling 30000 points. The points are

[6] `PRONE` GitHub repository: `https://github.com/boredoms/prone`

then mirrored so the center of mass remains at the origin. Finally, 5 points are placed on the origin. This is an adversarial example for lightweight coresets [29], which are unlikely to sample points close to the mean of the dataset.

5.7.1 Coreset Construction Comparison

Experimental Setup. Coreset constructions (with multiplicative approximation guarantees) always proceed by first finding an approximate clustering, which constitutes the bulk of the work. The approximate clustering defines a "sensitivity sampling distribution" (we expand on this in Section 5.3, see also [28]), and a coreset is constructed by repeatedly sampling from the sensitivity sampling distribution. In our first experiment, we evaluate the choice of initial approximation algorithm used to define the sensitivity sampling distribution. We compare the use of k-means++ and `PRONE`. In addition, we also compare the lightweight coresets of Bachem et al. [29], which uses the distance to the center of mass as an approximation of the sensitivity sampling distribution.

For the remainder of this section, we refer to sensitivity sampling using k-means++ as *Sensitivity* and lightweight coresets as *Lightweight*. All three algorithms produce a coreset, and the experiment will measure the running time of the three algorithms (Table 5.1) and the quality of the resulting coresets (Fig. 5.4).

Once a coreset is constructed for each of the algorithms, we evaluate the quality of the coreset by computing the cost of the centers found when clustering the coreset (see Definition 5.6). We run a state-of-the-art implementation of Lloyd's k-means algorithm from the `scikit-learn` library [256] with the default configuration (repeating 15 times and reporting the mean cost to reduce the variance). The resulting quality of the coresets is compared to a (computationally expensive) *baseline*, which runs `k-means++` from the `scikit-learn` library, followed by Lloyd's algorithm with the default configuration on the entire dataset (repeated 5 times to reduce variance).

We evaluate various choices of k ($\{10, 100, 1000\}$) as well as coresets at various relative sizes, $\{0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1\}$ times the size of the dataset. We use as performance metrics (1) a relative cost, which measures the average cost of the k-means solutions returned by Lloyd's algorithm on each coreset divided by the baseline, and (2) the running time of the coreset construction algorithm.

Results on Coreset Constructions. *Relative cost.* Figure 5.4 shows the coreset size (x-axis) versus the relative cost (y-axis). Each "row" of Fig. 5.4 corresponds to a different value for $k \in \{10, 100, 1000\}$, and each "column" corresponds to a different dataset. Recall that the first three datasets (i.e., the first three columns) are real-world datasets, and the fourth column is the synthetic Gaussian dataset. We note our observations below:

- As expected, on all real-world datasets and all settings of k, the relative cost decreases as the coreset size increases.

- In real-world datasets, the specific relative cost of each coreset construction (*Sen-*

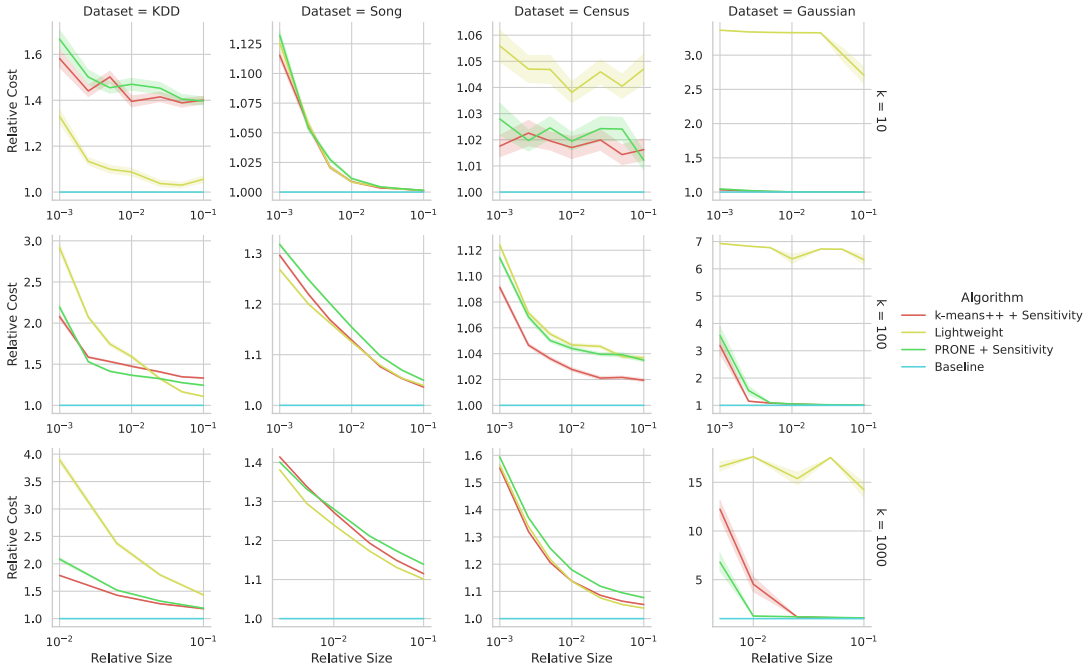

Figure 5.4: Plot of relative cost versus coreset size on our four datasets. The shaded region indicates standard error. There is no data for relative size values where the coreset size is less than k.

stivity, *Lightweight*, and ours) depends on the dataset[7], but roughly speaking, all three share a similar trend. Ours and *Sensitivity* are very close and never more than twice the baseline (usually much better).

- The big difference, distinguishing ours and *Sensitivity* from *Lightweight*, is the fourth column, the synthetic Gaussian dataset. For all settings of k, as the coreset size increases, *Lightweight* exhibits a minimal cost decrease and is a factor of 2.7-17x times worse than ours and *Sensitivity* (as well as the baseline). This is expected, as we constructed the synthetic Gaussian dataset to have arbitrarily high cost with *Lightweight*. Due to its multiplicative approximation guarantee, our algorithm does not suffer this degradation. In that sense, our algorithm is more "robust," and achieves worst-case multiplicative approximation guarantees for all datasets.

Running time. In (the first table in) Table 5.1, we show the running time of the coreset construction algorithms as k increases. Notice that as k increases, the relative speedup of our algorithm and *Lightweight* increases in comparison to *Sensitivity*. This is because our algorithm and *Lightweight* have running time which *does not grow with* k. In contrast, the running time of *Sensitivity* grows linearly in k. In summary, our coreset construction

[7]The spike in relative cost for algorithm *Sensitivity* on the KDD dataset for relative size $5 \cdot 10^{-3}$ is due to outliers.

is between **33-192x** faster than *Sensitivity* for large k. In addition, our algorithm runs about 3-5x slower than *Lightweight*, depending on the dataset. Our analysis also shows this; both algorithms make an initial pass over the dataset, using $\mathcal{O}(nd)$ time, but ours uses an additional $\mathcal{O}(n \log n)$ time to process.

5.7.2 Direct k-Means++ Comparison

Experimental Setup. This experiment compares our algorithm and k-means++ as a stand-alone clustering algorithm, as opposed to as part of a coreset pipeline. We implemented three variants of our algorithm. Each differs in how we sample the random one-dimensional projection. The first is a one-dimensional projection onto a standard Gaussian vector (zero mean and identity covariance). This approach risks collapsing an "important" feature, i.e., a feature with high variance. To mitigate this, we implemented two *data-dependent* variants that use the variance, resp. covariance of the data. Specifically, in the "variance" variant, we use a diagonal covariance matrix, where each entry in the diagonal is set to the empirical variance of the dataset along the corresponding feature. In the "covariance" variant, we use the empirical covariance matrix of the dataset. These variants aim to project along vectors that capture more of the variance of the data than when sampling a vector uniformly at random. Intuitively, the vectors sampled by the biased variants are more correlated with the first principal component of the dataset. For each of our algorithms, we evaluate the k-means cost of the output set C of centers when assigning points to the closest center ($\text{cost}_2(\mathbf{X}, C)$ in Definition 5.6) and when using our algorithm's assignment ($\text{cost}_2(\mathbf{X}, C, \sigma)$ defined in (5.1)).

We evaluated the algorithms for every k in $\{10, 25, 50, 100, 250, 500, 1000, 2500, 5000\}$ and $z = 2$, for solving k-means with the ℓ_2-metric. When evaluating the assignment cost, we ran each of our algorithms 100 times for each k and five times when computing the nearest neighbor assignment, and we report the average cost of the solutions and the average running time. Due to lower variance and much higher runtime, k-means++ was run five times.

Results on Direct k-Means++ Comparison. *Cost.* Figure 5.5 (on top) shows the cost of the centers found by our algorithm compared to those found by the k-means++ algorithm after computing *the optimal assignment of points to the centers* (computing this takes time $\mathcal{O}(ndk)$). That is, we compare the values of $\text{cost}_2(\mathbf{X}, C)$ in Definition 5.6. In summary, the k-means cost of all three variants of our algorithm are roughly the same and closely match that of k-means++. On the Gaussian dataset, one run of the biased algorithm failed to pick a center from the cluster at the origin, leading to a high "outlier" cost and a corresponding spike in the plot.

We also compared the k-means cost for the assignment *computed by our algorithm* (so that our algorithm only takes time $\mathcal{O}(nd + n \log n)$ and *not* $\mathcal{O}(ndk)$) with the cost of k-means++ (bottom row of Fig. 5.5). That is, we compare the values of $\text{cost}_2(\mathbf{X}, C, \sigma)$ defined in (5.1). The clustering cost of our algorithms is higher than that of k-means++. This is the predicted outcome from our theoretical results; recall Theorem 5.2 gives a poly(k)-approximation, as opposed to $\mathcal{O}(\log k)$ from k-means++.

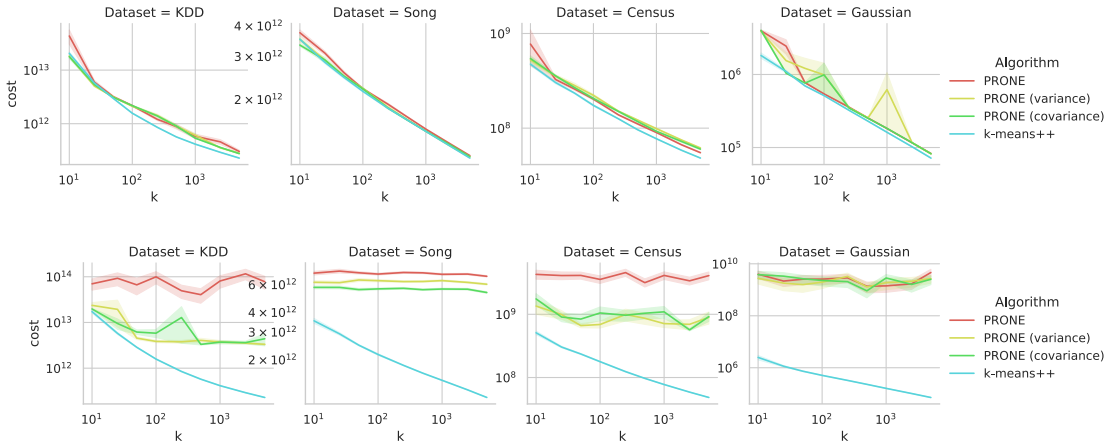

Figure 5.5: Clustering cost of all our variants compared to k-means++. The top row shows the k-means cost, and the bottom row shows the cost of the assignment produced by our algorithm.

On the real-world datasets, it is between one order of magnitude (for $k = 10$) and two orders of magnitude (for $k = 5000$) worse than k-means++ for our unbiased variant and between a factor 2 (for $k = 10$) and one order of magnitude (for $k = 5000$) worse than k-means++ for our biased and covariance variants.

Running time. Table 5.1 shows the relative running time of our algorithm compared to k-means++, assuming that no nearest-center assignment is computed. Our algorithms are designed to have a running time independent of k, so we can see, from the second table in Table 5.1, all of our variants offer significant speedups.

- The running time of our algorithm stays almost constant as k increases while the running time of k-means++ scales linearly with k. Specifically for $k = 25$, even our slowest variants have about the same running time as k-means++, while for $k = 5000$, it is at least **82x** faster, and our fastest version is up to **837x** faster over k-means++.

- The two variants can affect the quality of the chosen centers by up to an order of magnitude, but they are also significantly slower. The "variance" and "covariance" variants are slower (between 2-4x slower and up to 10x slower, respectively) than the standard variant, and they also become slower as the dimensionality d increases. We believe these methods could be further sped up, as the `blaze` library's variance computation routine appears inefficient for our use case.

5.7.3 Improved Approximation Ratio

Experimental Setup. This experiment aims to compare the algorithmic approach outlined in Corollary 5.3 to the direct use of `PRONE` as a clustering algorithm as was done

Dataset	k / Algorithm	10	100	1000
Census	Lightweight	7.3	69.4	670.2
	PRONE coreset	1.5	14.1	136.3
	Sensitivity	1.0	1.0	1.0
Song	Lightweight	8.9	87.2	875.3
	PRONE coreset	2.1	19.9	187.9
	Sensitivity	1.0	1.0	1.0
KDD	Lightweight	6.4	63.0	642.8
	PRONE coreset	2.1	19.6	192.6
	Sensitivity	1.0	1.0	1.0
Gaussian	Lightweight	2.4	17.6	174.8
	PRONE coreset	0.5	3.8	33.7
	Sensitivity	1.0	1.0	1.0

Dataset	k / Algorithm	50	500	5000
Census	PRONE	7.5	73.2	662.5
	PRONE (variance)	2.2	22.2	214.7
	PRONE (covariance)	1.1	10.7	117.4
	k-means++	1.0	1.0	1.0
Song	PRONE	9.7	95.5	837.5
	PRONE (variance)	2.3	23.1	217.2
	PRONE (covariance)	0.8	8.2	82.4
	k-means++	1.0	1.0	1.0
KDD	PRONE	6.9	68.3	727.5
	PRONE (variance)	3.1	32.0	312.4
	PRONE (covariance)	1.3	12.9	128.4
	k-means++	1.0	1.0	1.0
Gaussian	PRONE	1.9	18.3	165.9
	PRONE (variance)	2.0	17.7	162.9
	PRONE (covariance)	1.7	16.1	152.6
	k-means++	1.0	1.0	1.0

Table 5.1: Average speedup over sensitivity sampling across all relative sizes for constructing coresets (in the first table) and average speedup over k-means++ as a stand-alone clustering algorithm (in the second table). The tables with the full range of parameters can be found in the appendix.

in Section 5.7.2. For this, we use PRONE as the approximation algorithm for sensitivity sampling and then cluster the coreset using a weighted variant of the k-means++ algorithm. This approach is termed PRONE (boosted) in the rest of this section. This pipeline requires as parameters the number of centers k and a hyperparameter α indicating the size of the coreset produced by sensitivity sampling. We aim to compare the clustering cost (see Definition 5.6) and running time of our approach to that of k-means++.

We run both algorithms on the datasets described in Section 5.7 and choose $k \in \{10, 25, 50, 100, 250, 500, 1000, 2500, 5000\}$ and $\alpha \in \{0.001, 0.01, 0.1\}$. Each algorithm is run 5 times.

Results on Improved Approximation Ratio *Costs.* Figure 5.6 shows the costs of centers produced by this algorithm relative to the cost of centers produced by k-means++. It also contains the $(k, 2)$-clustering costs of PRONE relative to k-means++. We can see that on all real datasets, PRONE (boosted) produces solutions of the same or better quality than k-means++, as long as $\alpha n \ll k$. This shows that although PRONE by itself produces centers of worse quality, the PRONE (boosted) variant produces centers of the same quality as vanilla k-means++. When $\alpha n \approx k$, we observe an uptick in cost before the end of the lines corresponding to $\alpha \in \{0.1, 0.01\}$ in the plots for KDD, Song, and Gaussian. The boosted approach outperforms PRONE, which is usually worse by a constant factor compared to the other algorithms, and it helps to significantly reduce the amount of variance in the quality of solutions. On the Gaussian dataset, we observed a failure to sample a point from the central cluster, which explains the spike at $k = 2500$ for the line

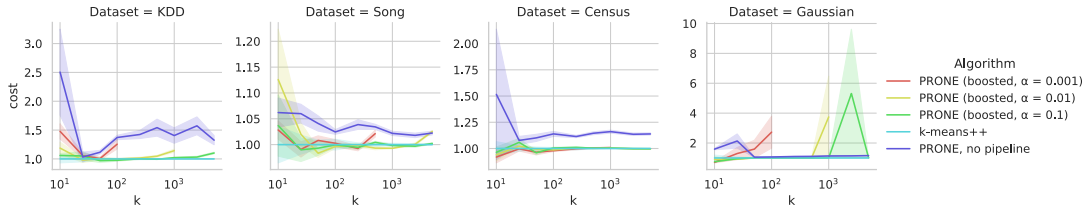

Figure 5.6: Clustering cost of the boosted variants compared to k-means++. Lines in the plot show the cost of centers produced by the boosted algorithm relative to k-means++ for centers ranging from 10 to 5000. Dark blue indicates the non-boosted version.

corresponding to $\alpha = 0.1$.

Running time. Table 5.2 shows the speedup of the boosted approach versus using plain k-means++, for the time taken to compute the centers. The running time of our algorithms now scales with k, but at a slower rate compared to k-means++, as we have to run it on a much smaller dataset. Once again, we observe significant speedups, especially as k grows.

- As expected, the speedup depends on the choice of the hyperparameter α. We observe diminishing returns for larger α as k scales, with the speedup remaining mostly constant for $k \geq 100$ across datasets, except for Gaussian. This is because the algorithm's running time is dominated by the time it takes to execute k-means++ on the coreset, which has $\mathcal{O}(ndk)$ asymptotic running time. The speedups we can achieve using this method are significant, up to 118x faster than k-means++. We expect that on massive datasets, even greater speedups can be achieved.

- Interestingly, the speedup can come very close to or even out scale α, as observed on the KDD and Song datasets. The final stage of the boosted approach executes k-means++ on a coreset of size αn, so the running time of this step should be $\mathcal{O}(\alpha ndk)$. The observed additional speedup may be due to better cache and memory utilization in the k-means++ step of the algorithm.

5.8 Conclusion and Limitations

To summarize, we present a simple algorithm that provides a new tradeoff between running time and approximation ratio. Our algorithm runs in expected time $\mathcal{O}(\mathsf{nnz}(\mathbf{X}) + n \log n)$ to produce a poly(k)-approximation; with additional poly$(kd) \cdot \log n$ time, we improve the approximation to $\mathcal{O}(\log k)$. This latter bound matches that of k-means++ but offers a significant speedup.

Within a pipeline for constructing coresets, our experiments show that the quality of the coreset produced (when using our algorithm as the initial approximation) outperforms the sensitivity sampling algorithm. It is slower than the lightweight coreset algorithm, but

Dataset	Centers Algorithm	10	25	50	100	250	500	1000	2500	5000
KDD	PRONE (boosted, $\alpha = 0.001$)	1.8	5.0	9.6	20.6	-	-	-	-	-
	PRONE (boosted, $\alpha = 0.01$)	1.9	4.8	9.0	17.5	37.1	59.2	92.5	-	-
	PRONE (boosted, $\alpha = 0.1$)	1.4	2.7	3.6	4.9	5.7	6.2	7.1	6.6	6.4
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Song	PRONE (boosted, $\alpha = 0.001$)	2.1	5.3	10.7	21.3	51.7	97.5	-	-	-
	PRONE (boosted, $\alpha = 0.01$)	2.1	5.0	9.8	18.5	38.0	59.1	81.5	108.3	117.0
	PRONE (boosted, $\alpha = 0.1$)	1.3	2.2	2.8	3.4	3.8	4.0	4.0	4.0	3.9
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Census	PRONE (boosted, $\alpha = 0.001$)	1.6	4.0	7.9	15.6	36.8	69.5	118.5	-	-
	PRONE (boosted, $\alpha = 0.01$)	1.5	3.7	6.7	11.7	20.8	28.5	30.1	36.2	41.9
	PRONE (boosted, $\alpha = 0.1$)	1.0	1.8	2.3	2.7	3.0	3.2	3.0	3.0	3.2
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Gaussian	PRONE (boosted, $\alpha = 0.001$)	0.5	0.9	1.9	3.6	-	-	-	-	-
	PRONE (boosted, $\alpha = 0.01$)	0.5	1.0	2.0	3.4	8.2	14.8	25.6	-	-
	PRONE (boosted, $\alpha = 0.1$)	0.4	0.8	1.4	2.2	4.0	5.0	6.1	6.8	7.2
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 5.2: Average speedup when computing a clustering and assignment for different datasets relative to k-means++. In other words, each cell contains $T_{k\text{-means++}}/T_{\texttt{PRONE}}$. Missing entries denote the case of $\alpha n > k$.

it is more "robust" as it is independent of the diameter of the dataset. It does not suffer from the drawback of having an additive error linear in the diameter of the dataset, which can arbitrarily increase the cost of the lightweight coreset algorithm. When computing an optimal assignment for the centers returned by our algorithm, its cost roughly matches the cost for k-means++. When directly using the assignment produced by one variant of our algorithm, its cost is between a factor 2 and 10 worse while being up to 300 times faster.

Our experiments and running time analysis show that our algorithm is very efficient. However, the clustering quality achieved by our algorithm is sometimes not as good as other, slower algorithms. We show that this limitation is insignificant when we use our algorithm to construct coresets. It remains an interesting open problem to understand the best clustering quality (e.g., in terms of approximation ratio) an algorithm can achieve while being as efficient as ours, i.e., running in time $\mathcal{O}(nd + n \log n)$. Another interesting problem is whether other means of projecting the dataset into a $\mathcal{O}(1)$ dimensional space exist, which lead to algorithms with improved approximation guarantees and running time faster than $\mathcal{O}(ndk)$.

5.9 Additional Data

In this section, we provide the running time data for the full range of parameters for the experiments performed in Section 5.7.2. Table 5.3 shows the speedups over k-means++, analogous to the right-hand-side table in Table 5.1. Additionally, Table 5.4 provides

absolute running times in milliseconds.

Dataset	Centers Algorithm	10	25	50	100	250	500	1000	2500	5000
Census	PRONE	1.5	3.8	7.5	15.1	36.2	73.2	142.2	351.9	662.5
	PRONE (variance)	0.5	1.1	2.2	4.6	11.0	22.2	43.7	109.5	214.7
	PRONE (covariance)	0.2	0.5	1.1	2.2	5.2	10.7	21.0	54.7	117.4
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Song	PRONE	2.0	5.0	9.7	19.1	46.1	95.5	188.2	443.0	837.5
	PRONE (variance)	0.5	1.1	2.3	4.5	11.4	23.1	44.2	110.9	217.2
	PRONE (covariance)	0.2	0.4	0.8	1.5	4.0	8.2	15.5	40.0	82.4
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
KDD	PRONE	1.5	3.7	6.9	16.3	39.5	68.3	158.5	414.7	727.5
	PRONE (variance)	0.7	1.7	3.1	6.3	16.1	32.0	63.4	159.6	312.4
	PRONE (covariance)	0.3	0.7	1.3	2.6	6.8	12.9	25.8	58.5	128.4
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Gaussian	PRONE	0.5	1.0	1.9	3.8	9.2	18.3	35.7	85.3	165.9
	PRONE (variance)	0.5	1.0	2.0	3.8	9.1	17.7	34.5	83.6	162.9
	PRONE (covariance)	0.4	1.0	1.7	3.6	8.2	16.1	31.8	79.9	152.6
	k-means++	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 5.3: Average speedup when computing a clustering and assignment for different datasets relative to k-means++. In other words, each cell contains $T_{k\text{-means++}}/T_{\texttt{PRONE}}$.

Dataset	Centers / Algorithm	10	25	50	100	250	500	1000	2500	5000
Census	PRONE	525.1 ± 15.3	534.3 ± 23.4	534.6 ± 20.8	530.1 ± 15.4	538.9 ± 15.2	533.9 ± 11.7	547.4 ± 23.8	548.2 ± 13.5	563.7 ± 8.7
	PRONE (variance)	1750.6 ± 71.7	1778.9 ± 38.1	1780.3 ± 23.9	1752.7 ± 56.9	1768.6 ± 28.2	1757.6 ± 37.8	1778.9 ± 27.5	1761.0 ± 73.3	1739.6 ± 10.9
	PRONE (covariance)	3769.0 ± 152.2	3882.5 ± 167.8	3743.9 ± 316.7	3714.2 ± 378.3	3766.4 ± 220.2	3662.9 ± 122.6	3708.7 ± 372.4	3525.6 ± 609.9	3182.2 ± 288.7
	k-means++	812.5 ± 20.9	2040.6 ± 25.6	3994.3 ± 25.3	7992.9 ± 91.5	19519.0 ± 161.5	39058.4 ± 337.8	77823.6 ± 527.5	192913.1 ± 3657.0	373488.2 ± 221.9
Song	PRONE	104.4 ± 5.5	101.8 ± 5.2	106.5 ± 4.1	109.9 ± 6.8	113.7 ± 8.8	109.0 ± 6.4	108.7 ± 3.9	117.2 ± 6.3	120.8 ± 12.9
	PRONE (variance)	443.3 ± 5.0	448.3 ± 8.0	450.1 ± 9.9	468.0 ± 31.6	458.8 ± 15.1	450.7 ± 10.0	462.6 ± 11.6	468.3 ± 14.0	465.7 ± 14.9
	PRONE (covariance)	1164.4 ± 162.1	1266.5 ± 136.6	1332.4 ± 116.3	1381.8 ± 108.4	1322.2 ± 173.4	1263.0 ± 175.6	1324.2 ± 108.4	1296.8 ± 177.8	1228.0 ± 175.2
	k-means++	207.4 ± 4.0	513.7 ± 9.8	1036.6 ± 17.0	2103.9 ± 44.2	5245.9 ± 143.6	10412.5 ± 119.3	20464.1 ± 196.9	51917.4 ± 554.3	101149.0 ± 1962.7
KDD	PRONE	31.2 ± 5.2	32.2 ± 7.4	34.1 ± 1.8	28.9 ± 4.9	29.2 ± 9.1	34.6 ± 6.3	30.8 ± 8.6	28.8 ± 5.2	33.4 ± 6.6
	PRONE (variance)	72.0 ± 1.4	71.8 ± 1.4	76.5 ± 7.2	75.1 ± 7.1	71.8 ± 1.2	73.9 ± 2.1	77.0 ± 4.6	74.7 ± 3.2	77.7 ± 5.9
	PRONE (covariance)	169.3 ± 2.9	173.5 ± 5.5	180.3 ± 13.1	184.1 ± 21.2	170.8 ± 8.9	182.6 ± 19.0	188.7 ± 16.9	204.1 ± 40.3	189.1 ± 11.6
	k-means++	48.0 ± 0.3	119.3 ± 3.7	233.9 ± 3.2	470.4 ± 11.6	1153.5 ± 2.1	2363.3 ± 108.6	4878.1 ± 171.1	11928.9 ± 234.3	24285.9 ± 243.8
Gaussian	PRONE	30.0 ± 2.3	29.1 ± 2.2	30.6 ± 3.3	29.1 ± 1.4	29.1 ± 1.0	28.7 ± 0.2	29.3 ± 0.4	30.2 ± 0.6	31.5 ± 0.6
	PRONE (variance)	29.8 ± 3.0	28.9 ± 0.3	29.5 ± 1.0	29.6 ± 0.6	29.6 ± 0.3	29.6 ± 0.4	30.2 ± 0.3	30.8 ± 0.2	32.1 ± 0.6
	PRONE (covariance)	31.8 ± 2.2	31.5 ± 2.1	34.6 ± 6.6	30.8 ± 0.3	32.7 ± 2.8	32.6 ± 2.4	32.8 ± 2.3	32.2 ± 0.5	34.3 ± 0.4
	k-means++	13.7 ± 2.2	30.0 ± 0.4	59.5 ± 2.2	111.5 ± 2.8	268.6 ± 1.5	525.3 ± 1.9	1043.3 ± 1.4	2575.3 ± 5.0	5229.6 ± 135.0

Table 5.4: Average running time and standard deviation in milliseconds when computing a clustering and assignment for different datasets relative to k-means++.

6 Expander Hierarchies for Normalized Cuts on Graphs

Expander decompositions of graphs have significantly advanced the understanding of many classical graph problems and led to numerous fundamental theoretical results. However, their adoption in practice has been hindered due to their inherent intricacies and large hidden factors in their asymptotic running times. Here, we introduce the first practically efficient algorithm for computing expander decompositions and their hierarchies and demonstrate its effectiveness and utility by incorporating it as the core component in a novel solver for the normalized cut graph clustering objective.

Our extensive experiments on a variety of large graphs show that our expander-based algorithm outperforms state-of-the-art solvers for normalized cut with respect to solution quality by a large margin on a variety of graph classes such as citation, e-mail, and social networks or web graphs while remaining competitive in running time.

6.1 Introduction

In recent years, expander decompositions and expander hierarchies have emerged as fundamental tools within the theory community for developing fast approximation algorithms and fast dynamic algorithms for such diverse problems as, e.g., routing [145], connectivity [139], (approximate) maximum flow [191, 72], or triangle enumeration [67]. Informally, an expander is a well-connected graph. Its *expansion* or *conductance* $\phi = \min_{S \subseteq V} \frac{\text{border}(S)}{\min(\text{vol}(S),\text{vol}(\bar{S}))}$ is the minimum ratio between the number of edges leaving any subset of vertices and the number of edges incident to vertices of the subset. A large value of ϕ (close to 1) indicates a graph in which no subset of vertices can be easily disconnected from the rest of the graph.

An *expander decomposition* of a graph partitions the vertices such that the subgraph induced by each part is an expander and the number of edges between different components of the partition is low.[1] Many results over the years [299, 322, 237, 238, 278] have demonstrated that such a decomposition not only exists for every graph, but can be computed in near-linear time. At a high level, the success of expander decomposition-based algorithms is due to the fact that many problems are 'easy' on expanders: One first identifies regions of a graph where a problem is easy to solve (the expanders), solves the problem (or a suitable sub-problem) within each region, and then combines the individual solutions to obtain a solution for the overall problem.

[1] This can be seen as a special form of a graph clustering.

Expander hierarchies Goranci et al. [139] apply expander decompositions in a hierarchical manner. We do this by computing an expander decomposition of the graph and then contracting the individual expanders into single vertices repeatedly. This process is then repeated until the entire graph has been contracted to a single vertex. From this recursive decomposition we obtain a corresponding decomposition tree T (the so-called *(flow) sparsifier*) where the root corresponds to the entire graph, the leaves correspond to vertices of the original graph, and inner vertices correspond to the expanders found during the decomposition procedure. The sparsifier approximately preserves the cut-flow structure of the original graph in a rigorous sense. This relationship has been exploited in numerous applications in static and dynamic graph problems [161, 207, 145] to obtain fast algorithms with provable guarantees.

While many algorithms for expander decomposition that offer strong asymptotic bounds on the running time have been suggested [299, 322, 237, 238, 278], [2] practical algorithms based on expander decomposition have not seen success thus far. This is due to these algorithms requiring us to solve a number of maximum flow problems for each node in their recursion tree, which results in these algorithms being prohibitively slow in practice for graphs with many edges. Arvestad [21], e.g., reports that decomposing a graph with approximately 10^5 edges can take almost 5 min for various values of ϕ in their implementation of Saranurak and Wang [278].

In practice, the *multilevel graph partitioning* framework has become the de-facto approach for computing high quality solutions for cut problems on graphs. The framework consists of a *coarsening* step, in which a smaller representation of the graph is computed, a *solving* step, where a solution is computed on the coarsened graph and a *refinement* step, during which the solution is improved using local heuristics during the graph uncoarsening process. Multilevel graph partitioning has been successfully applied for computing balanced cuts [186, 277, 250, 22] and normalized cuts in graphs [107].

We note that algorithms based on the expander hierarchy approach may also be regarded as a variant of multilevel graph partitioning. In this approach, a graph is first coarsened by recursive contractions to obtain a smaller graph that reflects the same basic cluster structure as the input graph. Then an initial partition is computed on the sparsifier and, afterwards, we can further improve the solution quality by employing heuristic improvements to the solution as we undo the coarsening. This refinement step has not been considered in the theoretical applications of expander decomposition, but leads to a considerable further improvement of solution quality in practice. Despite the strong theoretical guarantees brought forward by expander decompositions and expander hierarchies, the latter have not yet led to similar breakthroughs in the field of fast, high-quality experimental algorithms. The reason is that the expander decomposition required at each coarsening step becomes a significant performance bottleneck in practice (see also the earlier discussion). In contrast, solvers based on the multilevel graph partitioning paradigm such as METIS [186] and KaHiP [277] only compute maximal matchings[3] for

[2] Including the near-linear-time algorithm by Saranurak and Wang [278], which finds a decomposition where each component has expansion at least ϕ in time $\mathcal{O}\left(m\phi^{-1}\log^4 m\right)$.

[3] A matching is a set of edges M such that each vertex is incident to at most one edge in M.

each graph in the hierarchy, which is considerably faster than computing an expander decomposition.

In this work we mitigate the computational bottleneck by introducing a novel, practically efficient random-walk-based algorithm for expander decomposition. Based on this, we give the first implementation of the expander hierarchy and, thus, an algorithm to compute tree flow sparsifiers, allowing us to solve various cut problems on graphs effectively. Additionally, we show that our approach is eminently suitable to compute normalized k-cuts on graphs, where the goal is to partition the vertex set into k clusters such that the sum over the number of edges leaving each cluster, normalized by the cluster's volume, is minimized. Normalized cut is a popular graph clustering objective and particularly able to capture imbalanced clusterings. Its manifold applications include, e.g., community detection [208] and mining [59], topic reconstruction [45], story segmentation [334], bioinformatics [107, 323], tumor localization [273], and image segmentation [290, 318]. It is closely related to spectral clustering, which uses spectral properties of eigenvalues and eigenvectors of the graph's Laplacian. However, the spectral approach suffers from both very large running times and large memory requirements to compute and store the eigenvectors, and thus does not scale well [107]. This problem was addressed by Dhillon et al. [107] as well as Zhao et al. [337], who presented algorithms for normalized cut that either use spectral methods only after coarsening [107] or apply them on a spectrally sparsified graph [337].

Contributions. We present the first practically efficient algorithm for expander decomposition and the first implementation to compute an expander hierarchy. Our approach is based on random walks and is justified by rigorous theoretic and empirical analysis. We report on a comprehensive experimental study on normalized cut solvers, comprising 50 medium-sized to very large graphs of various types, where we compare our expander-based algorithm, XCut, to Graclus by Dhillon et al. [107], the approach by Zhao et al. [337], as well as the state-of-the-art graph partitioners METIS and KaHiP, which do not specifically optimize towards the normalized cut objective, but are fast and in practice often used for this task. The experiments show that our algorithm produces superior normalized cuts on graph classes such as citation, e-mail, and social networks, web graphs, and generally scale-free graphs, and is only slightly worse on others. On average, it is still distinctly the best across all graphs and values of k.

If only a single value of k is desired, its running time is on average only 3 times slower than the runner-up, Graclus, and never exceeded 18 min. A notable advantage of XCut is that it can quickly compute solutions for multiple values of k once a sparsifier is computed, which can be faster than running Graclus multiple times.

6.2 Related Work

Our work is motivated by recent theoretical results [145, 139, 191, 72, 67] building on expander decompositions [299, 322, 237, 238, 278] and the expander hierarchy [139] and inspired by non-spectral approaches [107] to tackle the normalized cut problem.

Mohar [233] showed that the computation of the so-called isoperimetric number or conductance of a graph (see Section 6.3) is NP-hard, which implies the hardness of the normalized k-cut problem for $k \geq 2$. Normalized cut remains NP-hard on weighted trees [97].

A number of tools that have been used to solve normalized cut use spectral methods [244, 329, 240, 260, 74, 247]. This usually requires to compute k eigenvectors of the Laplacian matrix, which was shown to scale badly in practice [107, 337]. Afterwards, an additional discretization step is necessary to obtain the clustering.

Dhillon et al. [107] therefore suggest an algorithm called Graclus, which is based on the multilevel graph partitioning framework. It applies the same coarsening steps as METIS, but with a modified matching procedure. The coarsened graph can then be partitioned using different approaches, including a spectral one. In the refinement step, Graclus uses a kernel k-means-based local search algorithm for improving the normalized cut objective value. The authors evaluate their algorithm experimentally against METIS as well as a spectral clustering algorithm [329]. They show that it outperforms the spectral method w.r.t. normalized cut value, running time, and memory usage. It also produces better results than METIS and is comparable w.r.t. running time.

Zhao et al. [337] employ a joint spectral sparsification and coarsening scheme to produce a smaller representation of the graph that preserves the eigenvectors of the Laplacian in near-linear time $\tilde{\mathcal{O}}(m)$. Afterwards, a normalized cut is computed on the sparsifier using spectral clustering. Their sparsification scheme obtains a significant reduction in the number of edges and nodes, which makes it feasible to apply the spectral method on the reduced graphs, without the quadratic term in the running time growing too large. The authors again perform an experimental comparison with METIS and observe that their algorithm overall outperforms METIS w.r.t. the normalized cut value, while being slightly slower.

METIS [186], KaHiP [277], and many other graph partitioning tools [314, 156] do not solve the normalized cut problem and instead aim to find good solutions to a balanced graph partitioning problem, where the vertex set is to be partitioned into k sets of (roughly) equal size, while minimizing the number of edges cut. CHACO [156] implements a spectral graph partitioning approach, but the number of clusters is limited to at most $k = 8$.

Nie et al. [247] recently presented a spectral normalized cut solver based on the coordinate descent method along with several speedup strategies. They evaluate their algorithm against two other spectral methods [244, 74] on a number of medium-sized datasets with a known ground truth and show that it consistently computes the best solution for recovering the data labels and also requires less time to run. A notable difference in their algorithm is that k is not an input parameter, but rather gets chosen using an auto-tuning routine that attempts to guess the number of clusters in the ground truth dataset.

6.3 Preliminaries

For an undirected graph $G = (V, E)$ we use d_v to denote the degree of vertex $v \in V$, $\vec{d}$ to denote the *degree vector*, i.e., the vector of vertex degrees, and $D = I\vec{d}$ to denote the corresponding *degree* matrix. Δ is used to denote the maximum degree of a vertex in G. For a subset S of vertices we define the *volume* $\mathrm{vol}(S)$ as the sum of vertex degrees of vertices in S. Its *border* $\mathrm{border}(S)$ (or *capacity*) is defined as $|E(S, \bar{S})|$, where $E(X, Y) = \{\{x, y\} \in E \mid x \in X, y \in Y\}$ is the set of edges between subsets $X, Y \subseteq V$, and $\bar{S} = V \setminus S$.

A *k-cut* is a partition $\mathcal{P}$ of the vertex set into k non-empty parts. Given such a partition $\mathcal{P} = (S_1, \ldots, S_k)$, its *normalized cut value* θ is defined as

$$\theta(\mathcal{P}) = \sum_{i=1}^{k} \frac{\mathrm{border}(S_i)}{\mathrm{vol}(S_i)} \quad .$$

For $k = 2$ we will usually specify a 2-cut (or just cut) by its smaller side, i.e., we will refer to the cut $(S, \bar{S})$ by just S, where $\mathrm{vol}(S) \leq \mathrm{vol}(\bar{S})$. The *conductance* of a (2-)cut is defined as $\Phi(S) = \mathrm{border}(S)/\mathrm{vol}(S)$ and the *conductance of a graph* $G = (V, E)$ is $\Phi(G) = \min_{\text{cuts } S} \Phi(S)$.[4] The problem of finding a 2-cut with minimum conductance is closely related to the problem of finding a 2-cut with minimum normalized cut value, because for any 2-cut, $\Phi(S) \leq \theta(S) \leq 2\Phi(S)$. The normalized k-cut objective can be seen as a generalization of conductance to k-cuts.

To properly describe our random walks we need some further notation. For a cut S we call $\mathrm{vol}(S)/\mathrm{vol}(V)$ the *balance* of the cut S and denote it with $b(S)$. For a subset $A \subseteq V$ we use $G\{A\}$ to denote the subgraph induced by A with self-loops added so that the vertex degrees do not change (a self-loop counts 1 to the degree of a vertex). This means the degree of a vertex $a \in A$ is the same in G as in $G\{A\}$. Whenever the graph in question is not clear from the context we use subscripts to indicate the graph, i.e., we write $\mathrm{vol}_G(S)$, $\Phi_G(S)$, $E_G(X, Y)$, etc. To avoid notational clutter we write $\mathrm{vol}_A(S)$, $\Phi_A(S)$, $E_A(X, Y)$ when we are referring to the graph $G\{A\}$.

We say a graph $G = (V, E)$ is a *ϕ-expander* if $\Phi(G) \geq \phi$, and we call a vertex partition $(V_1, \ldots, V_\ell)$ a *ϕ-expander decomposition* if $\Phi(G\{V_i\}) \geq \phi$ for all i.

6.4 Expander Decomposition using Random Walks

In this section, we outline the theoretical foundations of XCut, namely the fast, flow-free expander decomposition routine which lies at the core of XCut. We only give a brief overview and state the main theorems in this chapter, for the complete description and proof details refer to the full version of the paper [148].

Our approach follows the usual meta-algorithm for expander decomposition. That is, given a target expansion ϕ, we check whether the graph contains a cut with conductance smaller than ϕ, and if so, we partition the graph into two parts, removing the cut edges.

[4] If there are no cuts in G (i.e., $|V| = 1$) we define its conductance to be 1.

Algorithm 19: Cut Procedure

Input: Graph $G = (V, E)$, Target expansion ϕ
Output: Expander or Balanced$(S, \bar{S})$ or Unbalanced$(S, \bar{S})$

$T \leftarrow 1/(12\phi)$;
$\gamma \leftarrow 343\sqrt{\phi \log(32m^3)\log^2(n)}$; // cut threshold
$\beta \leftarrow 2/\log^2 m$; // balance
$\mathbf{W} \leftarrow \mathbf{I}$; // random walk matrix
$A \leftarrow V$, $L \leftarrow \emptyset$;
for $t = 1, \ldots, T$ **do**
  $\mathbf{r} \leftarrow$ random unit vector in $\mathbb{R}^n$;
  $\mathbf{u} \leftarrow \mathbf{W}\mathbf{D}^{-1}\mathbf{r}$; // apply random walk
  $\mathbf{u} \leftarrow \mathbf{u} - \langle \mathbf{u}, \mathbf{d} \rangle / \operatorname{vol}(V) \cdot \mathbf{1}$; // ensure $\mathbf{u} \perp \mathbf{d}$
  $\mathbf{A}$, $\mathbf{D}_A \leftarrow$ adjacency and degree matrix of $G\{A\}$;
  $\mathbf{W} \leftarrow \left(\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{A}\mathbf{D}_A^{-1}\right)\mathbf{W}$; // extend walk matrix
  $S \leftarrow \text{SweepCut}(\mathbf{u}, \gamma)$;
  if $\operatorname{vol}(S) \geq \beta m$ **then**
    return Balanced$(S, \bar{S})$;
  else if $S \neq \emptyset$ **then**
    $L \leftarrow L \cup S$, $A \leftarrow A \setminus S$;
    if $\operatorname{vol}(L) \geq \beta m$ **then**
      return Balanced(A, L);
if $L = \emptyset$ **then**
  return Expander;
else
  return Unbalanced(A, L);

If we perform this until no more cuts are found, then we know that each remaining partition will be a ϕ expander. The challenge with expander decomposition algorithms is to control the recursion depth and therefore the running time of the algorithm, as well as the maximum number of edges being cut. Assuming all the sparse cuts are balanced, say each part contains some logarithmic fraction of the volume, then the recursion depth of the algorithm is $\mathcal{O}(\log m)$. One core observation which is used to control the running time if the cuts are unbalanced is that if only unbalanced cuts are found, the graph likely contains a large ϕ-expander.

Saranurak and Wang [278] employ this framework with two core components, the cut-matching game, which is used for identifying the sparse cuts, and a trimming step, which is used to find the large ϕ-expander inside a so-called *near ϕ-expander* which is obtained after a number of unbalanced cuts has been made.

As stated before, while this framework gives very favorable running time guarantees of $\tilde{\mathcal{O}}(m/\phi)$ and cuts $\tilde{\mathcal{O}}(\phi m)$ edges, the cut-matching game necessitates us to solve a logarithmic number of maxmimum-flow problems on our graph. Unfortunately, maintaining the data structure for this problem is too slow for large graph instances in practice, even when reusing the flow information using a modified Push-Relabel algorithm [159].

This motivates us to instead use random walks for the theoretical algorithm. A random walk iteration can be performed in time $\mathcal{O}(m)$ and is efficiently implementable, using either modern linear algebra packages for sparse matrix multiplication or a custom data structure. Our approach gives worse theoretical guarantees, due to the Cheeger barrier, which means that we cut $\sqrt{\phi}$ rather than ϕ edges. However, for our practical algorithm, where ϕ is chosen to be a large constant close to 1, this does not pose a problem.

We essentially modify the algorithm of Saranurak and Wang, keeping the same trimming step, but replacing the cut-matching game by Algorithm 19. Using the random walk lets us exploit the expander-mixing lemma, which states that the rate of convergence of the random walk is governed by the magnitude of the second eigenvalue λ_2 of the graph's lazy random walk matrix. Furthermore, this eigenvalue is also related to sparse cuts in the graph via the Cheeger inequality.

This tells us that if the graph is an expander, our walk converges quickly, and we are unlikely to find a low conductance cut. On the other hand, if there is a sparse cut, convergence takes more iterations and we are likely to identify a sparse cut, as we are approximating the eigenvector via power-iteration of the random walk matrix. In order to round our approximate eigenvector to a low conductance cut, we use the well known sweep cut scheme, see Algorithm 20, which takes time $\mathcal{O}(m)$. Our main theorem, which establishes the guarantees of Algorithm 19 is as follows:

Theorem 6.1 (Cut Procedure)**.** *Given a graph $G = (V, E)$ with m edges and a parameter ϕ, the cut procedure takes $\mathcal{O}((m + n \log n)/\phi)$ steps and terminates with one of these three cases:*

1. *We certify that G has conductance $\Phi(G) \geq \phi$.*
2. *We find a cut $(A, \bar{A})$ in G that has conductance at most $\Phi_G(A) \in \mathcal{O}\left(\sqrt{\phi} \log^{3/2} m\right)$. Then one of the following holds:*

a) either $\operatorname{vol}(A), \operatorname{vol}\bar{A}$ *are both* $\Omega\left(m/\log^2 m\right)$*, i.e.,* $(A, \bar{A})$ *is a relatively balanced low conductance cut;*

b) or $\operatorname{vol}\bar{A} \in \mathcal{O}\left(m/\log^2 m\right)$ *and* A *is a near* 6ϕ*-expander.*

Using this cut procedure in place of the cut matching game in the framework of Saranurak and Wang [278], we obtain an expander decomposition algorithm running in near linear time $\tilde{\mathcal{O}}\left(m/\phi\right)$. We use their trimming step in order to limit the recursion depth.

More precisely, we obtain the following theoretical guarantees:

Theorem 6.2 (Expander Decomposition)**.** *Given a graph* G *with* m *edges and a parameter* ϕ*, there is a random-walk-based algorithm that with high probability finds a* ϕ*-expander decomposition of* G *and cuts at most* $\mathcal{O}\left(\sqrt{\phi}\, m \log^{5/2} m\right)$ *edges. The running time is* $\mathcal{O}\left(\frac{m+n\log n}{\phi}\log^3 m\right)$.

6.5 XCut – A New Normalized Cut Algorithm

In this section, we introduce the algorithm XCut, which is based on the previous section's novel random walk-based expander decomposition. We note the apparent similarity between multilevel graph partitioning and the expander hierarchy and use this as the basis of XCut. As an outline, we use the novel random walks to construct the expander hierarchy to obtain a coarse representation of the graph, the tree flow sparsifier. We then compute an initial solution on the tree. Finally, we use an iterative refinement step while descending the hierarchy to improve the solution we found. Compared to other contraction schemes, which lead to each vertex in the coarsest graph representing roughly the same number of nodes in the base graph, the subtrees on each level in the tree flow sparsifier can represent a vastly different number of vertices.

Expander Decomposition. While the expander decomposition outlined in Section 6.4 is much simpler to implement than that of Saranurak and Wang [278], as we do not rely on maximum flow computations at all, we made several choices in the implementation to speed up computation. We iterate a single random walk, and after each iteration, we check whether we can find a sparse cut. If we find a suitable cut, we disconnect the edges going across it, but contrary to the algorithm in Section 6.4, we do not restart the random walk, as we find we can extract further information about the cut structure of the graph from the state of the random walk. For example, if the random walk has mixed very well on one of the new components, it is likely to be an expander, while if there is another sparse cut in the component, then the random walk will likely not have mixed well on the component. One may think that reducing the number of random walks might lead to a loss of guarantees and higher variance of the algorithm, but in experiments conducted while designing the algorithm, we found that on real graph instances running multiple concurrent random walks is not necessary. In fact, only a single graph in our 50 graph benchmark had noticeable variance. See also the discussion in Section 6.6.3.

The main parameter of the expander decomposition is the cut value γ, which is the minimal sparsity of the cuts our algorithm makes. This parameter is used as the cut threshold for the sweep cut routine outlined in Algorithm 20. Additionally, we introduce a parameter ρ, to be used as a threshold for "certifying" that a component is an expander, as we found that choosing the threshold to be $1/(4\,\mathrm{vol}(V)^2)$ does not offer any benefits over a much larger value. See also Section 6.6.2 for details on choosing the value for this parameter. We make a final modification to the theoretical algorithm in that we omit the trimming step on unbalanced cuts, since it does not provide any further speedup of the expander decomposition routine in practice. That is to say, we return $\mathsf{Balanced}(A, L)$ if we reach line 22 in Algorithm 19.

Algorithm 20: Sweep cut routine

Input: Vector of node values $\mathbf{u}$, Cut threshold γ, Graph $G = (V, E)$
Output: $S \subset V$ such that $\frac{\mathrm{border}(S)}{\min(\mathrm{vol}(S),\mathrm{vol}(V\setminus S))} \leq \gamma$ or $\emptyset$ if no sparse cut was found

1 $S \leftarrow \emptyset$;
2 Let $\mathbf{v}$ be the vector of vertices $(v_1, v_2, \ldots, v_n)$ sorted in ascending order according to the values of v_i in $\mathbf{u}$;
3 **for** $v \in \mathbf{v}$ **do**
4 $S \leftarrow S \cup \{v\}$;
5 **if** $\frac{\mathrm{border}(S)}{\min(\mathrm{vol}(S),m-\mathrm{vol}(S))} \leq \gamma$ **then**
6 **return** the smaller of S or $V \setminus S$;
7 **return** $\emptyset$;

Automatically Choosing γ. The theoretical analysis of Goranci et al. [139] suggests a choice for γ and ϕ that is sufficient to prove the theoretical results. In preliminary experiments we found this choice to be too pessimistic and, in fact, by adapting ϕ and γ to the graph we can obtain better results. However, there is a trade-off to be made: If γ is too small, the expander decomposition will not find many sparse cuts, and we obtain sparsifiers of low quality. On the other hand, if γ is too large, most cuts will be sparser than γ, which leads to many cuts being made and increases the running time. In the worst case, this can even prevent the algorithm from terminating.

Thus, our goal is to choose γ such that it offers a good quality vs. time trade-off. When choosing γ, we can only observe whether this was a good choice in a post hoc fashion. A naive strategy would be to start with a large γ and decrease it until the expander hierarchy terminates within a reasonable amount of time. However, this approach is wasteful, as we discard previously computed decompositions, even if they were good. Instead, we decrease γ by multiplying it with constant factor $\varepsilon < 1$ whenever the expander decomposition on a specific level cuts too many edges in G_i. We then use the new $\gamma' = \epsilon\gamma$ for the remaining expander decompositions, decreasing it further as required.

Solving on the Sparsifier. To solve normalized k-cut on the tree sparsifier obtained from the hierarchy, we want to remove $k-1$ edges to decompose it into a forest of k trees. Given a solution, i.e., a tuple of edges $(e_1, \ldots, e_{k-1})$, we assign vertex $v \in V$ to the cluster C_j associated with e_j if e_j is the first edge we encounter on the path from v to the root. If no edge in the solution lies on the path to the root, we assign v to cluster C_k. As normalized cut is NP-hard also on trees (see Section 6.2), we introduce two heuristic approaches that take time $\mathcal{O}(nk)$ each:

Greedy: The simple greedy heuristic picks the edge in the sparsifier which minimizes the increase in the normalized cut objective in every step. By simply computing the cost of cutting each remaining edge, it takes $\mathcal{O}(n)$ time to find this edge, assuming the number of vertices in the sparsifier is $\mathcal{O}(n)$. To partition a graph into k clusters, we repeat this process $k-1$ times. For $k=2$, this algorithm produces the optimal solution on the tree as we pick the edge that minimizes the cut objective.

Dynamic Programming: Each row of the dynamic program corresponds to a level, and we make one cell for each pair (v, i) of the row, where v is a vertex in the sparsifier and $i \in [0, k]$. The value of each cell is the normalized cut value θ of decomposing the subtree rooted at v into i parts. We write $DP(v, i)$ for this value. Additionally, we write $\text{cut}(v, i)$ for the weight of cut edges in the solution incident to the subtree rooted at v, as well as $\text{vol}(v, i)$ for the volume remaining in the subtree.

Without loss of generality, assume the tree is binary, as otherwise we can binarize the tree by inserting edges of infinite cost (see Henzinger et al. [157] for details). The value of a cell is computed according to the following rule:

$$\begin{aligned}\text{DP}(v, j) = \min(\,&\text{cutParent}(\text{DP}(v, j-1)),\\ &\min_{0 \le i \le j}(\text{DP}(v_l, i) + \text{DP}(v_r, j-i))),\end{aligned}$$

where $\text{cutParent}(\text{DP}(v, j-1)) = \text{DP}(v, j-1) + \frac{w(v,v') + \text{cut}(v, j-1)}{\text{vol}(v, j-1)}$ is the best solution where the edge going to the parent is cut. For each vertex v of G_0, we initialize the bottom row of the program with $\text{DP}(v, 0) = 0$, $\text{DP}(v, 1) = 1$ and $\text{DP}(v, j) = \infty$ for $j \in [2, k]$. In the root vertex we use the special rule $\text{DP}(v, j) = \min_i \text{DP}(v_l, i) + \text{DP}(v_r, j-i) + \frac{\text{cut}(v_l, i) + \text{cut}(v_r, j-i)}{\text{vol}(v_l, i) + \text{vol}(v_r, j-i)}$, where the last term ensures we do not produce a solution with $\text{vol}(v, i) = 0$, leading to cluster j being empty.

The Refinement Step. Finally, we perform an iterative refinement as we descend the hierarchy. While descending, we introduce new clusters according to the edges in the solution. On each level we then perform vertex swaps that improve the normalized cut objective, by either reducing the number of cut edges or making the partition more balanced. For details see Section 6.9.

Support for Variable Number of Partitions k. A notable strength of our algorithm is that with both heuristics, it solves the problem for any value $k' \le k$ during their execution.

Suppose we are still determining the number of clusters needed. In that case, we can compute the solution for the maximum k we are interested in and obtain solutions for all smaller numbers of partitions during exploratory data analysis. The only step that needs to be rerun is the refinement step.

6.6 Experimental Evaluation

In the previous sections, we have shown that our approach provides provable guarantees on the approximation ratio for the value θ of the normalized cut[5] (and its relatives, sparse cut, and low-conductance cut) if $k = 2$. We now turn to evaluate XCut experimentally in different configurations. We compare the objective value of normalized cuts produced by XCut and its running time against the normalized cut solver Graclus by Dhillon et al. [107] and the state-of-the-art graph partitioning packages METIS [186] and KaHiP (kaffpa) [277], all of which are available publicly. These algorithms are based on the multilevel graph partitioning framework and produce disjoint partitions of the vertices into k clusters, where k is a freely choosable parameter. We note that METIS and KaHiP solve the balanced k-partitioning problem rather than normalized cut. Nevertheless, we include these solvers in our comparison, as they are used for this task in practice and we found that they can outperform Graclus on some of the graphs in our benchmark dataset. By this, we follow the methodology of [107] and [337].

In addition, we compare our results to the values reported for the normalized cut algorithm by Zhao et al. [337][6]. We omit comparisons to solvers employing spectral methods such as the recent works by Chen et al. [74] and Nie et al. [247], as this approach does not scale well to large datasets of millions of nodes [107]. In preliminary experiments we found that the solver of Nie et al. [247] uses over 330GB of memory on instance CN3, whereas our solver used less than 400MB of memory. Furthermore, the algorithm presented in [247] does not necessarily produce k clusters, thus making direct comparisons difficult. Lastly, the space complexity of spectral methods becomes prohibitive for larger values of k, as noted by Dhillon et al. [107].

6.6.1 Experimental Setup

Instances. Our setup includes 50 graphs from various applications. See Table 6.1 for an overview and Table 6.3 for details. To facilitate comparability, our collection contains the eight instances used by Dhillon et al. [107] (BP1, CF1, CS1–3, DM1, OP1, and OP2) as well as the 21 instances used by Zhao et al. [337]. In addition, we selected 21 real-world networks that cover various application areas, including some of larger sizes. All instances are available publicly in the Network Repository [270] or the SuiteSparse Matrix collection [99].

Note that a graph with k or more connected components always has a (normalized) k-cut of size 0. Surprisingly, we found that XCut is the only solver tested here that

[5] See Section 6.3 for the precise definition.
[6] Unfortunately, the code is not available publicly.

finds the trivial optimal solution if k is less than the number of connected components. However, testing connectivity before starting a solver remedies this problem, which is why we decided to exclude graphs with more than 128 connected components except for one (graph BP1), which we keep for consistency reasons as it was used in previous comparisons [107].

Methodology. As XCut, KaHiP, and METIS are randomized, we ran each of them $\ell = 10$ times per instance with different seeds. Graclus is deterministic, and we ran it three times to obtain a stable value for the running time. We use the arithmetic mean over the ℓ runs for each instance to approximate the expected value of θ and the running time. When reporting values, we write $\mathsf{XCut}_{\mathsf{mean}}$ for the mean value across these 10 runs, and $\mathsf{XCut}_{\mathsf{min}}$ for the minimum. As KaHiP and METIS behaved almost identically over all ℓ runs, we only report mean values for them. For each algorithm and each graph, we compute a partitioning consisting of $k \in \{2, 4, 8, 16, 32, 64, 128\}$ clusters as well as θ (smaller is better). As a second criterion, we compare the algorithms' running times.

All experiments were conducted on a server with an Intel Xeon 16 Core Processor and 1.5 TB of RAM running Ubuntu 22.04 with Linux kernel 5.15. XCut is implemented in `C++` and compiled using `gcc` 11.4 with full optimization[7]. For all other solvers, we followed the build instructions shipped with their code. As Graclus is single-threaded, we ran the single-threaded version of every algorithm. METIS was run in its default configuration. We used KaHiP with the `fsocial` flag, which is tailored to quickly partitioning social network-like graphs[8] and Graclus with options `-l 20 -b` to enable the local search step and only consider boundary vertices during local search, as suggested by the authors for larger graphs [107].

6.6.2 Configuring XCut

Greedy vs. Dynamic Programming (DP) Line 7 describes two heuristics for computing normalized cuts on the sparsifier. We compare for both heuristics the running time and θ across ten precomputed sparsifiers each for 19 representative graphs. The quality returned by DP was never better than that of Greedy. While the running times for both heuristics are linear in the size of the sparsifier, the DP approach scaled worse in k. For example, for $k = 32$, DP was three times slower than Greedy, and seven times slower for $k = 128$. See Fig. 6.7 for a plot with the results for $\rho = 10^{-4}$. Greedy was faster and produced no worse quality than DP for all values of ρ. Thus, we only report results for Greedy for all further experiments.

Parameter Choice for the Expander Decomposition (ρ, γ) In Fig. 6.1, we examine the effect of the threshold parameter ρ on the quality of solutions and running time. If the parameter is chosen too large, especially greater than one, the entire graph will likely be certified as an expander before any cuts are made, leading to very large θ. Furthermore,

[7] `-O3 -march=native -mtune=native`

[8] We chose this setting as we are especially interested in social network-like graphs.

Type (Abbreviation)	#	$\|V\|$	$\|E\|$	Δ
Bipartite (BP)	1	1.4M	4.3M	1.7k
Computational Fluids (CF)	1	17k	1.4M	269
Clustering (CL)	2	4.8k-100k	6.8k-500k	3-17
Citation Network (CN)	9	226k-1.1M	814k-56M	238-1.1k
Circuit Simulation (CS)	3	5k-30k	9.4k-54k	31-573
Duplicate Materials (DM)	1	14k	477k	80
Email Network (EM)	2	33k-34k	54k-181k	623-1383
Finite Elements (FE)	2	78k-100k	453k-662k	39-125
Infrastructure Network (IF)	2	2.9k-49k	6.5k-16k	19-242
Numerical Simulation (NS)	2	11k-449k	75k-3.3M	28-37
Optimization (OP)	2	37k-62k	131k-2.1M	54-8.4k
Random Graph (RD)	1	14k	919k	293
Road Network (RN)	4	114k-6.7M	120k-7M	6-12
Social Network (SN)	7	404k-4M	713k-28M	626-106k
Triangle Mixture (TM)	7	10k-77k	54k-2M	22-18k
US Census Redistricting (US)	1	330k	789k	58
Web Graph (WB)	3	1.3k-1.9M	2.8k-4.5M	59-2.6k

Table 6.1: Types and number of graphs in our benchmark dataset. Δ is the maximum degree, k and M are shorthand notations for 10^3 and 10^6, respectively. See the Table 6.3 for a detailed list.

we no longer obtain any significant improvement in θ for values of ρ below 10^{-4}. At the same time, the running time increases inversely with ρ as extra iterations are needed to converge, which is shown by the vertical arrangement of the dots in Fig. 6.1. For graph instance TM6, we also find that non-Pareto-optimal choices of ρ exist, namely 0.1 and 0.01, where both the running time and the returned value are worse than 10^{-3} and 10^{-4}. This suggests that the choice of ρ is an important design decision. On all our instances, $\rho = 10^{-4}$ produced Pareto-optimal results, so we conclude that we can configure this parameter to be a constant independently of the graph's structure.

For the automatic tuning of γ, we chose a starting threshold of 0.3, as this is around the largest value for which the expander decomposition finds structurally interesting cuts. Whenever γ is too large, i.e., the node reduction $|G_{i+1}|/|G_i| > 0.95$, we multiply γ by a factor of $\varepsilon = 0.8$ and restart the expander decomposition.

6.6.3 Comparison to Graclus, METIS, and KaHiP

Fig. 6.5 depicts the solution quality of every solver relative to that produced by Graclus on all instances for $k = 32$, showing both the mean and the minimum of the ten runs for XCut. For other values of k, the overall picture remains the same, see Fig. 6.2 and Section 6.11. We group the graphs by type, with Fig. 6.5 containing two plots, the upper showing

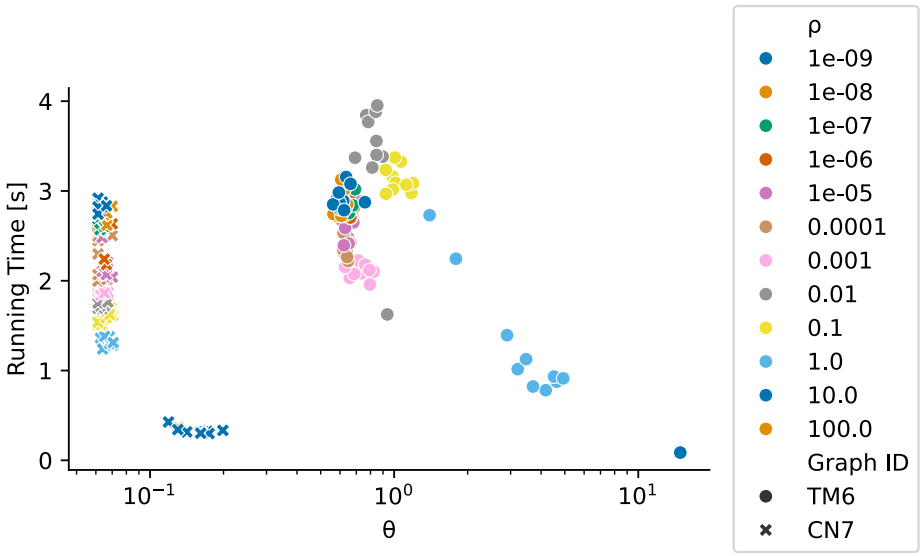

Figure 6.1: Running time vs. normalized cut for different choices of $\boldsymbol{\rho}$ on two different graphs for $\boldsymbol{k = 16}$. Colors denote different levels of $\boldsymbol{\rho}$, while shapes indicate the graph.

the disconnected IMDB graph and email, citation, and social network graphs as well as infrastructure networks, as these are the graphs on which XCut is particularly strong.

Looking at absolute values of θ, we find that across all instances, the geometric mean is at least 70 % lower than our competitors, see Table 6.2. Interestingly, when we only consider the seven graphs from [107], the geometric means become 0.84 for $\mathsf{XCut}_{\mathsf{min}}$ and 0.90 for $\mathsf{XCut}_{\mathsf{mean}}$, while they are 1.11, 1.19 and 1.03 for Graclus, METIS, and KaHiP respectively, which implies that KaHiP slightly outperforms Graclus in terms of θ on their benchmark (but not XCut).

One point of note is that XCut's variance on the social network SN7 representing user-user interactions in the Foursquare social network is high, meaning it does not always find small normalized cuts. This appears to be due to a very high-degree node that connects to approximately 15 % of all vertices, leading to fast convergence of the random walk. Due to the averaging effect of such a vertex, many nodes v have almost identical values u_v that vary only slightly between runs. Thus, when sorting by u-value, their order can vary greatly depending on the initial values, leading to very different candidate cuts. This is the only graph where the minimum and mean of the ten runs of XCut differ significantly (−33 % for $\mathsf{XCut}_{\mathsf{min}}$ and +39 % for $\mathsf{XCut}_{\mathsf{mean}}$ relative to Graclus). This indicates that the theoretical algorithm sketched in Section 6.4, which uses multiple concurrent random walks (corresponding to more attempts to find a good cut), would likely have reduced the variance here. This is also the only graph where the fact that we only use a single random walk impairs the quality of the result.

The graph classes on which XCut does not perform as well have fairly homogeneous

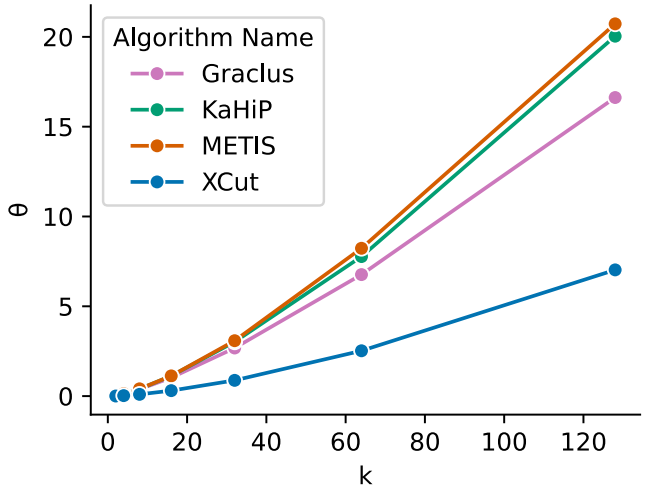

Figure 6.2: Geometric mean of the cut value $\boldsymbol{\theta}$ across all graphs for each $\boldsymbol{k}$ for $\mathsf{XCut}_{\mathsf{mean}}$, Graclus, METIS, and KaHiP.

degree distributions and often appear grid-like when drawn. We conjecture that in these grid-like graphs, there are no good expanders (which XCut is trying to find). Instead, sparse cuts arise mainly from the fact that the cut is balanced, i.e., the components we disconnect are all large enough, rather than being sparsely interconnected, which is exploited by the other solvers.

Our observed quantitative advantage of XCut also translates into a qualitative advantage, where we find that the partitions produced by XCut are more meaningful, than those produced by Graclus. For an example, see Fig. 6.3. In this figure, we contrast the partition produced by XCut and Graclus on the graph instance CL1, which is a synthetic graph used to benchmark clustering algorithms. We note that that the cuts made by XCut tend to be across the bottlenecks of the graph, while also producing parts of different sizes, if this means we can cut across bottlenecks. Graclus on the other hand, will produce parts that are more balanced, since it begins with a graph partitioning step similar to METIS, leading to a balanced initial partition. We can observe the effects of this in the center of the graph. Here, XCut produces 4 large parts by cutting across bottlenecks (pink, light green, dark olive green, and cyan) and 4 small separated communities (blue, purple, brown and gray-green), while Graclus splits this region of the graph into 11 similarly sized pieces. The difference in the objective value for this instance is only about a 25 % decrease, yet XCut yields a partition with more clearly separated communities.

6.6.4 Comparison to Zhao et al.

In Table 6.6 we compare θ for XCut and Graclus to the values reported by Zhao et al. [337] for $k = 30$. We observe a similar but weaker pattern as in the previous section. On graphs arising from numerical simulation and finite element problems, XCut performs

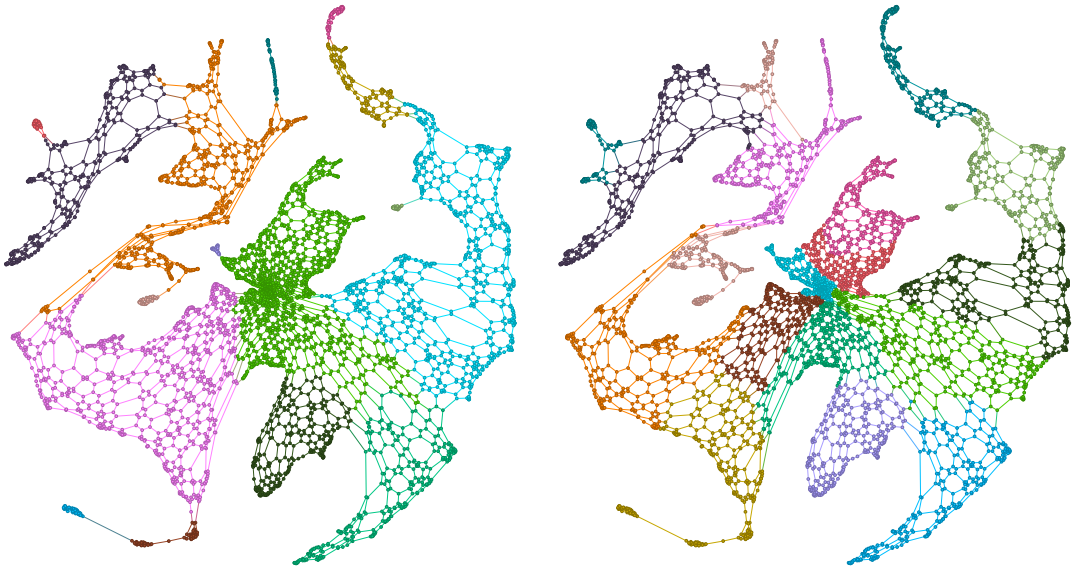

Figure 6.3: Qualitative comparison between XCut (left) and Graclus (right) for $k = 16$ on the graph CL1. We note that the solution of XCut only cuts through bottlenecks in the graph and produces clusters of different sizes, while the clusters produced by are Graclus more evenly sized.

Type	XCut$_{\text{mean}}$	XCut$_{\text{min}}$	Graclus	METIS	KaHiP
BP	**0.00**	**0.00**	1.18	1.38	1.58
CF	4.61	4.18	3.44	3.51	**3.15**
CL	0.80	**0.72**	1.04	1.13	1.02
CN	**0.07**	**0.07**	1.42	1.72	1.58
CS	0.44	**0.41**	0.51	0.59	0.57
DM	2.58	2.42	**2.11**	2.12	2.23
EM	0.44	**0.43**	3.44	3.78	3.80
FE	0.56	**0.53**	0.54	0.54	0.55
IF	**0.00**	**0.00**	0.93	1.11	1.12
NS	0.85	0.78	0.77	**0.76**	0.80
OP	0.71	**0.66**	1.47	1.48	0.99
RD	13.37	13.37	12.08	12.01	**11.94**
RN	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**
SN	0.25	**0.19**	3.43	3.98	4.00
TM	2.04	**1.93**	2.87	3.29	3.04
US	0.04	**0.03**	0.06	0.06	0.05
WB	**0.01**	**0.01**	0.10	0.14	0.14
All*	0.25	**0.22**	0.89	1.0	0.94

Table 6.2: Cut value ($\boldsymbol{\theta}$) of different algorithms. The geometric mean is taken across all graphs and values of $\boldsymbol{k \in \{2, 4, 8, 16, 32, 64, 128\}}$ for each graph type. *For the overall geometric mean, instances with $\boldsymbol{\theta = 0}$ were omitted. Only XCut detected such cases.

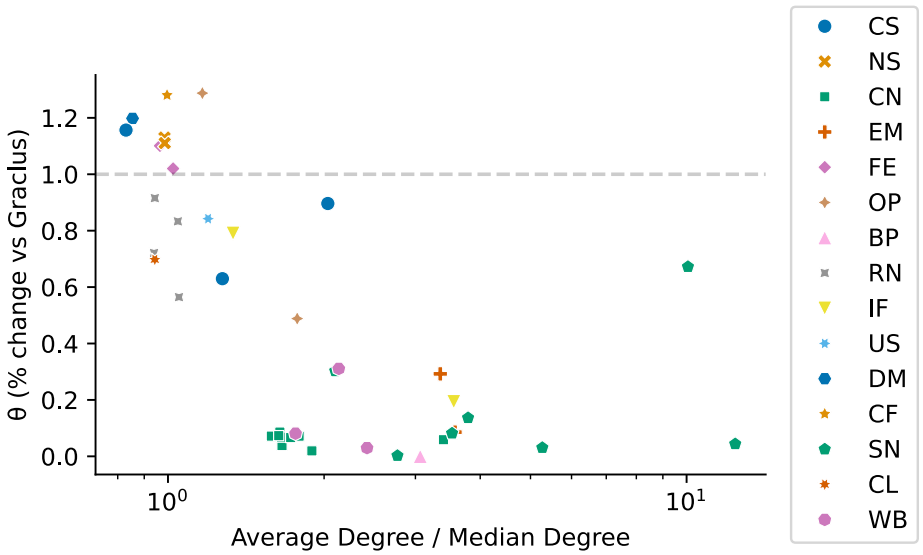

Figure 6.4: Relative improvement over Graclus (y-axis) vs. the average degree divided by the median degree (x-axis). Larger x-values signify that the graph exhibits a skewed, power-law-like degree distribution. Note the negative trend, except for the outlier towards the right corresponding to instance SN7.

worse than Zhao et al., but never by more than 36 %. On the triangle mixture instances, XCut achieves better θ on four instances, while their solver outperforms XCut on two instances. On citation networks, clustering instances, and those based on maps (RN1 and US1), XCut outperforms the algorithm by Zhao et al., with θ being up to 2.5 times lower on US1 and CN7. Altogether, XCut is better than Zhao et al. on roughly 2/3 of the instances, while Graclus does best on one instance. The geometric mean across all instances is 1.46 for $\text{XCut}_{\text{mean}}$, 1.39 for XCut_{min}, 1.64 for Zhao et al., and 3.06 for Graclus. We note that ours is 11 % lower for $\text{XCut}_{\text{mean}}$ and 15 % lower for XCut_{min}, while Graclus's value is almost twice that of the other solvers in the comparison due to it producing 5–30 times greater θ on the citation network (CN) instances.

While it is difficult to draw good conclusions for the running times from the values reported in [337] as no source code is available, we note that on some instances XCut takes less than 10 % of the reported time while producing higher-quality solutions, which might be indicative of a running time advantage of our solver.

6.6.5 Running Time

In our experiments, we found that on many graphs, the running time is spent mainly on computing the expander hierarchy, where on some instances, this step accounts for over 80% of the running time, even for $k = 128$. See Fig. 6.10 for some examples. However, even on the largest instances in our benchmark, the absolute running time never exceeded

Figure 6.5: Percentage deviation of the returned normalized cut value relative to Graclus for $\boldsymbol{k = 32}$. This means that a value of -75% indicates that the normalized cut value is 75% lower (i.e., better). The thin black bars indicate the standard error across our runs. The top graph shows the disconnected IMDB graph (BP1), citation network instances (CN), email networks (EM), infrastructure graphs (IF), social networks (SN), and web graphs (WB), while the bottom shows the remaining instances. See Table 6.3 for details.

18 minutes.

Overall, XCut is on average three times slower than Graclus, 20.8 times slower than METIS and 6.7 times slower than KaHiP for the mean execution time across all choices of k and all instances. Interestingly, we find that XCut's running time is lower than Graclus's on several social network graphs and the triangle mixture instances while it produces a better solution. See Fig. 6.11 for detailed running times on some exemplary instances.

Finally, recall that once our algorithm has computed a sparsifier, we can obtain a solution on the sparsifier for different values of k without recomputing the sparsifier, which is a unique feature among the solvers tested here. If we are interested in all seven values of k, e.g., and only count the time to compute the sparsifier once for XCut, our experiments take 0.56, 3.82, and 1.24 times the running time to compute partitions across all graphs and values of k using Graclus, METIS, and KaHiP, respectively. In particular, XCut then is 44 % *faster* than Graclus. This demonstrates the utility of XCut as a tool for exploratory data analysis. We could achieve further speedups by only computing a solution for $k = 128$ and then choosing the initial subsets of edges for the other values of k, only performing the refinement.

6.6.6 Discussion

XCut outperforms other software when computing normalized cuts on social, citation, email, and infrastructure networks and web graphs. It performs slightly worse on graphs arising from specific computational tasks, such as finite elements, circuit, or numerical simulations. The graphs on which this behavior occurs tend to have degree distributions concentrated around the average degree, suggesting that they are not graphs with a scale-free structure.

In Fig. 6.4 we plot the relationship between our improvement over Graclus and the value of $\frac{\frac{1}{n}\sum_{v \in V} d_v}{\text{median}_{v \in V} d_v}$ for the non-synthetic graphs of our benchmark. This value measures how much the mean and median diverge due to outlier nodes with very high degrees. The graphs with power-law distributions tend to have a higher value on this measure, and we find that there appears to be a negative correlation, with one outlier due to instance SN7. We also observe a similar relationship when plotting the relative improvement over $\frac{\Delta}{\text{median}_{v \in V} d_v}$, see Fig. 6.6.

6.7 Conclusion

In this work we introduced XCut, a new algorithm for solving the normalized cut problem. It is based on a novel expander decomposition algorithm and to the best of our knowledge, it is the first practical application of the expander hierarchy. XCut clearly outperforms other solvers in the experimental study on social, citation, email, and infrastructure networks and web graphs, and also in the geometric mean over all instances. It scales to instances with tens of millions of edges and can produce solutions for multiple numbers of clusters k with little overhead by comparison. We are confident that with further optimization and the use of parallelism it will be possible to scale our algorithm to even larger graphs, while further improving the solution quality, especially since computing the expander decomposition appears to be highly parallelizable.

We also believe that the expander hierarchy and our expander decomposition can be applied to other graph cut problems in the future, as the tree flow sparsifiers approximate *all* cuts in the graph, and there are theoretical results that suggest this might be the case.

An interesting direction in both theoretical and practical terms may be to extend our efficient expander decomposition to dynamic graphs.

XCut is open source software and its code is freely available on GitLab [149].

6.8 Additional Related Work

The Expander Hierarchy The expander hierarchy, introduced by Goranci et al. [139] is a form of hierarchical graph clustering, which produces a tree cut sparsifier. A tree cut sparsifier of the graph $G = (V, E)$ is a weighted tree where the vertices in V are the leaves, and for each $S \subset V$ we have that the minimum weight of a cut separating the leaves in S from those in $V \setminus S$ in T is within a multiplicative factor q to the value of the minimum cut of $(S, V \setminus S)$ in G. See [265] for details. Goranci et al. [139] show that by

using expander hierarchy in an iterative fashion by repeatedly contracting each expander we can construct a tree cut sparsifier of subpolynomial quality $n^{o(1)}$ in nearly linear time. Additionally, they show that the cut sparsifier can be maintained dynamically under insertions and deletions of edges. The update time is also $n^{o(1)}$.

The authors use the expander decomposition of Saranurak and Wang [278], but crucially modify it to decompose the graph into so-called *boundary-linked expanders*. This notion of expander decomposition is stronger than regular expander decomposition, as it says that the conductance of each component is still some value ϕ, even when increasing the weight of vertices on the boundary, an operation which can only decrease the value of the conductance, as the denominator can only increase for each set of vertices.

The boundary linkedness is crucial for obtaining the theoretical guarentees of [139], but we found that in practice it was not necessary to use stronger notions of expander decomposition to obtain competitive results from the sparsifier.

Multilevel Graph Partitioning This framework has been employed by many successful graph partitioning tools, e.g., METIS [186], Graclus [107], or KaHiP [277]. The general idea of the paradigm is to compute a small summary ("coarsening") of the graph, on which it is easier to solve the problem we are interested in, and then mapping this coarse solution onto the original graph. In more technical terms, a solver following this paradigm has three phases: *(i) Coarsening*: Given an original graph $G = (V, E)$ that we want to partition, we compute a series of successively smaller graphs $G = G_0, G_1, \ldots, G_\ell$ such that $n = |V_0| > |V_1| > \cdots > |V_\ell|$ with the aim to create a coarse summary of the original graph. *(ii) Solving*: Obtain a solution to the initial problem on the graph G_ℓ. Although G_ℓ is small, heuristics are usually employed here. *(iii) Refinement*: Successively map the solution from graph G_i to graph G_{i-1}, while applying heuristics that increase the solution quality.

6.9 Implementation Details

In this section we give some additional details on the implementation of XCut, especially the local search step.

Local search Whenever we go from graph G_i to graph G_{i-1}, we create a new cluster for each solution edge going from level i to level $i-1$ in the sparsifier. Hence, the number of clusters will increase monotonically as we descend the hierarchy, and we may have less than k clusters on an intermediate level. After introducing the new clusters, we perform a local search in graph G_{i-1}. For each round of refinement, we first insert all vertices that lie on the boundary of a cluster into a priority queue, with the priority of vertex v in cluster C being $deg_{G_i \setminus C}(v) - deg_C(v)$. Next, we attempt to move every vertex from the queue to the the neighboring cluster C' such that $E(v, C') = \max_j E(v, C_j)$, i.e., the cluster it is most strongly connected to. f this leads to a decrease in the target function, through making the partition more balanced or a reduction of the number of edges crossing between clusters, we move the vertex to C' and update the priorities of

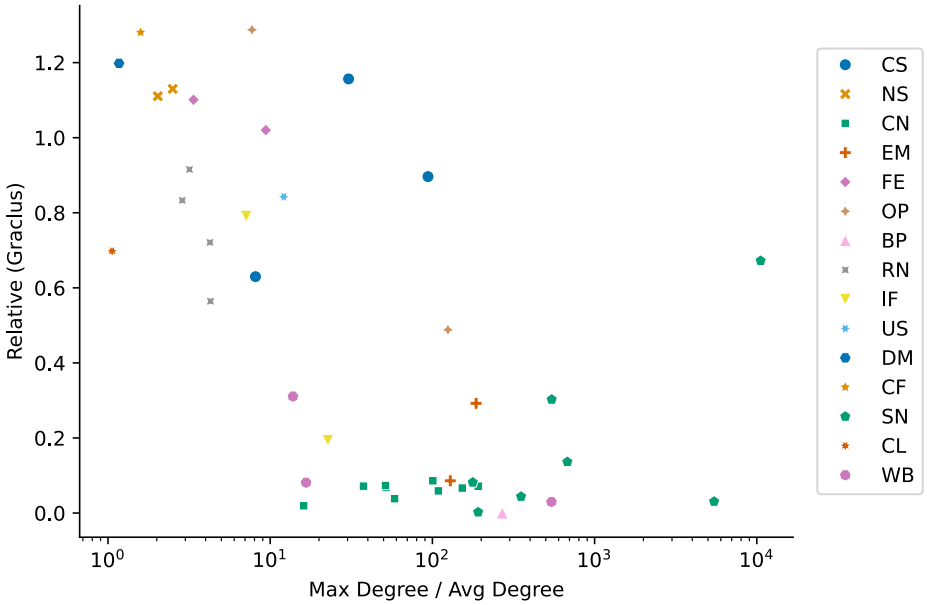

Figure 6.6: Relative improvement over Graclus (y-axis) vs. the ratio of the maximum degree and the median degree (x-axis). The value on the x-axis is larger if the graphs exhibit a distribution with large outliers. Note the negative trend, except for the outlier towards the right corresponding to instance SN7.

#	Name	Type	$\|V\|$	$\|E\|$	Δ	d_{mean}	25th	50th	75th	90th
BP1	imdb	Bipartite	1 403 278	4 303 383	1652	6.13	1	2	6	15
CF1	ramage02	Computational Fluids	16 830	1 424 761	269	169.31	131	170	170	269
CL1	uk	Clustering	4824	6837	3	2.83	3	3	3	3
CL2	smallworld	Clustering	100 000	499 998	17	10.00	9	10	11	12
CN1	citationCiteseer	Citation Network	268 495	1 156 647	1318	8.62	2	5	10	18
CN2	ca-hollywood-2009	Citation Network	1 069 126	56 306 653	11 467	105.33	13	31	75	212
CN3	coAuthorsDBLP	Citation Network	299 067	977 676	336	6.54	2	4	7	14
CN4	ca-MathSciNet	Citation Network	332 689	820 644	496	4.93	1	3	5	11
CN5	ca-coauthors-dblp	Citation Network	540 486	15 245 729	3299	56.41	13	34	74	135
CN6	ca-dblp-2012	Citation Network	317 080	1 049 866	343	6.62	2	4	7	14
CN7	ca-citeseer	Citation Network	227 320	814 134	1372	7.16	2	4	8	15
CN8	coPapersCiteseer	Citation Network	434 102	16 036 720	1188	73.88	15	39	92	177
CN9	ca-dblp-2010	Citation Network	226 413	716 460	238	6.33	2	4	7	13
CS1	add32	Circuit Simulation	4960	9462	31	3.82	2	3	4	9
CS2	rajat10	Circuit Simulation	30 202	50 101	101	3.32	2	4	4	4
CS3	memplus	Circuit Simulation	17 758	54 196	573	6.10	2	3	4	11
DM1	pcrystk02	Duplicate Materials	13 965	477 309	80	68.36	53	80	80	80
EM1	email-enron-large	Email Network	33 696	180 811	1383	10.73	1	3	7	19
EM2	email-EU	Email Network	32 430	54 397	623	3.35	1	1	1	3
FE1	fe_tooth	Finite Elements	78 136	452 591	39	11.58	6	12	15	18
FE2	fe_rotor	Finite Elements	99 617	662 431	125	13.30	11	13	14	17
IF1	inf-openflights	Infrastructure Network	2939	15 677	242	10.67	2	3	8	28
IF2	inf-power	Infrastructure Network	4941	6594	19	2.67	2	2	3	5
NS1	wing_nodal	Numerical Simulation	10 937	75 488	28	13.80	12	14	16	17

Table 6.3: Statistics on the number of vertices, edges and degree distribution of the graph instances in our benchmark dataset. Continued in 6.4 Δ denotes the maximum, d_{mean} the average vertex degree. The last four columns are percentiles of the degree distribution.

other boundary vertices and insert any vertices which now lie on the boundary. To prevent a cluster from becoming empty during the projection step, we disallow moves that leave only vertices in the cluster whose vertex-leaf paths cross an edge of the solution, as these vertices will become part of another cluster during the projection step later on. During a single round, each vertex can move at most once, and if no improvement to the target function is observed after several moves, we terminate the round. At the end of the round, we commit to the initial segment of the move sequence that gives the best local optimum. On each level, we perform refinement rounds until no more progress is made during a round or we have exceeded the maximal number of iterations.

6.10 Instance List

See Table 6.3 for a full list of instances and information on their degree distribution.

#	Name	Type	$\|V\|$	$\|E\|$	Δ	d_{mean}	25th	50th	75th	90th
NS2	auto	Numerical Simulation	448 695	3 314 611	37	14.77	13	15	16	18
OP1	gupta2	Optimization	62 064	2 093 111	8412	67.45	25	38	47	49
OP2	finance256	Optimization	37 376	130 560	54	6.99	4	6	8	12
RD1	appu	Random Graph	14 000	919 552	293	131.36	109	131	154	176
RN1	inf-roadNet-CA	Road Network	1 957 027	2 760 388	12	2.82	2	3	3	4
RN2	inf-roadNet-PA	Road Network	1 087 562	1 541 514	9	2.83	2	3	4	4
RN3	inf-italy-osm	Road Network	6 686 493	7 013 978	9	2.10	2	2	2	3
RN4	luxembourg_osm	Road Network	114 599	119 666	6	2.09	2	2	2	3
SN1	soc-youtube-snap	Social Network	1 134 890	2 987 624	28 754	5.27	1	1	3	8
SN2	soc-flickr	Social Network	513 969	3 190 452	4369	12.41	1	1	5	17
SN3	soc-lastfm	Social Network	1 191 805	4 519 330	5150	7.58	1	2	4	11
SN4	soc-twitter-follows	Social Network	404 719	713 319	626	3.53	1	1	1	3
SN5	soc-pokec	Social Network	1 632 803	22 301 964	14 854	27.32	4	13	35	70
SN6	soc-livejournal	Social Network	4 033 137	27 933 062	2651	13.85	2	5	15	35
SN7	soc-FourSquare	Social Network	639 014	3 214 986	106 218	10.06	1	1	4	19
TM1	vsp_vibrobox_[...]	Triangle Mixture	77 328	435 586	669	11.27	3	5	8	26
TM2	vsp_bump2_[...]	Triangle Mixture	56 438	300 801	604	10.66	5	7	11	18
TM3	vsp_barth5_[...]	Triangle Mixture	32 212	101 805	22	6.32	6	6	7	7
TM4	vsp_model1_[...]	Triangle Mixture	45 101	189 976	17 663	8.42	3	5	7	9
TM5	vsp_p0291_[...]	Triangle Mixture	10 498	53 868	229	10.26	3	7	11	16
TM6	vsp_bcsstk30_[...]	Triangle Mixture	58 348	2 016 578	219	69.12	45	64	92	106
TM7	vsp_befref_[...]	Triangle Mixture	14 109	98 224	1531	13.92	6	9	11	15
US1	mi2010	US Census	329 885	789 045	58	4.78	3	4	5	8
WB1	web-it-2004	Web Graph	509 338	7 178 413	469	28.19	14	16	19	39
WB2	web-google	Web Graph	1299	2773	59	4.27	1	2	5	12
WB3	web-wikipedia2009	Web Graph	1 864 433	4 507 315	2624	4.84	1	2	5	10

Table 6.4: Statistics on the number of vertices, edges and degree distribution of the graph instances in our benchmark dataset. Continued from 6.3. Δ denotes the maximum, d_{mean} the average vertex degree. The last four columns are percentiles of the degree distribution. The full names of TM1 through TM7 are vsp_vibrobox_scagr7-2c_rlfddd, vsp_bump2_e18_aa01_model1_crew1, vsp_barth5_1Ksep_50in_5Kout, vsp_model1_crew1_cr42_south31, vsp_p0291_seymourl_iiasa, vsp_bcsstk30_500sep_10in_1Kout, and vsp_befref_fxm_2_4_air02, respectively.

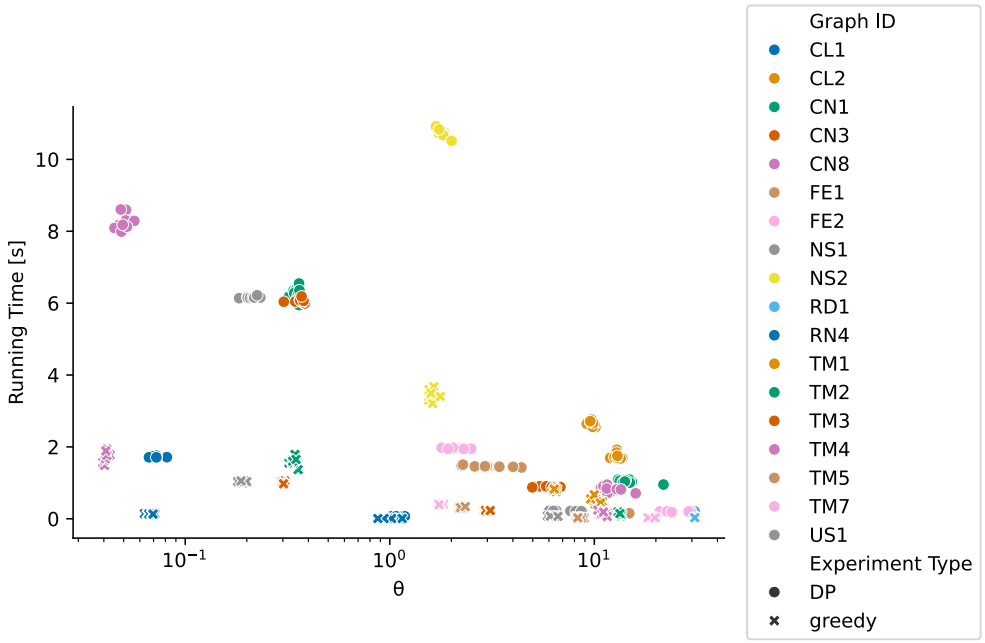

Figure 6.7: XCut with the greedy vs. dynamic programming strategy for a subset of instances for $k = 32$ and $\rho = 0.0001$.

6.11 Additional Figures

In this section we include a number of figures omitted from the experimental section of the chapter.

Type	k / Algorithm Name	2	4	8	16	32	64	128
Bipartite	Graclus	0.05	0.17	0.45	1.09	3.55	9.18	24.41
	KaHiP	0.04	0.27	0.67	1.92	4.84	13.03	33.65
	METIS	0.03	0.17	0.59	1.66	4.66	13.07	34.55
	XCut	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Circuit Simulation	Graclus	**0.01**	0.06	**0.18**	**0.54**	1.57	4.76	14.04
	KaHiP	**0.01**	0.07	0.21	0.62	1.78	5.65	16.99
	METIS	0.02	0.07	0.22	0.59	1.72	5.49	16.90
	XCut	**0.01**	**0.05**	**0.18**	**0.54**	**1.41**	**4.21**	**11.90**
Citation Network	Graclus	0.08	0.26	0.67	1.57	3.75	8.22	17.76
	KaHiP	0.07	0.27	0.73	1.81	4.27	9.57	21.47
	METIS	0.08	0.29	0.81	2.01	4.80	11.04	24.32
	XCut	**0.00**	**0.01**	**0.03**	**0.08**	**0.22**	**0.61**	**1.69**
Clustering	Graclus	0.04	0.14	0.43	1.20	3.14	8.19	20.69
	KaHiP	0.04	0.13	0.43	1.15	3.11	7.91	19.95
	METIS	0.04	0.15	0.46	1.25	3.37	8.58	21.38
	XCut	**0.02**	**0.09**	**0.29**	**0.93**	**2.52**	**7.07**	**19.49**
Computational Fluids	Graclus	**0.12**	0.46	**1.32**	**3.86**	**10.49**	**27.76**	**70.28**
	KaHiP	**0.12**	0.46	1.47	4.18	10.98	29.64	74.51
	METIS	**0.12**	**0.44**	1.35	4.10	10.95	28.65	72.55
	XCut	0.15	0.55	1.63	4.75	14.30	44.39	108.44
Duplicate Materials	Graclus	**0.04**	**0.24**	**0.80**	**2.58**	**7.71**	**20.42**	**53.60**
	KaHiP	0.05	0.32	0.85	2.97	8.20	21.45	56.65
	METIS	**0.04**	**0.24**	**0.80**	2.60	7.75	20.67	55.85
	XCut	0.05	0.26	0.90	3.27	9.54	26.81	81.90
Email Network	Graclus	0.18	0.60	1.65	4.07	9.11	20.13	42.43
	KaHiP	0.19	0.66	1.70	4.36	10.11	23.19	51.73
	METIS	0.19	0.64	1.80	4.40	10.21	23.06	51.24
	XCut	**0.01**	**0.04**	**0.17**	**0.51**	**1.46**	**4.07**	**11.76**
Finite Elements	Graclus	**0.01**	**0.06**	0.21	**0.63**	**1.82**	**5.16**	**13.97**
	KaHiP	**0.01**	**0.06**	0.21	0.66	2.00	5.63	15.40
	METIS	**0.01**	**0.06**	**0.20**	0.65	1.88	5.23	14.42
	XCut	**0.01**	**0.06**	0.21	0.67	1.99	5.67	15.64
Infrastructure Network	Graclus	0.02	0.09	0.39	1.09	3.19	9.97	27.44
	KaHiP	0.02	0.09	0.44	1.51	4.41	12.24	32.27
	METIS	0.02	0.11	0.49	1.36	4.16	11.60	31.29
	XCut	**0.00**	**0.00**	**0.00**	**0.23**	**1.38**	**5.30**	**17.73**
Numerical Simulation	Graclus	0.02	**0.08**	0.28	0.94	2.70	**7.24**	**19.40**
	KaHiP	0.02	0.09	0.31	0.93	2.76	7.77	20.76
	METIS	0.02	0.09	**0.26**	**0.89**	**2.63**	7.25	19.43
	XCut	**0.01**	0.09	0.33	1.10	3.18	8.88	23.89
Optimization	Graclus	0.04	0.14	0.40	1.85	5.65	16.68	42.32
	KaHiP	**0.00**	0.07	0.32	1.86	5.84	18.65	48.48
	METIS	0.04	0.09	0.40	1.83	5.55	16.60	47.46
	XCut	0.01	**0.03**	**0.22**	**1.38**	**4.63**	**13.60**	**33.06**
Random Graph	Graclus	0.91	**2.59**	**6.19**	**13.52**	28.37	**57.79**	**116.97**
	KaHiP	0.88	2.67	6.37	13.83	28.62	58.17	118.02
	METIS	**0.85**	2.61	**6.19**	13.53	**28.36**	58.18	117.98
	XCut	0.99	2.99	6.98	14.99	30.99	62.99	126.99
Road Network	Graclus	**0.00**	**0.00**	**0.00**	**0.01**	**0.03**	0.10	**0.30**
	KaHiP	**0.00**	**0.00**	**0.00**	**0.01**	**0.03**	**0.09**	**0.30**
	METIS	**0.00**	**0.00**	**0.00**	**0.01**	**0.03**	0.10	**0.30**
	XCut	**0.00**	**0.00**	**0.00**	**0.01**	**0.03**	0.11	0.34
Social Network	Graclus	0.16	0.66	1.64	4.01	9.03	19.96	44.26
	KaHiP	0.20	0.73	1.92	4.66	10.59	24.11	51.14
	METIS	0.19	0.68	1.87	4.67	10.91	24.54	53.15
	XCut	**0.01**	**0.03**	**0.11**	**0.31**	**0.83**	**2.19**	**6.02**
Triangle Mixture	Graclus	0.08	0.43	1.43	3.64	8.94	20.71	48.90
	KaHiP	0.08	0.50	1.54	3.94	9.62	22.73	52.85
	METIS	0.09	0.56	1.60	4.14	10.13	23.10	51.51
	XCut	**0.03**	**0.25**	**1.08**	**3.08**	**7.70**	**18.44**	**44.26**
US Census Redistricting	Graclus	**0.00**	0.01	**0.02**	0.07	0.21	0.64	1.92
	KaHiP	**0.00**	**0.00**	**0.02**	**0.06**	0.20	**0.59**	**1.80**
	METIS	**0.00**	0.01	**0.02**	0.07	0.21	0.68	1.99
	XCut	**0.00**	**0.00**	**0.02**	0.07	**0.19**	**0.59**	1.92
Web Graph	Graclus	**0.00**	0.02	0.04	0.10	0.26	0.77	2.02
	KaHiP	**0.00**	0.01	0.04	0.11	0.47	2.09	7.36
	METIS	0.01	0.02	0.03	0.08	0.42	2.46	9.01
	XCut	**0.00**	**0.00**	**0.00**	**0.01**	**0.02**	**0.10**	**0.36**

Table 6.5: Geometric mean of normalized cut value across graph types over k for the multilevel graph partitioning algorithms from Section 6.6.3.

GID	XCut$_{mean}$	XCut$_{min}$	Zhao	Graclus
CL1	0.87	**0.77**	1.05	1.15
CL2	6.00	**5.87**	7.05	7.61
CN1	0.31	**0.29**	0.52	4.06
CN3	**0.27**	**0.27**	0.49	3.66
CN5	**0.13**	**0.13**	0.14	3.24
CN7	0.17	**0.16**	0.41	2.21
CN8	**0.04**	**0.04**	0.06	1.88
FE1	2.06	1.96	**1.68**	1.74
FE2	1.58	1.55	1.50	**1.45**
NS1	5.73	5.35	**4.71**	**4.71**
NS2	1.45	1.39	**1.08**	1.17
RD1	28.98	28.98	**23.80**	26.48
RN4	**0.06**	**0.06**	0.07	0.07
TM1	9.53	8.91	**6.85**	8.48
TM2	12.34	**12.06**	13.55	16.63
TM3	2.78	**2.65**	2.72	2.78
TM4	10.09	**9.58**	10.48	14.45
TM5	7.77	**7.48**	7.88	13.25
TM6	2.11	**1.94**	2.09	2.1
TM7	17.51	16.90	**12.83**	15.6
US1	0.17	**0.16**	0.41	0.19
All	1.46	**1.39**	1.64	3.06

Table 6.6: Cut values and running time of our Algorithm vs the values reported by Zhao et al. [337]. All values are for $k = 30$. The Cut-columns contain the normalized cut value (lower is better). The minimum value in each row is marked bold. The bottom row contains the geometric mean of all values.

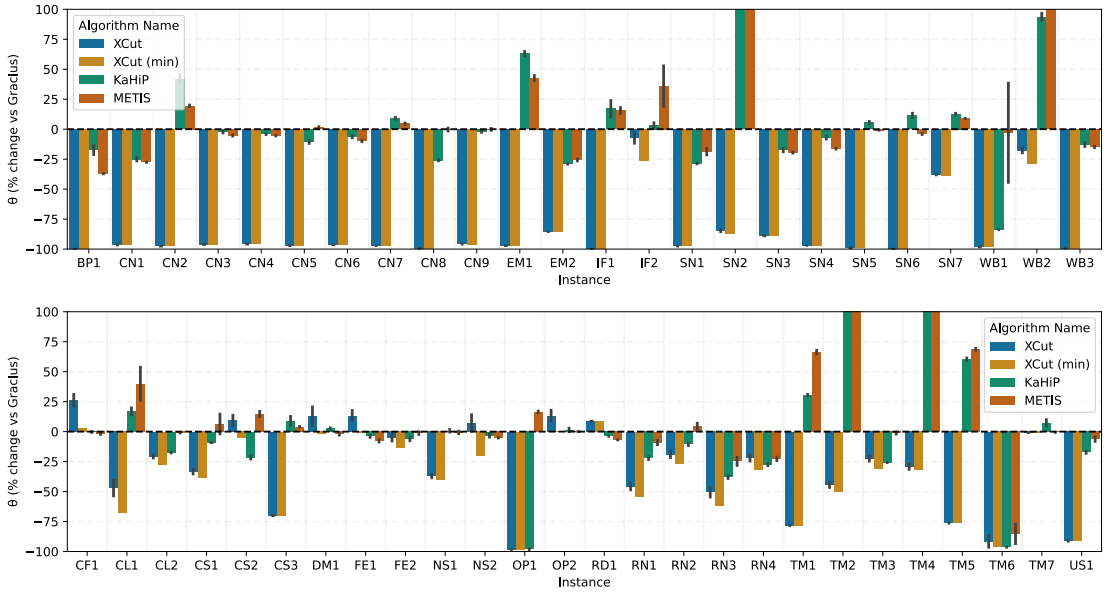

Figure 6.8: Percentage deviation of the returned normalized cut value relative to Graclus for $k = 2$. This means that a value of -75% means that the normalized cut value is 75% lower (i.e., better). The thin black bars indicate the standard error across our runs. The top graph shows the disconnected IMDB graph (BP1), citation network instances (CN), email networks (EM), infrastructure graphs (IF), social networks (SN) and web graphs (WB), while the bottom shows the remaining instances. See Table 6.3 for details.

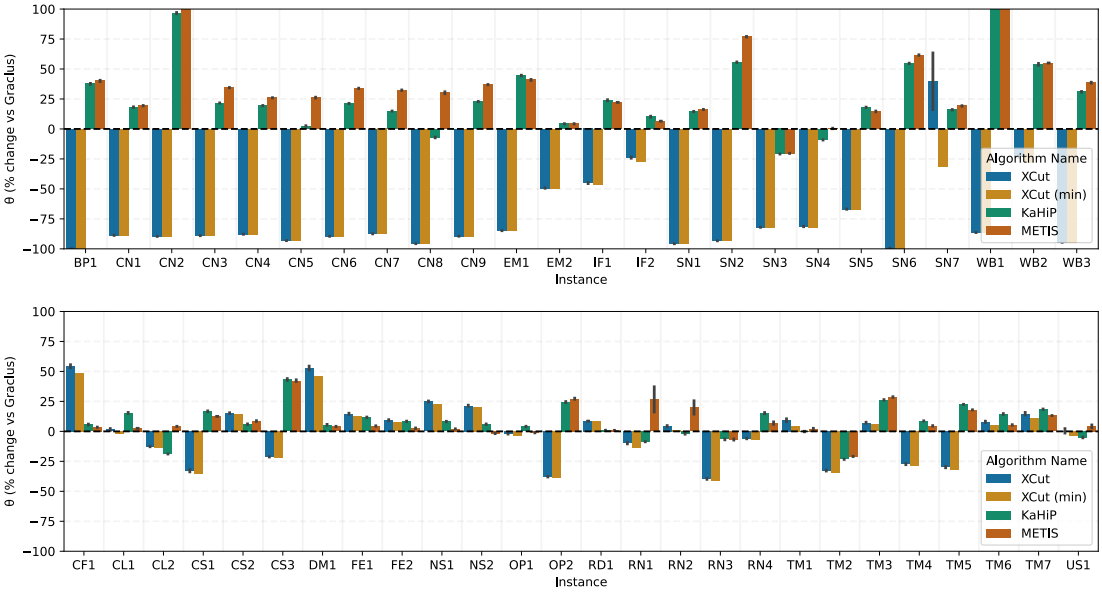

Figure 6.9: Percentage deviation of the returned normalized cut value relative to Graclus for $k = 128$. This means that a value of -75% means that the normalized cut value is 75% lower (i.e., better). The thin black bars indicate the standard error across our runs. The top graph shows the disconnected IMDB graph (BP1), citation network instances (CN), email networks (EM), infrastructure graphs (IF), social networks (SN) and web graphs (WB), while the bottom shows the remaining instances. See Table 6.3 for details.

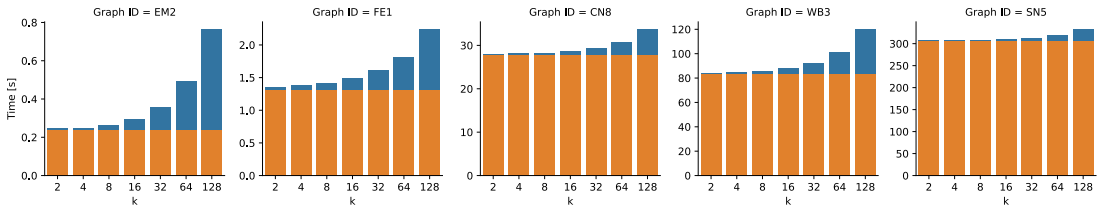

Figure 6.10: Plot showcasing the time taken to compute the expander hierarchy in orange and the total time to compute a normalized cut in blue for five graph instances of different sizes.

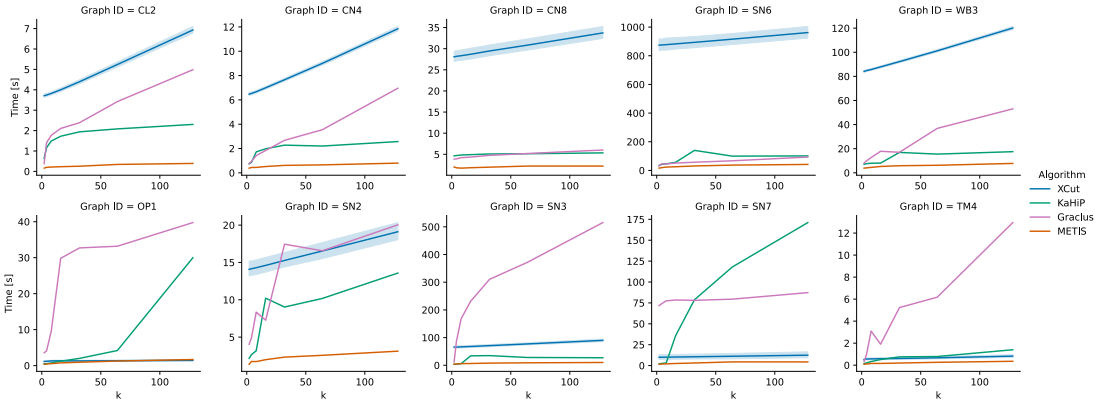

Figure 6.11: Plots of running time versus k. The first row shows the usual behavior of the solvers in our comparison, with our solver starting at a much higher level than the competition. This is due to the fact that we need to first compute the expander hierarchy, which gives a hard lower bound on the running time of our algorithm for any choice of k. The running time then grows linearly with k, due to Greedy taking time $\mathcal{O}(nk)$. The bottom row contains some instances on which the running time results are unusual. Often Graclus running time grows very quickly, but we are unsure what causes this on these instances in particular.

Bibliography

[1] Mohsen Abbasi, Aditya Bhaskara, and Suresh Venkatasubramanian. Fair clustering via equitable group representations. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 504–514, 2021.

[2] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[3] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

[4] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–1, 2012.

[5] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *International conference on machine learning*, pages 60–69. PMLR, 2018.

[6] Alekh Agarwal, Miroslav Dudík, and Zhiwei Steven Wu. Fair regression: Quantitative definitions and reduction-based algorithms. In *International Conference on Machine Learning*, pages 120–129. PMLR, 2019.

[7] Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52(1):1–30, 2005.

[8] Charu C Aggarwal and Chandan K Reddy. Data clustering. *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra*, 2014.

[9] Sara Ahmadian, Alessandro Epasto, Marina Knittel, Ravi Kumar, Mohammad Mahdian, Benjamin Moseley, Philip Pham, Sergei Vassilvitskii, and Yuyan Wang. Fair hierarchical clustering. *Advances in Neural Information Processing Systems*, 33:21050–21060, 2020.

[10] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017.

[11] Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Computer Science Review*, 37:100253, 2020.

[12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 5–14, 2012.

[13] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Spectral sparsification in dynamic graph streams. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 1–10. Springer, 2013.

[14] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75:245–248, 2009.

[15] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[16] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996.

[17] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3287–3318. World Scientific, 2018.

[18] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms from precision sampling. *arXiv preprint arXiv:1011.1263*, 2010.

[19] Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 106–113, New York, NY, USA, 1998. Association for Computing Machinery.

[20] David Arthur and Sergei Vassilvitskii. `k-means++`: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035. ACM, New York, 2007.

[21] Isaac Arvestad. Near-linear time expander decomposition in practice, 2022.

[22] Bas Fagginger Auer and Rob H Bisseling. Graph coarsening and clustering on the gpu. *Graph Partitioning and Graph Clustering*, 588(223):2, 2012.

[23] Haim Avron, Kenneth L Clarkson, and David P Woodruff. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1116–1138, 2017.

[24] Yossi Azar, Umang Bhaskar, Lisa Fleischer, and Debmalya Panigrahi. Online mixed packing and covering. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 85–100. SIAM, 2013.

[25] Yossi Azar, Niv Buchbinder, T.-H. Hubert Chan, Shahar Chen, Ilan Reuven Cohen, Anupam Gupta, Zhiyi Huang, Ning Kang, Viswanath Nagarajan, Joseph Naor, and Debmalya Panigrahi. Online algorithms for covering and packing problems with convex objectives. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 148–157. IEEE Computer Society, 2016.

[26] Olivier Bachem, Mario Lucic, Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[27] Olivier Bachem, Mario Lucic, S. Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016.

[28] Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

[29] Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k -means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1119–1127, New York, NY, USA, 2018. Association for Computing Machinery.

[30] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *International Conference on Machine Learning*, pages 405–413. PMLR, 2019.

[31] Metin Balaban, Niema Moshiri, Uyen Mai, Xingfan Jia, and Siavash Mirarab. Treecluster: Clustering biological sequences using phylogenetic trees. *PloS one*, 14(8):e0221068, 2019.

[32] Frank Ban, Vijay Bhattiprolu, Karl Bringmann, Pavel Kolev, Euiwoong Lee, and David P. Woodruff. A PTAS for ℓ_p-low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 747–766, 2019.

[33] Frank Ban, David P. Woodruff, and Qiuyi (Richard) Zhang. Regularized weighted low rank approximation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 4061–4071, 2019.

[34] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012.

[35] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4), aug 2012.

[36] Nikhil Bansal, Joseph Naor, and Ohad Talmon. Efficient online weighted multi-level paging. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 94–104, 2021.

[37] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and machine learning: Limitations and opportunities*. MIT press, 2023.

[38] Yair Bartal, Amos Fiat, Howard Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 51–58, 1992.

[39] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: Beyond subspaces and the johnson-lindenstrauss lemma. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 1039–1050, New York, NY, USA, 2019. Association for Computing Machinery.

[40] Radim Belohlávek and Vilém Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. Syst. Sci.*, 76(1):3–20, 2010.

[41] András A Benczúr and David R Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.

[42] Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. *Advances in Neural Information Processing Systems*, 32, 2019.

[43] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth. Fairness in criminal justice risk assessments: The state of the art. *Sociological Methods & Research*, 50(1):3–44, 2021.

[44] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[45] Charles-Edmond Bichot. Co-clustering documents and words by minimizing the normalized cut objective function. *J. Math. Model. Algorithms*, 9(2):131–147, 2010.

[46] Marcin Bienkowski, David Fuchssteiner, Jan Marcinkowski, and Stefan Schmid. Online dynamic b-matching: With applications to reconfigurable datacenter networks. *ACM SIGMETRICS Performance Evaluation Review*, 48(3):99–108, 2021.

[47] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[48] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, 29, 2016.

[49] Thomas Bottesch, Thomas Bühler, and Markus Kächele. Speeding up k-means by approximating Euclidean distances via block vectors. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2578–2586, New York, New York, USA, 20–22 Jun 2016. PMLR.

[50] Christos Boutsidis, Anastasios Zouzias, Michael W Mahoney, and Petros Drineas. Randomized dimensionality reduction for k-means clustering. *IEEE Transactions on Information Theory*, 61(2):1045–1062, 2014.

[51] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *CoRR*, abs/1612.00889, 2016.

[52] Vladimir Braverman, Dan Feldman, Harry Lang, Adiel Statman, and Samson Zhou. Efficient coreset constructions via sensitivity sampling. In *Asian Conference on Machine Learning, ACML*, pages 948–963, 2021.

[53] Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *arXiv preprint arXiv:1011.2571*, 2010.

[54] James Briggs. Faiss: The missing manual. `https://www.pinecone.io/learn/faiss/`.

[55] Karl Bringmann, Pavel Kolev, and David P. Woodruff. Approximation algorithms for l_0-low rank approximation. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 6648–6659, 2017.

[56] J Paul Brooks, José H Dulá, and Edward L Boone. A pure l1-norm principal component analysis. *Computational statistics & data analysis*, 61:83–98, 2013.

[57] Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, page 737–756, USA, 2015. Society for Industrial and Applied Mathematics.

[58] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual, textual and link information. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 952–959, 2004.

[59] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. Mining hidden community in heterogeneous social networks. In Jafar Adibi, Marko Grobelnik, Dunja Mladenic, and Patrick Pantel, editors, *Proceedings of the 3rd international workshop on Link discovery, LinkKDD 2005, Chicago, Illinois, USA, August 21-25, 2005*, pages 58–65. ACM, 2005.

[60] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering*, 30(9):1616–1637, 2018.

[61] Marco Capó, Aritz Pérez, and Jose A Lozano. An efficient k-means clustering algorithm for massive data. *arXiv preprint arXiv:1801.02949*, 2018.

[62] Simon Caton and Christian Haas. Fairness in machine learning: A survey. *ACM Computing Surveys*, 56(7):1–38, 2024.

[63] M Emre Celebi. Improving the performance of k-means for color quantization. *Image and Vision Computing*, 29(4):260–271, 2011.

[64] L Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K Vishnoi. Classification with fairness constraints: A meta-algorithm with provable guarantees. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 319–328, 2019.

[65] Parinya Chalermsook, Sandy Heydrich, Eugenia Holm, and Andreas Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Proceedings*, volume 8737, pages 235–246, 2014.

[66] L. Sunil Chandran, Davis Issac, and Andreas Karrenbauer. On the parameterized complexity of biclique cover and partition. In *11th International Symposium on Parameterized and Exact Computation, IPEC*, pages 11:1–11:13, 2016.

[67] Yi-Jun Chang, Seth Pettie, Thatchaphol Saranurak, and Hengjie Zhang. Near-optimal distributed triangle enumeration via expander decompositions. *J. ACM*, 68(3):21:1–21:36, 2021.

[68] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[69] Moses Charikar and Erik Waingarten. Polylogarithmic sketches for clustering. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[70] Antoine Chatalic, Rémi Gribonval, and Nicolas Keriven. Large-scale high-dimensional clustering with fast sketching. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4714–4718. IEEE, 2018.

[71] Ke Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

[72] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022.

[73] Sitan Chen, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Symmetric sparse boolean matrix factorization and applications. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, pages 46:1–46:25, 2022.

[74] Xiaojun Chen, Feiping Nie, Joshua Zhexue Huang, and Min Yang. Scalable normalized cut with improved spectral rotation. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1518–1524. ijcai.org, 2017.

[75] Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In *International conference on machine learning*, pages 1032–1041. PMLR, 2019.

[76] Xue Chen and Eric Price. Active regression via linear-sample sparsification. In *Conference on Learning Theory, COLT*, pages 663–695, 2019.

[77] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2007.

[78] Anshuman Chhabra and Prasant Mohapatra. Fair algorithms for hierarchical agglomerative clustering. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 206–211. IEEE, 2022.

[79] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. *Advances in neural information processing systems*, 30, 2017.

[80] Davin Choo, Christoph Grunau, Julian Portmann, and Vaclav Rozhon. k-means++: few more steps yield constant approximation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1909–1917. PMLR, 13–18 Jul 2020.

[81] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. In *SODA*, pages 291–300. SIAM, 1990.

[82] Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference, STOC*, pages 81–90, 2013.

[83] Edith Cohen, Haim Kaplan, and Uri Zwick. Connection caching. In *Proceedings of the thirty-first annual ACM symposium on Theory of Computing*, pages 612–621, 1999.

[84] Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, page 163–172, New York, NY, USA, 2015. Association for Computing Machinery.

[85] Vincent Cohen-Addad. A fast approximation scheme for low-dimensional k-means. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 430–440, USA, 2018. Society for Industrial and Applied Mathematics.

[86] Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. *J. ACM*, 68(6), Oct 2021.

[87] Vincent Cohen-Addad and Karthik C. S. Inapproximability of clustering in lp metrics. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 519–539, 2019.

[88] Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 353–364, 2016.

[89] Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1038–1051, 2022.

[90] Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Fast and accurate k-means++ via rejection sampling. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16235–16245. Curran Associates, Inc., 2020.

[91] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 169–182. ACM, 2021.

[92] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 169–182, 2021.

[93] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[94] Elliot Creager, David Madras, Jörn-Henrik Jacobsen, Marissa Weis, Kevin Swersky, Toniann Pitassi, and Richard Zemel. Flexibly fair representation learning by disentanglement. In *International conference on machine learning*, pages 1436–1445. PMLR, 2019.

[95] Ryan R Curtin. A dual-tree algorithm for fast k-means clustering with large k. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 300–308. SIAM, 2017.

[96] Chen Dan, Kristoffer Arnsfelt Hansen, He Jiang, Liwei Wang, and Yuchen Zhou. Low rank approximation of binary matrices: Column subset selection and generalizations. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 41:1–41:16, 2018.

[97] Amir Daneshgar and Ramin Javadi. On the complexity of isoperimetric problems on trees. *Discret. Appl. Math.*, 160(1-2):116–131, 2012.

[98] Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W Mahoney. Sampling algorithms and coresets for \ell_p regression. *SIAM Journal on Computing*, 38(5):2060–2078, 2009.

[99] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[100] Erik D Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. On streaming and communication complexity of the set cover problem. In *Distributed Computing: 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings 28*, pages 484–498. Springer, 2014.

[101] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review*, 19(4):1–12, 1989.

[102] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.

[103] Urška Demšar, Olga Špatenková, and Kirsi Virrantaus. Identifying critical locations in a spatial network with graph theory. *Transactions in GIS*, 12(1):61–82, 2008.

[104] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[105] Sybil Derrible and Christopher Kennedy. Applications of graph theory and network science to transit network design. *Transport reviews*, 31(4):495–519, 2011.

[106] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015.

[107] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[108] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 579–587, Lille, France, 07–09 Jul 2015. PMLR.

[109] Jonathan Drake. *Faster k-means clustering.* PhD thesis, Baylor University, 2013.

[110] Julia Dressel and Hany Farid. The accuracy, fairness, and limits of predicting recidivism. *Science advances*, 4(1):eaao5580, 2018.

[111] Petros Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and Vishwanathan Vinay. Clustering large graphs via the singular value decomposition. *Machine learning*, 56:9–33, 2004.

[112] Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

[113] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX and 10th International Workshop on Randomization and Computation, RANDOM, Proceedings*, pages 316–326, 2006.

[114] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-row-based methods. In *Algorithms - ESA 2006, 14th Annual European Symposium, Proceedings*, pages 304–314, 2006.

[115] Flavio du Pin Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. Data pre-processing for discrimination prevention: Information-theoretic optimization and analysis. *IEEE Journal of Selected Topics in Signal Processing*, 12(5):1106–1119, 2018.

[116] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[117] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[118] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 214–226, 2012.

[119] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 147–153. AAAI Press, 2003.

[120] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[121] Guillaume Evanno, Sebastien Regnaut, and Jérôme Goudet. Detecting the number of clusters of individuals using the software structure: a simulation study. *Molecular ecology*, 14(8):2611–2620, 2005.

[122] Dan Feldman. Introduction to core-sets: an updated survey. *arXiv preprint arXiv:2011.09384*, 2020.

[123] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 569–578, 2011.

[124] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657, 2020.

[125] Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 259–268, 2015.

[126] Yu Feng and Greg Hamerly. Pg-means: learning the number of clusters in data. *Advances in neural information processing systems*, 19, 2006.

[127] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[128] Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, and Stefan Szeider. Covering graphs with few complete bipartite subgraphs. *Theor. Comput. Sci.*, 410(21-23):2045–2053, 2009.

[129] Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. *ACM SIGACT News*, 50(2):62–79, 2019.

[130] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Approximation schemes for low-rank binary matrix approximation problems. *ACM Trans. Algorithms*, 16(1):12:1–12:39, 2020.

[131] Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a ptas for k-means in doubling metrics. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 365–374, 2016.

[132] Yinghua Fu, Nianping Jiang, and Hong Sun. Binary matrix factorization and consensus algorithms. In *2010 International Conference on Electrical and Control Engineering*, pages 4563–4567. IEEE, 2010.

[133] Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data clustering: theory, algorithms, and applications*. SIAM, 2020.

[134] Sumit Ganguly. Estimating frequency moments of data streams using random linear combinations. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 369–380. Springer, 2004.

[135] Mehrdad Ghadiri, Samira Samadi, and Santosh Vempala. Socially fair k-means clustering. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 438–448, 2021.

[136] Nicolas Gillis. The why and how of nonnegative matrix factorization. *Regularization, optimization, kernels, and support vector machines*, 12(257):257–291, 2014.

[137] Nicolas Gillis and Stephen A. Vavasis. On the complexity of robust PCA and ℓ_1-norm low-rank matrix approximation. *Math. Oper. Res.*, 43(4):1072–1084, 2018.

[138] Rachel Goodman. Why amazon's automated hiring tool distcriminated against women. *American Civil Liberties Union*, 2018.

[139] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. The expander hierarchy and its applications to dynamic graph algorithms. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2212–2228. SIAM, 2021.

[140] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[141] Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for Euclidean k-means. *Inf. Process. Lett.*, 176(C), jun 2022.

[142] Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[143] Chengcheng Guo, Bo Zhao, and Yanbing Bai. Deepcore: A comprehensive library for coreset selection in deep learning. In *International Conference on Database and Expert Systems Applications*, pages 181–195. Springer, 2022.

[144] Harold W. Gutch, Peter Gruber, Arie Yeredor, and Fabian J. Theis. ICA over finite fields - separability and algorithms. *Signal Process.*, 92(8):1796–1808, 2012.

[145] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1325–1338, 2022.

[146] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE intelligent systems*, 24(2):8–12, 2009.

[147] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 130–140. SIAM, 2010.

[148] Kathrin Hanauer, Monika Henzinger, Robin Münk, Harald Räcke, and Maximilian Vötsch. Expander hierarchies for normalized cuts on graphs. *CoRR*, abs/2406.14111, 2024.

[149] Kathrin Hanauer, Monika Henzinger, Robin Münk, Harald Räcke, and Maximilian Vötsch. Xcut/xcut v1.0.0. `https://doi.org/10.5281/zenodo.12108189`, 2024.

[150] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms–a quick reference guide. *ACM Journal of Experimental Algorithmics*, 27:1–45, 2022.

[151] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 291–300, New York, NY, USA, 2004. Association for Computing Machinery.

[152] Elfarouk Harb and Ho Shan Lam. Kfc: A scalable approximation algorithm for k-center fair clustering. *Advances in neural information processing systems*, 33:14509–14519, 2020.

[153] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. *Advances in neural information processing systems*, 29, 2016.

[154] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proc. VLDB Endow.*, 2(1):1282–1293, aug 2009.

[155] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *Journal of the ACM*, 69(6):1–14, 2022.

[156] Bruce Hendrickson and Robert W. Leland. A multi-level algorithm for partitioning graphs. In Sidney Karin, editor, *Proceedings Supercomputing '95, San Diego, CA, USA, December 4-8, 1995*, page 28. ACM, 1995.

[157] Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Dynamic maintenance of monotone dynamic programs and applications. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 36:1–36:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[158] Monika Henzinger, Alexander Noe, and Christian Schulz. Practical fully dynamic minimum cut algorithms. In *2022 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 13–26. SIAM, 2022.

[159] Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. *SIAM Journal on Computing*, 49(1):1–36, 2020.

[160] Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

[161] Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. Maintaining expander decompositions via sparse cuts. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 48–69. SIAM, 2023.

[162] SHI Huaizhou, R Venkatesha Prasad, Ertan Onur, and IGMM Niemegeers. Fairness in wireless networks: Issues, measures and challenges. *IEEE Communications Surveys & Tutorials*, 16(1):5–24, 2013.

[163] Lingxiao Huang, Shaofeng Jiang, and Nisheeth Vishnoi. Coresets for clustering with fairness constraints. *Advances in neural information processing systems*, 32, 2019.

[164] Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1416–1429, 2020.

[165] Zan Huang, Wingyan Chung, and Hsinchun Chen. A graph model for e-commerce recommender systems. *Journal of the American Society for information science and technology*, 55(3):259–274, 2004.

[166] Zan Huang, Wingyan Chung, Thian-Huat Ong, and Hsinchun Chen. A graph-based recommender system for digital library. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 65–73, 2002.

[167] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 261–276, 2009.

[168] Jennifer Jang and Heinrich Jiang. Dbscan++: Towards fast and scalable density clustering. In *International conference on machine learning*, pages 3019–3029. PMLR, 2019.

[169] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[170] Weikuan Jia, Meili Sun, Jian Lian, and Sujuan Hou. Feature dimensionality reduction: a review. *Complex & Intelligent Systems*, 8(3):2663–2693, 2022.

[171] Peng Jiang, Jiming Peng, Michael Heath, and Rui Yang. A clustering approach to constrained binary matrix factorization. In *Data Mining and Knowledge Discovery for Big Data*, pages 281–303. Springer, 2014.

[172] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[173] Matthew Joseph, Michael Kearns, Jamie H Morgenstern, and Aaron Roth. Fairness in learning: Classic and contextual bandits. *Advances in neural information processing systems*, 29, 2016.

[174] Mohammad Mahdi Kamani, Farzin Haddadpour, Rana Forsati, and Mehrdad Mahdavi. Efficient fair principal component analysis. *Machine Learning*, pages 1–32, 2022.

[175] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and information systems*, 33(1):1–33, 2012.

[176] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. Fairness-aware classifier with prejudice remover regularizer. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part II 23*, pages 35–50. Springer, 2012.

[177] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[178] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[179] Ravi Kannan and Santosh S. Vempala. Spectral algorithms. *Found. Trends Theor. Comput. Sci.*, 4(3-4):157–288, 2009.

[180] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[181] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine Piatko, Ruth Silverman, and Angela Y. Wu. The analysis of a simple k-means clustering algorithm. In *Proceedings of the Sixteenth Annual Symposium on Computational Geometry*, SCG '00, page 100–109, New York, NY, USA, 2000. Association for Computing Machinery.

[182] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004.

[183] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM journal on applied mathematics*, 37(3):513–538, 1979.

[184] Anna R Karlin, Mark S Manasse, Lyle A McGeoch, and Susan Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

[185] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 244–254, 1986.

[186] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[187] 2004 KDD Cup. Protein homology dataset. *available at https://osmot.cs.cornell.edu/kddcup/datasets.html*, 2004.

[188] Qifa Ke and Takeo Kanade. Robust subspace computation using l1 norm, 2003.

[189] Qifa Ke and Takeo Kanade. Robust l_1 norm factorization in the presence of outliers and missing data by alternative convex programming. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 739–746, 2005.

[190] Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *International conference on machine learning*, pages 2564–2572. PMLR, 2018.

[191] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226. SIAM, 2014.

[192] Rohit Khandekar, Subhash Khot, Lorenzo Orecchia, and Nisheeth K Vishnoi. On a cut-matching game for the sparsest cut problem. *Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2007-177*, 6(7):12, 2007.

[193] Jon Kleinberg, Jens Ludwig, Sendhil Mullainathan, and Ashesh Rambachan. Algorithmic fairness. In *Aea papers and proceedings*, volume 108, pages 22–27. American Economic Association 2014 Broadway, Suite 305, Nashville, TN 37203, 2018.

[194] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *arXiv preprint arXiv:1609.05807*, 2016.

[195] Matthäus Kleindessner, Pranjal Awasthi, and Jamie Morgenstern. Fair k-center clustering for data summarization. In *International Conference on Machine Learning*, pages 3448–3457. PMLR, 2019.

[196] Matthäus Kleindessner, Michele Donini, Chris Russell, and Muhammad Bilal Zafar. Efficient fair pca for fair representation learning. In *International Conference on Artificial Intelligence and Statistics*, pages 5250–5270. PMLR, 2023.

[197] Stavros G. Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the Euclidean k-median problem. In Jaroslav Nešetřil, editor, *Algorithms - ESA' 99*, pages 378–389, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[198] Elias Koutsoupias. The k-server problem. *Comput. Sci. Rev.*, 3(2):105–118, 2009.

[199] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[200] Emmanuel Kowalski. *An introduction to expander graphs*. Société mathématique de France Paris, 2019.

[201] Mehmet Koyutürk and Ananth Grama. PROXIMUS: a framework for analyzing very high dimensional discrete-attributed datasets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 147–156, 2003.

[202] Ravi Kumar, Rina Panigrahy, Ali Rahimi, and David P. Woodruff. Faster algorithms for binary matrix factorization. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 3551–3559, 2019.

[203] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. Counterfactual fairness. *Advances in neural information processing systems*, 30, 2017.

[204] Nojun Kwak. Principal component analysis based on l1-norm maximization. *IEEE transactions on pattern analysis and machine intelligence*, 30(9):1672–1680, 2008.

[205] Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 09–15 Jun 2019.

[206] Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Inf. Process. Lett.*, 120:40–43, 2017.

[207] Jason Li. Deterministic mincut in almost-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 384–395, 2021.

[208] Jia Li, Honglei Zhang, Zhichao Han, Yu Rong, Hong Cheng, and Junzhou Huang. Adversarial attack on community detection by hiding individuals. In Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen, editors, *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pages 917–927. ACM / IW3C2, 2020.

[209] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 901–910, New York, NY, USA, 2013. Association for Computing Machinery.

[210] Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 174–183, 2014.

[211] Weiwei Liu, Xiaobo Shen, and Ivor Tsang. Sparse embedded k-means clustering. *Advances in neural information processing systems*, 30, 2017.

[212] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[213] Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri, and Yuan Hong. Constraint-aware role mining via extended boolean matrix decomposition. *IEEE Trans. Dependable Secur. Comput.*, 9(5):655–669, 2012.

[214] J Macqueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*, 1967.

[215] Malik Magdon-Ismail. Row sampling for matrix algorithms via a non-commutative bernstein bound. *CoRR*, abs/1008.0587, 2010.

[216] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. *Theoretical Computer Science*, 442:13–21, 2012.

[217] Arvind V. Mahankali and David P. Woodruff. Optimal ℓ_1 column subset selection and a fast PTAS for low rank approximation. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 560–578, 2021.

[218] Michael W. Mahoney. Randomized algorithms for matrices and data. *Found. Trends Mach. Learn.*, 3(2):123–224, 2011.

[219] Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 1027–1038, New York, NY, USA, 2019. Association for Computing Machinery.

[220] Karima Makhlouf, Sami Zhioua, and Catuscia Palamidessi. Survey on causal-based machine learning fairness notions. *arXiv preprint arXiv:2010.09553*, 2020.

[221] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333. ACM, 1988.

[222] Panos P. Markopoulos, George N. Karystinos, and Dimitrios A. Pados. Optimal algorithms for l_1-subspace signal processing. *IEEE Trans. Signal Process.*, 62(19):5046–5058, 2014.

[223] Ivan Markovsky. *Low rank approximation: algorithms, implementation, applications*, volume 906. Springer, 2012.

[224] Jiří Matoušek. On variants of the johnson–lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

[225] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1-6):816–825, jun 1991.

[226] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 54(6):1–35, 2021.

[227] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22–es, 2007.

[228] Raphael A. Meyer, Cameron Musco, Christopher Musco, David P. Woodruff, and Samson Zhou. Fast regression for structured inputs. In *The Tenth International Conference on Learning Representations, ICLR*, 2022.

[229] Raphael A. Meyer, Cameron Musco, Christopher Musco, David P. Woodruff, and Samson Zhou. Near-linear sample complexity for l_p polynomial regression. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 3959–4025, 2023.

[230] Adam Meyerson, Akash Nanavati, and Laura Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 954–959. Citeseer, 2006.

[231] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE transactions on knowledge and data engineering*, 20(10):1348–1362, 2008.

[232] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[233] Bojan Mohar. Isoperimetric numbers of graphs. *J. Comb. Theory, Ser. B*, 47(3):274–291, 1989.

[234] Andrew W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, page 397–405, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[235] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[236] Cameron Musco, Christopher Musco, David P. Woodruff, and Taisuke Yasuda. Active linear regression for ℓ_p norms and beyond. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 744–753, 2022.

[237] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o (n1/2-ε)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1122–1129, 2017.

[238] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 950–961. IEEE, 2017.

[239] Maria CV Nascimento and Andre CPLF De Carvalho. Spectral methods for graph clustering–a survey. *European Journal of Operational Research*, 211(2):221–231, 2011.

[240] Maxim Naumov and Timothy Moon. Parallel spectral graph partitioning. *NVIDIA, Santa Clara, CA, USA, Tech. Rep., NVR-2016-001*, 2016.

[241] Maryam Negahbani and Deeparnab Chakrabarty. Better algorithms for individually fair k-clustering. *Advances in Neural Information Processing Systems*, 34:13340–13351, 2021.

[242] Stefan Neumann. Bipartite stochastic block models with tiny clusters. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS*, pages 3871–3881, 2018.

[243] James Newling and Francois Fleuret. Fast k-means with accurate bounds. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 936–944, New York, New York, USA, 20–22 Jun 2016. PMLR.

[244] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 849–856. MIT Press, 2001.

[245] HP Ng, SH Ong, KWC Foong, Poh-Sun Goh, and WL Nowinski. Medical image segmentation using k-means clustering and improved watershed algorithm. In *2006 IEEE southwest symposium on image analysis and interpretation*, pages 61–65. IEEE, 2006.

[246] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE transactions on knowledge and data engineering*, 14(5):1003–1016, 2002.

[247] Feiping Nie, Jitao Lu, Danyang Wu, Rong Wang, and Xuelong Li. A novel normalized-cut solver with nearest neighbor hierarchical initialization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(1):659–666, 2024.

[248] Liadan O'callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings 18th international conference on data engineering*, pages 685–694. IEEE, 2002.

[249] James Orlin. Contentment in graph theory: covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977.

[250] Vitaly Osipov and Peter Sanders. n-level graph partitioning. In *Algorithms–ESA 2010: 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I 18*, pages 278–289. Springer, 2010.

[251] Amichai Painsky, Saharon Rosset, and Meir Feder. Generalized independent component analysis over finite alphabets. *IEEE Transactions on Information Theory*, 62(2):1038–1053, 2015.

[252] Amichai Painsky, Saharon Rosset, and Meir Feder. Linear independent component analysis over finite fields: Algorithms and bounds. *IEEE Transactions on Signal Processing*, 66(22):5875–5886, 2018.

[253] Young Woong Park and Diego Klabjan. Three iteratively reweighted least squares algorithms for l_1-norm principal component analysis. *Knowledge and Information Systems*, 54(3):541–565, 2018.

[254] Aditya Parulekar, Advait Parulekar, and Eric Price. L1 regression with lewis weights subsampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 49:1–49:21, 2021.

[255] Amandalynne Paullada, Inioluwa Deborah Raji, Emily M Bender, Emily Denton, and Alex Hanna. Data and its (dis) contents: A survey of dataset development and use in machine learning research. *Patterns*, 2(11), 2021.

[256] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[257] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[258] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, page 277–281, New York, NY, USA, 1999. Association for Computing Machinery.

[259] Dan Pelleg, Andrew Moore, et al. X-means: Extending k-means with e cient estimation of the number of clusters. In *ICML'00*, pages 727–734. Citeseer, 2000.

[260] Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! *SIAM J. Comput.*, 46(2):710–743, 2017.

[261] James Philbin. *Scalable object retrieval in very large image collections*. PhD thesis, Oxford University, 2010.

[262] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[263] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. *Advances in neural information processing systems*, 30, 2017.

[264] Feng Qian, Abhinav Pathak, Yu Charlie Hu, Zhuoqing Morley Mao, and Yinglian Xie. A case for unsupervised-learning-based spam filtering. *SIGMETRICS Perform. Eval. Rev.*, 38(1):367–368, jun 2010.

[265] Harald Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.

[266] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 255–264, 2008.

[267] Bozidar Radunovic and Jean-Yves Le Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Transactions on networking*, 15(5):1073–1083, 2007.

[268] Siamak Ravanbakhsh, Barnabás Póczos, and Russell Greiner. Boolean matrix factorization and noisy completion via message passing. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 945–954, 2016.

[269] Ilya P. Razenshteyn, Zhao Song, and David P. Woodruff. Weighted low rank approximations with provable guarantees. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 250–263, 2016.

[270] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[271] M Ali Rostami, Alieh Saeedi, Eric Peukert, and Erhard Rahm. Interactive visualization of large similarity graphs and entity resolution clusters. In *EDBT*, pages 690–693, 2018.

[272] Tim Roughgarden. Cs369e: Communication complexity (for algorithm designers) lecture# 1: Data streams: Algorithms and lower bounds. 2015.

[273] Tapasmini Sahoo and Kunal Kumar Das. Brain tumor localization using n-cut. *International Journal of Online & Biomedical Engineering*, 19(15), 2023.

[274] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, pages 4–11, 2014.

[275] Samira Samadi, Uthaipon Tantipongpipat, Jamie H Morgenstern, Mohit Singh, and Santosh Vempala. The price of fair pca: One extra dimension. *Advances in neural information processing systems*, 31, 2018.

[276] Peter Sanders. Algorithm engineering–an attempt at a definition. In *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 321–340. Springer, 2009.

[277] Peter Sanders and Christian Schulz. High quality graph partitioning. *Graph Partitioning and Graph Clustering*, 588(1):1–17, 2012.

[278] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2616–2635. SIAM, 2019.

[279] Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 737–746, 2009.

[280] Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In *Approximation and Online Algorithms: 17th International Workshop, WAOA 2019, Munich, Germany, September 12–13, 2019, Revised Selected Papers 17*, pages 232–251. Springer, 2020.

[281] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[282] Erich Schubert and Peter J Rousseeuw. Faster k-medoids clustering: improving the pam, clara, and clarans algorithms. In *Similarity Search and Applications: 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2–4, 2019, Proceedings 12*, pages 171–187. Springer, 2019.

[283] John Scott. *What is social network analysis?* Bloomsbury Academic, 2012.

[284] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 1177–1178, New York, NY, USA, 2010. Association for Computing Machinery.

[285] Jouni K. Seppänen, Ella Bingham, and Heikki Mannila. A simple algorithm for topic identification in 0-1 data. In *Knowledge Discovery in Databases: PKDD 2003, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Proceedings*, pages 423–434, 2003.

[286] Mina Sheikhalishahi, Andrea Saracino, Mohamed Mejri, Nadia Tawbi, and Fabio Martinelli. Fast and effective clustering of spam emails based on structural similarity. In Joaquin Garcia-Alfaro, Evangelos Kranakis, and Guillaume Bonfante, editors, *Foundations and Practice of Security*, pages 195–211, Cham, 2016. Springer International Publishing.

[287] Shubhanshu Shekhar, Greg Fields, Mohammad Ghavamzadeh, and Tara Javidi. Adaptive sampling for minimax fair classification. *Advances in Neural Information Processing Systems*, 34:24535–24544, 2021.

[288] Bao-Hong Shen, Shuiwang Ji, and Jieping Ye. Mining discrete patterns via binary matrix factorization. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 757–766, 2009.

[289] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[290] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[291] Chiappa Silvia, Jiang Ray, Stepleton Tom, Pacchiano Aldo, Jiang Heinrich, and Aslanides John. A general approach to fairness with optimal transport. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3633–3640, 2020.

[292] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 177–189. PMLR, 2022.

[293] Tomás Singliar and Milos Hauskrecht. Noisy-or component analysis and its application to link analysis. *J. Mach. Learn. Res.*, 7:2189–2213, 2006.

[294] Samarth Sinha, Han Zhang, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, and Augustus Odena. Small-gan: Speeding up gan training using core-sets. In *International Conference on Machine Learning*, pages 9005–9015. PMLR, 2020.

[295] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[296] Christian Sohler and David P Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 802–813. IEEE, 2018.

[297] Qinbao Song, Jingjie Ni, and Guangtao Wang. A fast clustering-based feature subset selection algorithm for high-dimensional data. *IEEE transactions on knowledge and data engineering*, 25(1):1–14, 2011.

[298] Zhao Song, David P. Woodruff, and Peilin Zhong. Low rank approximation with entrywise ℓ_1-norm error. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 688–701, 2017.

[299] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

[300] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[301] Hugo Steinhaus et al. Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci*, 1(804):801, 1956.

[302] Catherine A Sugar and Gareth M James. Finding the number of clusters in a dataset: An information-theoretic approach. *Journal of the American Statistical Association*, 98(463):750–763, 2003.

[303] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[304] Shazia Tabassum, Fabiola SF Pereira, Sofia Fernandes, and João Gama. Social network analysis: An overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(5):e1256, 2018.

[305] Kunal Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 281–290, New York, NY, USA, 2004. Association for Computing Machinery.

[306] Leandros Tassiulas and Saswati Sarkar. Maxmin fair scheduling in wireless networks. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 763–772. IEEE, 2002.

[307] Srujan Teja Thomdapu, Palash Katiyar, and Ketan Rajawat. Dynamic cache management in content delivery networks. *Computer Networks*, 187:107822, 2021.

[308] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[309] Mo Tiwari, Martin J Zhang, James Mayclin, Sebastian Thrun, Chris Piech, and Ilan Shomorony. Banditpam: Almost linear time k-medoids clustering via multi-armed bandits. *Advances in Neural Information Processing Systems*, 33:10211–10222, 2020.

[310] Berk Ustun, Yang Liu, and David Parkes. Fairness without harm: Decoupled classifiers with preference guarantees. In *International Conference on Machine Learning*, pages 6373–6382. PMLR, 2019.

[311] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 175–184, 2007.

[312] Andrea Vattani. K-means requires exponentially many iterations even in the plane. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 324–332, 2009.

[313] Erik Waingarten. Notes for algorithms for big data: Clustering. `https://drive.google.com/file/d/1T5YYGrA3kdi4_QGvF3c_foOZvXMDaRPw/view`.

[314] Chris Walshaw, Mark Cross, and Martin G. Everett. A localized algorithm for optimizing unstructured mesh partitions. *Int. J. High Perform. Comput. Appl.*, 9(4):280–295, 1995.

[315] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[316] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *The world wide web conference*, pages 3307–3313, 2019.

[317] Jing Wang, Jingdong Wang, Qifa Ke, Gang Zeng, and Shipeng Li. Fast approximate k-means via cluster closures. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3037–3044, 2012.

[318] Yangtao Wang, Xi Shen, Yuan Yuan, Yuming Du, Maomao Li, Shell Xu Hu, James L. Crowley, and Dominique Vaufreydaz. Tokencut: Segmenting objects in images and videos with self-supervised transformer and normalized cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(12):15790–15801, 2023.

[319] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[320] Yifan Wang, Weizhi Ma, Min Zhang, Yiqun Liu, and Shaoping Ma. A survey on the fairness of recommender systems. *ACM Transactions on Information Systems*, 41(3):1–43, 2023.

[321] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10(1-2):1–157, 2014.

[322] Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1130–1143, 2017.

[323] Eric P. Xing and Richard M. Karp. CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. In *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology, July 21-25, 2001, Copenhagen, Denmark*, pages 306–315, 2001.

[324] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[325] Arie Yeredor. Independent component analysis over galois fields of prime order. *IEEE Trans. Inf. Theory*, 57(8):5342–5359, 2011.

[326] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[327] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. *arXiv preprint arXiv:2106.01085*, 2021.

[328] Neal E. Young. On-line file caching. In *SODA*, pages 82–86. ACM/SIAM, 1998.

[329] Stella X. Yu and Jianbo Shi. Multiclass spectral clustering. In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*, pages 313–319. IEEE Computer Society, 2003.

[330] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rogriguez, and Krishna P Gummadi. Fairness constraints: Mechanisms for fair classification. In *Artificial intelligence and statistics*, pages 962–970. PMLR, 2017.

[331] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *International conference on machine learning*, pages 325–333. PMLR, 2013.

[332] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma. Learning to cluster web search results. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217, 2004.

[333] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340, 2018.

[334] Jin Zhang, Lei Xie, Wei Feng, and Yanning Zhang. A subword normalized cut approach to automatic story segmentation of chinese broadcast news. In Gary Geunbae Lee, Dawei Song, Chin-Yew Lin, Akiko N. Aizawa, Kazuko Kuriyama, Masaharu Yoshioka, and Tetsuya Sakai, editors, *Information Retrieval Technology, 5th Asia Information Retrieval Symposium, AIRS 2009, Sapporo, Japan, October 21-23, 2009. Proceedings*, volume 5839 of *Lecture Notes in Computer Science*, pages 136–148. Springer, 2009.

[335] Zhongyuan Zhang, Tao Li, Chris Ding, and Xiangsun Zhang. Binary matrix factorization with applications. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 391–400. IEEE, 2007.

[336] Han Zhao and Geoffrey J Gordon. Inherent tradeoffs in learning fair representations. *Journal of Machine Learning Research*, 23(57):1–26, 2022.

[337] Zhiqiang Zhao, Yongyu Wang, and Zhuo Feng. Nearly-linear time spectral graph reduction for scalable graph partitioning and data visualization. *arXiv preprint arXiv:1812.08942*, 2018.

[338] Yinqiang Zheng, Guangcan Liu, Shigeki Sugimoto, Shuicheng Yan, and Masatoshi Okutomi. Practical low-rank matrix approximation under robust l1-norm. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1410–1417, 2012.

[339] Marinka Zitnik and Blaz Zupan. Nimfa: A python library for nonnegative matrix factorization. *Journal of Machine Learning Research*, 13:849–853, 2012.