New Algorithmic Tools for Distributed Similarity Search and Edge Estimation

Cyrus Rashtchian

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2018

Reading Committee:

Paul Beame, Chair

Magdalena Balazinska

Luis Ceze

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

New Algorithmic Tools for Distributed
Similarity Search and Edge Estimation

Cyrus Rashtchian

Chair of the Supervisory Committee:
Professor Paul Beame
Computer Science and Engineering

We present several foundational results on computational questions related to similarity search, clustering, and parameter estimation. The problems center around the theme of improving algorithms by utilizing geometric or graphical structure. Some contributions include:

- Improved upper and lower bounds for computing a similarity join under Hamming distance in a simultaneous distributed model. The core of our analysis involves novel connections between similarity joins and extremal graph theory.

- An edge-isoperimetric inequality for powers of the binary hypercube. The insights here help us to develop new similarity join algorithms that are nearly-optimal for a theoretical MapReduce model.

- A distributed clustering algorithm for edit distance, with applications to DNA data storage. By using random structure found in real datasets, we achieve new hashing, embedding, and convergence results for an otherwise challenging clustering problem.

- The first polylogarithmic query algorithm for estimating the number of edges in a graph using a natural graph query. Our randomized, adaptive algorithm uses bipartite independent set queries to quickly learn an unknown graph.

TABLE OF CONTENTS

Page

List of Figures iii

Chapter 1: Introduction 1
1.1 Clustering in Edit Distance for DNA Data Storage 4
1.2 Communication-Efficient Similarity Join 7
1.3 Edge-Isoperimetry 9
1.4 Distance Correlations 11
1.5 Edge Estimation with Independent Set Oracles 12
1.6 Outline of Thesis 14

Chapter 2: Background on Distributed Similarity Search 15
2.1 Gentle Overview of Metric Spaces 15
2.2 Flavors of Similarity Search 18
2.3 Clustering 20
2.4 Locality Sensitive Hashing (LSH) 23
2.5 Random Walks and a Conjecture of Erdős-Simonovitz and Sidorenko 24
2.6 Concentration Bounds 27

Chapter 3: Hamming Similarity Join 29
3.1 Introduction 29
3.2 Related Work 31
3.3 Similarity Graphs, Edge-Coverings, and Overhead 34
3.4 Our Results 38
3.5 Randomized Edge-Coverings using Edge-Isoperimetric Sets 40
3.6 The Lower Bound 46
3.7 Discussion and Future Work 59

Chapter 4: Approximate Edge-isoperimetric Inequalities . . . 61
4.1 Introduction . . . 61
4.2 The distance two case . . . 67
4.3 The general case for even r . . . 69
4.4 The general case for odd r . . . 76
4.5 Tightness . . . 76
4.6 Conclusion . . . 77

Chapter 5: Clustering in Edit Distance for DNA Storage . . . 78
5.1 DNA Data Storage Model and Preliminaries . . . 85
5.2 Our Algorithm . . . 90
5.3 Algorithm Convergence and Hash Analysis . . . 92
5.4 Clusters are Well-Separated Under Binary Signatures . . . 101
5.5 Outlier Robustness . . . 106
5.6 Experiments . . . 107
5.7 Related Work . . . 114
5.8 Conclusion . . . 116

Chapter 6: Edge Estimation with Independent Set Queries . . . 117
6.1 Introduction . . . 117
6.2 Preliminaries . . . 129
6.3 Edge sparsification by random coloring . . . 135
6.4 Edge estimation using BIS queries . . . 139
6.5 Edge estimation using IS queries . . . 151
6.6 Conclusions . . . 158

Chapter 7: Conclusion and Future Work . . . 159
7.1 Summary of Thesis . . . 159
7.2 Better Embeddings for Edit Distance . . . 159
7.3 Larger Context: Data Science Algorithms . . . 161

Bibliography . . . 163

LIST OF FIGURES

Figure Number Page

1.1 Depiction of the DNA data storage pipeline 5
1.2 Simultaneous protocol for a distributed similarity join 7
1.3 Distance one similarity graph for $\{0,1\}^4$ and a similarity join 10
1.4 Group testing, independent set, bipartite independent set oracles 12
2.1 Example of nearest neighbor search and similarity join 18
5.1 Well-separated clusters in DNA storage datasets 79
5.2 Empirical comparisons between our algorithm and Starcode 110
5.3 Empirical results for our algorithm's convergence and running time 111
6.1 Depiction of our edge estimation algorithm 124

ACKNOWLEDGMENTS

Contrary to popular belief, a Ph.D. turned out to be far from an individual effort, and many people contributed to my growth throughout the long journey.

First and foremost, I would like to thank my advisor, Paul Beame, for his tireless efforts over the years. His patient guidance has been fundamental in transforming me from a young grad student to an (almost) independent researcher. In retrospect, I'm undeniably impressed with his ability to push me when I'm stuck and leave me alone when I need space. Thanks to him I've matured in uncountable ways as a thinker, writer, editor, and speaker. I strive even now to reach his extraordinary efficiency and clarity in reasoning and writing. Every aspect of this thesis has been positively influenced by his research taste and intellectual ambition.

I also want to thank the rest of my committee, Magda Balazinska, Mark Oskin, and Jevin West. You've been much more integral in my development than you probably realize, well beyond the formalities, and you've taught me so much by example.

The theory group has been an uplifting and dependable academic home in CSE. The faculty, especially Anna Karlin, Anup Rao, Thomas Rothvoß, and Shayan Oveis Gharan, have taught me a lot about good communication and relentless research goals. I've enjoyed the camaraderie of the current and former students: Alex, Alireza, Becca, Elaine, Jeffrey, Jiachen, Kira, Makrand, Mert, Mohammad, Paris, Robbie, Sami, Siva, Swati, Vincent, Widad, and Xin. You've all been a great sounding board for ideas large and small, as well as a huge inspiration.

Microsoft Research has been a second academic home, and my time there has been formative and enjoyable in many ways. I'm grateful to Karin Strauss for being

a wonderful manager, collaborator, and mentor. I also want to thank Luis Ceze for his mentorship and collaboration, and for inviting me to join the DNA Data Storage project when I needed a new direction. Thanks to Kostya Makarychev for being a core collaborator on the clustering algorithm and for teaching me a lot about theory in general; and, Sergey Yekahnin, who's astute vision and dedicated efforts helped glue the team together. I really appreciate having the opportunity to work with Siena Dumas Ang, Miki Rácz, and Djordje Jevdjic. Lastly, I want to thank the members of the MLO group, including Ilya Razenshteyn for many great conversations, and Sébastien Bubeck and Yuval Peres for being brilliant, inspiring, and remarkably welcoming.

In CSE, I'm indebted to Elise deGoede Dorough and Lindsay Michimoto for being phenomenal graduate program advisors, helping with many formalities, and offering support throughout the years. I also want to thank Sam Sudar for being a great office mate, and Adrian Sampson and Abe Friesen for infinite useful advice.

Going back to my time at UIUC, I want to acknowledge Julia Hockenmaier for getting me started down the research path, and for her tolerant guidance over those years, teaching me the basics of academic writing, speaking, coding, and NLP. I would also like to thank Graham Heimberg and Saarah Malik for being amazing friends and partners in crime, and I'm so glad we still keep in touch.

I'm also very grateful to my additional coauthors. I appreciate Sariel Har-Peled for showing me how jokes and a bar of chocolate can lead to longer and more productive research meetings. I acknowledge David Ellis for being my main collaborator on the edge-isoperimetric results. Although our paper didn't make it in this thesis, I want to recognize Shay Moran for the enjoyable collaboration and much good advice.

Outside of the academic world, I want to thank my fantastic friends in Seattle for making it fun and keeping me sane: Angie, Bridget, Camille, Chris, Dave, Genevieve and Benjamin, Hannah, Hannah, Jack, Julia, John, Kim, Lindsey, Patrick, Sarah,

Stephanie, ... I'm sure I'm forgetting people, and you're awesome, too! Special shout outs to Dan Bida: I've enjoyed all of our roommating, cooking, brewing, hosting, and traveling adventures, and I appreciate the many layers of support over the years; and to Mike Udelhofen, the third boyz.

Music has been a big aid in my productivity. Much of my concentrated thinking and writing happened while listening to Caribou's album *Swim* and Marek Hemmann's album *In Between*, and Kuinka always instantly created an uplifting mood.

My family is probably the main reason I went on to get a Ph.D. in the first place. I want to thank my father, Hassan, for getting me started on an intellectual path, for all the guidance over the years, and for teaching me more than anyone else about the world. And I want to thank my mother, Anastasia, for always believing in me, for teaching me to follow my dreams and to never settle, and for helping me to remember to always look for the best in people.

Last but not least, I want to express my immense gratitude to Alyshia Olsen for being a loving, thoughtful, and very reasonable partner for the past several years. In countless ways, your support and companionship has helped me grow as a person, and overall, you've made so much of this possible, and way more delicious.

DEDICATION

To Baba,
a beacon of truth and wisdom,
especially in trying times.

Chapter 1

INTRODUCTION

Theoretical computer science research develops the algorithmic foundations of central computational problems. It also illuminates connections between these problems and other areas of science and mathematics. In this era of data-driven decision making, it is pertinent to understand the algorithmic questions that underly important data science tasks.

The motivation often begins with a common story: the textbook algorithm works perfectly, but the running time becomes unbearable on data of real interest. When the input fits in main memory on an average computer, most problems in P have suitable solutions. In contrast, when the input size n grows large, even algorithms with a running time of $\Theta(n^2)$ require days or weeks to complete. Quadratic-time algorithms typically arise when the input structure or objective function centers around pairwise relationships. For example, the goal of clustering is to partition a dataset into smaller groups based on the distances between pairs of elements. Similarity search finds pairs of close objects or vectors. Graphs on n vertices have up to $\binom{n}{2}$ edges. Database joins combine tables based on pairs of records that share a key.

When algorithms take quadratic time and datasets get larger, it is natural to wonder: Are there exact or approximate methods that need only nearly-linear time? If not in general, are there assumptions on the data or trade-offs in the results that enable extremely efficient algorithms for relevant instances? This thesis explores the complexity of fundamental computational problems that involve pairwise relationships. By utilizing randomness and approximation, we replace $O(n^2)$ with much smaller bounds. We aim for results that take advantage of specific properties of the problem instance (*e.g.*, dimensionality, distance scale, or noise rate) or the computational model (*e.g.*, number of processors or oracle accesses).

A main theme of this thesis concerns the similarity of objects, along with associated algorithmic tasks. Example problems include clustering DNA sequences or finding all close pairs of vectors in a dataset. These fall under the larger umbrella of *similarity search*, which also includes query-based tasks, such as finding the nearest neighbors of a particular object. A standard first step in many similarity search pipelines involves transforming complicated objects to simpler representations. A common approach is to generate a high-dimensional *feature vector*, which summarizes a large number of statistics about individual elements in a dataset. For example, we may reduce an image to the identity and attributes of depicted objects. Now, vector similarity serves as a surrogate for true image similarity, and it suffices to develop algorithms for finding close pairs of vectors. For this reason, similarity search tasks typically revolve around vectors and the distances between them.

Besides vector-based approaches, another paradigm involves the use of graphs as a way to represent the similarity between objects. Taking a birds-eye view, we can generate an undirected graph that connects every pair of similar items in a dataset. Alternatively, if the input already exhibits graphical structure, then we can use the edges as our notion of similarity between pairs of vertices. A graph-based viewpoint has many advantages. Purely combinatorial results may shed new light on otherwise complicated tasks. Properties of real datasets can inspire novel models for theoretical study. In general, adding this level of abstraction helps connect new problems to previously studied areas of research.

Regardless of the objective or representation, similarity search problems face many computational challenges. The naive solution to many tasks involves comparing all pairs of items, or comparing a query to all objects in a dataset. In this area, the number of pairwise comparisons often dictates the asymptotic running time. Therefore, the goal is to develop algorithms that make many fewer comparisons, while still provably achieving certain guarantees, such as producing very few false negatives.

Since feature vectors contain hundreds or thousands of dimensions, and the distance function varies across problems, it has been difficult to develop a unified understanding of similarity search tasks. In particular, this regime leads to a very different playing field than

geometric algorithms for $\mathbb{R}^2$ or $\mathbb{R}^3$, where computational problems are better understood. Nonetheless, many significant innovations enable tractability. We will explore when more complicated distances can be well-approximated by simpler ones, and how techniques that work well for some distances transfer over to others.

Even with favorable theoretical guarantees, many practical hurdles arise. Data must be stored somewhere that is easy to access. If a processor has enough memory to hold the whole dataset, then there is simply a one time cost for loading the data from disk. For smaller devices or larger datasets, it is impossible to store or access the whole input, and different models of computing take the stage. One alternative is to offload the intense computation to the cloud. Another option is to connect several processors and keep the data in memory, but distributed across physical locations. In both cases, the *communication* between devices becomes a serious bottleneck. For many problems, vastly new techniques are needed to design scalable algorithms, and new trade-offs emerge based on the number of devices.

For very large datasets, accessing the data multiple times may be a concern or limitation. When we are only interested in certain statistics, it may be possible to accurately estimate these without looking at all of the data. For example, is it possible to approximate the number of edges or triangles in a graph by examining only a small portion of the graph? The *oracle* used to access the large object plays a big role in this case. Different algorithms and guarantees are possible when accessing objects locally versus gaining a small amount of information about a global property. Discovering algorithms that are extremely query-efficient may motivate new hardware or sensors that effectively implement certain oracles.

The uncertainty and size of data, along with the diversity of computing paradigms, leads to an exciting time for algorithm design and analysis. Besides time and space, other complexity measures, such as communication or queries, become relevant. Noise in the dataset means that it might suffice to *approximate* the true solution, gaining speed by reducing accuracy. Assumptions on the data can inspire simpler methods, but the analysis and modeling must reasonably represent real datasets. Depending on the application and environment, the best trade-offs of these various measures may lead to significantly better algorithms.

In the rest of this chapter, we review the research contributions presented in this thesis. We first describe a new clustering algorithm for DNA data storage, where the objective is to find all close pairs of strings. Our improvements include several methods for reducing the number of comparisons. The next result concerns similarity joins in a distributed model, and we describe new algorithms and lower bounds for the communication among a large number of processors. After this we describe theoretical results on the maximal number of close pairs in a set of binary vectors. This investigation into extremal properties is motivated by and informs our similarity join algorithms. Finally, we present results in an oracle model, giving new randomized algorithms for estimating the number of edges in a graph using only independent set queries.

1.1 Clustering in Edit Distance for DNA Data Storage

As digital data continues to be produced at an astronomical rate, researchers and engineers are compelled to look for new storage media. Biology offers natural inspiration, since information about life is stored in DNA. The description of DNA as a sequence of $\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$ characters[1] even feels digital, and the already small constituent nucleotides fold into a minuscule amount of space. This suggests that silicon-based devices might be very far from the physical limits of information density. After all, a cubic inch of DNA could feasibly store an exabyte of data, that is, a trillion gigabytes, at least 1000x denser than magnetic tape. Beyond density, DNA offers other favorable properties, such as efficient replication and immense longevity – data stored in DNA should last hundreds of years, while still allowing perfect retrieval.

The storage density and durability of DNA comes with many costs, both biochemical and computational. Reading and writing arbitrary data using synthetic DNA requires complex biochemical instruments and processes. Fortunately, these rapidly developing tools already suffice to build relatively large DNA-based storage systems [40, 68, 74, 123, 152].

Figure 1.1 depicts the pipeline for storing data in DNA. During the writing step, arbitrary

[1]The characters $\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$ stand for the four molecules adenine, cytosine, guanine, and thymine, which are bases contained in the nucleic acid polymer known as deoxyribonucleic acid (DNA).

Figure 1.1: A depiction of the DNA data storage pipeline.

digital files are compressed and encoded to the $\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}$ alphabet using a global error correcting code. In current systems, DNA strands contain hundreds of characters, and files must be split into smaller blocks (with addresses appended during the encoding process). During retrieval, the short strands are amplified and sequenced, resulting in an unordered digital file containing a small number of noisy copies of (almost) every original strands (despite best efforts, strands get dropped during the various stages of the pipeline). Overall, gigabytes of data lead to billions of noisy strings.

Although there are many interesting questions, we focus on a large-scale clustering problem – the most significant computational bottleneck in data retrieval from DNA. The goal is to group these noisy copies, based on the original but unseen strands. This will aid in the error-correcting process needed to recover the data. More precisely, after the clusters are found, a consensus-finding step will predict the most likely center for each cluster. This will produce candidates for the true strands. Finally, the global code corrects remaining errors, resulting in the original file.

In algorithmic terms, we tackle the problem of quickly clustering noisy strings. There are two main challenges. First, in DNA storage datasets, the clusters are small, containing about 5 to 15 elements. Therefore, the number of clusters k satisfies $k = \Omega(n)$ for n input strings. Off-the-shelf algorithms taking $\Theta(kn) = \Theta(n^2)$ time would be prohibitively slow for datasets with billions of strings. The second challenge is the errors in the strings. These include insertions, deletions, and substitutions of individual characters (in other words, the strings differ in edit distance). Also, the noise rate of real datasets is usually quite high, and edits occur in 2-3% of the positions of each string. Moreover, these numbers are for high-fidelity

biochemical instrumentation, and the error rate is higher, up to 12%, for more affordable processes. Existing clustering algorithms for edit distance scale quadratically with the amount of data, both theoretically and empirically, in this high-noise, many-cluster regime.

Our Contributions

We present a new algorithm for finding groups of similar strings in edit distance. Our approximation algorithm efficiently computes a highly-accurate clustering, runs in near-linear time, and scales well with the number of processors. We exhibit a 1000x speed-up over the state-of-the-art DNA clustering method, while simultaneously achieving higher accuracy.

We also propose a generative model for the underlying clusters of strings in DNA storage datasets. With respect to this model, we analyze strings and clusters in high-dimensional edit distance space, and we provide theoretical guarantees for our algorithm.

At a high-level, our algorithm is an agglomerative method. It initializes each input string as a separate cluster and iteratively merges similar clusters. A simple solution would be to find the closest pair of clusters and merge those, but this would require quadratic time. To circumvent this obstacle, our algorithm uses a new hashing scheme and binary embedding for edit distance that both take into account the structure of datasets arising from DNA storage systems. Our clustering algorithm and analysis have been presented in the following paper.

> Cyrus Rashtchian, Konstantin Makarychev, Miklos Racz, Siena Dumas Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. *Clustering Billions of Reads for DNA Data Storage*. In Proc. of the Advances in Neural Information Processing Systems (NIPS), pages 3362-3373, 2017. [133]

Our result enables the recovery of terabytes of data stored in DNA. Indeed, it is currently in use as part of our system, described by Organick *et. al.*, that has stored the largest quantity of digital data in DNA (400+ megabytes, to date) [123]. In Chapter 5, we provide more algorithm details, and in Chapter 7, we identify theoretical open questions.

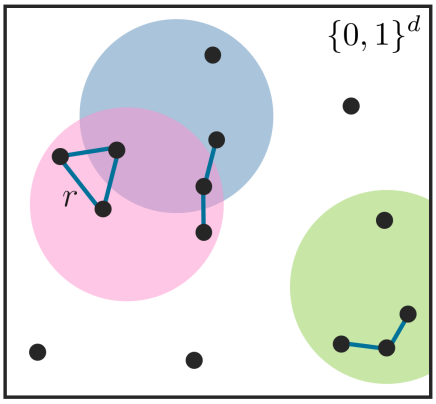


Figure 1.2: Simultaneous protocol for a distributed similarity join.

1.2 Communication-Efficient Similarity Join

Similarity joins have many applications, including recommendations, data deduplication, and multimedia search. As mentioned before, the simplest algorithm compares all pairs of input points and finds the pairs with distance below a given threshold. This requires $\Theta(n^2)$ comparisons. We ask when it is possible to do better, especially for large datasets.

Distributed Model

We prove upper and lower bounds for a distributed model with p processors. The n input points start initially partitioned, with each processor having n/p locally. Then, the algorithm executes in rounds, alternating between local computation and global communication. During a communication round, we allow replication of the data, so that some points go to multiple processors, and the data is no longer equally partitioned.

The number of communication rounds is often a dominant factor in overall runtime. Therefore, we aim to deeply understand the complexity of this problem for *one communication round*, where the processors must output all (or most) close pairs after a single shuffle. We assume that the local computation time increases as a function of the maximum number of points received by any processor, and we focus on minimizing the latter quantity.

Figure 1.2 portrays an example protocol, where the colored circles correspond to the points *received* by individual processors. Close pairs must have both endpoints inside some circle, so that this processor can output this pair. Points contained in multiple circles are replicated and sent to multiple processors. The challenge in designing an efficient distributed algorithm comes from outputting every close *pair* while balancing the workload among the processors and completing the overall similarity join quickly.

Our Contributions

We provide a new distributed algorithm and prove new communication lower bounds for similarity joins in a distributed, shared-nothing environment. Our algorithm employs a randomized load-balancing strategy tailored to the underlying metric space. For Hamming distance[2] on $\{0,1\}^d$ with threshold $r > 1$, we improve upon the best previous work due to Afrati, Das Sarma, Rajaraman, Rule, Salihoglu, and Ullman [5].

The main insight is to use edge-isoperimetric sets to design communication protocols. An *edge-isoperimetric set* in a graph is a set of vertices of a given size that contains the largest number of edges. For Hamming distance similarity graphs, an edge-isoperimetric set contains the maximal number of pairs of vectors with distance at most r. Previous algorithms use other methods [4, 5, 84, 126, 136, 149], and we are the first to use edge-isoperimetric sets for this problem. Though the optimal edge-isoperimetric sets for Hamming distance $r > 1$ are not known, in our improved algorithm we use Hamming balls, since we can show that they are nearly-optimal. We also prove a general lower bound that applies much more broadly than this edge-isoperimetric construction, extending to randomized, approximately-correct algorithms. These results appear in the following paper.

> Paul Beame and Cyrus Rashtchian. *Massively-Parallel Similarity Join, Edge-Isoperimetry, and Distance Correlations on the Hypercube.* In Proc. 28th ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 289-306, 2017. [32]

[2]The Hamming distance between two binary vectors equals the number of differing bits.

This work is the topic of Chapter 3. To get a sense of our results, consider having a dataset $S \subseteq \{0,1\}^d$ with $|S| = n$, and using p processors. The best algorithms, including ours, lead to roughly $d^{O(r)} \cdot n/p$ vectors being sent to each processor. For our new upper bounds, we improve the constant in the exponent by a multiplicative factor for large datasets. For our lower bounds, we show that $d^{\Omega(r)} \cdot n/p$ vectors must go to some processor, even if the algorithm is only approximately correct.

This research led to many connections to areas of extremal graph theory and combinatorics. See Section 1.3 for our new edge-isoperimetric results. For our lower bound, we proved new bounds on certain distance correlations in the hypercube (see Section 1.4).

1.3 Edge-Isoperimetry

Isoperimetric questions ask to maximize density while respecting a size budget. This area goes back to the classical result that balls in $\mathbb{R}^d$ maximize volume for a given surface area. While there are many discrete analogues, we will maximize the edge-density of induced subgraphs in a global graph. This makes sense for *symmetric* graphs, which have the property that for any pair of edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ in G, there is an automorphism f such that $f(u_1) = u_2$ and $f(v_1) = v_2$. Many graphs are symmetric, such as similarity graphs for Hamming distance. An *edge-isoperimetric inequality* for a symmetric graph is an upper bound on the maximum number of edges with both endpoints contained in a subset of vertices of a given size.

Figure 1.3 shows the graph connecting pairs at distance $r = 1$ in $d = 4$ dimensions. In this case, it is easy to see that among sets of size 2, 4, or 8, an optimal set can be achieved by fixed 3, 2, or 1 coordinates, respectively. The red points in this picture demonstrate a similarity join for distance one, for the set $\{0100, 1000, 1100, 1001\}$, which induces three edges.

We study edge-isoperimetric inequalities for distance r similarity graphs under Hamming distance, for $1 \leqslant r \leqslant d$. In other words, for a budget m, we will bound the number of pairs with distance at most r over all subgraphs induced by sets $\mathcal{A} \subseteq \{0,1\}^d$ with $|\mathcal{A}| = m$. The distance $r = 1$ case is well-understood: Any subset $\mathcal{A} \subseteq \{0,1\}^d$ induces a subgraph with at most $(1/2)|\mathcal{A}| \log_2 |\mathcal{A}|$ edges, proved by Harper [85], Bernstein [34], Hart [88], and

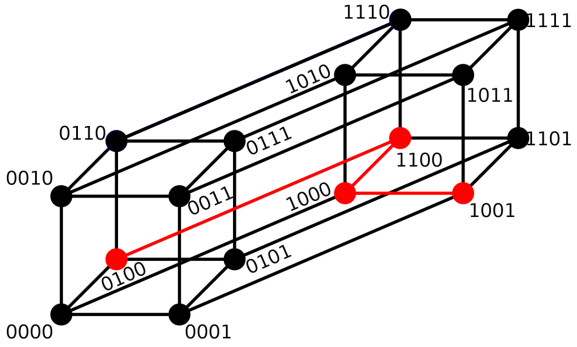


Figure 1.3: Distance one similarity graph for $\{0,1\}^4$ and an example similarity join.

Lindsey [112]. This is tight when $\mathcal{A}$ is a *subcube*, that is, when $\mathcal{A}$ is defined by fixing some number of coordinates and letting the others vary in $\{0,1\}$.

The exact result remains unknown for distances $r > 1$ [8, 86, 95]. Surprisingly, for distance $r \geqslant 2$, subcubes are no longer optimal and other sets dominate. Consider a sphere $\mathcal{S}_k$ of radius k around 0^d, defined as all vectors with precisely k ones. This set $\mathcal{S}_k$ has size $\binom{d}{k}$ and has $\frac{k(d-k)}{2} \cdot |\mathcal{S}_k|$ pairs at distance exactly two. In contrast, a subcube $\mathcal{C}$ of a similar size has only $O((k \log d)^2 \cdot |\mathcal{C}|)$ such pairs, which is many fewer when $k \ll d/\log^2 d$.

Our Contributions

We prove new upper bounds for $r \geqslant 2$, resulting in approximate edge-isoperimetric inequalities for Hamming distance similarity graphs. We also show that there is a dichotomy depending on whether r is even or odd. In both cases, our new results are tight up to a constant depending only on r. In particular, we show that Hamming balls are nearly-optimal for even distances, and certain hybrids between balls and subcubes are nearly-optimal for odd distances. The details can be found in Chapter 4. As one consequence, we verify that our distributed algorithms for Hamming distance similarity joins are indeed almost the best possible for a large class of one-round protocols.

1.4 Distance Correlations

Extremal graph theory studies the interplay between global graph properties (*e.g.,* edge density) and local structures (*e.g.,* copies of certain subgraphs). In Section 2.5, we describe a beautiful conjecture of Erdős-Simonovitz and Sidorenko on the density of homomorphisms of a (fixed) graph H in another graph G, where we minimize over all graphs G with a certain edge density [67, 138]. For technical reasons, it is necessary to assume that H and G are bipartite. Let G be a bipartite graph with n vertices on each side and $\delta \cdot n^2$ edges, and let H be any bipartite graph with m edges. The conjecture is that G must contain at least δ^m homomorphic copies of H. This is tight for G being a random graph with edge probability δ. The conjecture is known to be true in many cases, such as when H is a tree.

Inspired by this line of work, we ask a related question for Hamming distance similarity graphs. When can we guarantee a *large* number of pairs within a certain distance in a subgraph? In some ways, this is the converse of an edge-isoperimetric inequality. Notice that a pair at distance r is connected by a "path" of length r, which is comprised of distance one edges. Therefore, analogous to the Erdős-Simonovitz-Sidorenko conjecture, we could hope for a number of paths of length r that grows in terms of the number of distance one pairs. More generally, we could lower bound distance R pairs in terms of distance r pairs, for any $R > r$. In Chapter 3, we demonstrate that such a question is at the core of proving lower bounds for distributed similarity joins for all distance scales $r \geqslant 1$.

Our Contributions

We prove the analogous result for Hamming distance similarity graphs, and achieve nearly-tight bounds (up to constants depending on the distances). More generally than distance r versus distance one, we show that for any set of boolean vectors, if there is a sufficiently high density of pairs at some Hamming distance, then there is also a high density of pairs at all larger distances (where the growth depends on the distance ratio). This property can be viewed as an Erdős-Simonovitz-Sidorenko result for shortest paths in powers of the hypercube.

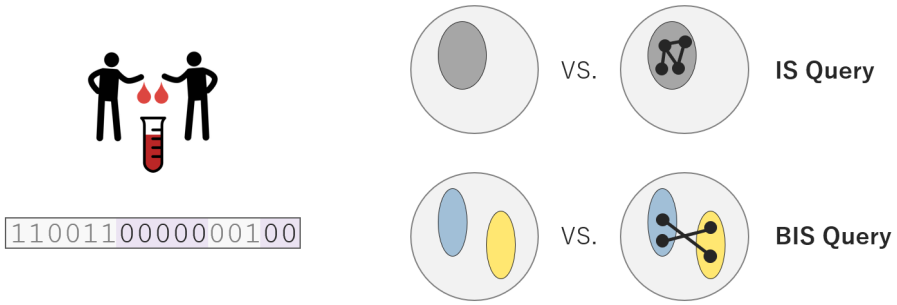


Figure 1.4: Group testing, independent set (IS), bipartite independent set (BIS) oracles.

1.5 Edge Estimation with Independent Set Oracles

Besides distributed algorithms, another way to handle large datasets is to design approximate methods that only access the data a small number of times. Sometimes this is the sole option: for massive graphs or populations, it may be infeasible to exactly compute statistics. In other cases, certain data accesses might be very informative but especially expensive or slow. This motivates oracle models, where we aim to estimate some parameter while bounding the number of queries to an oracle that tells us something about the dataset.

Consider testing a population of people to estimate the number of infected individuals. By pooling together multiple samples of blood, we can detect in one shot whether everyone in a group is healthy or there is at least one infected person. Since blood tests can be expensive, such *group tests* have been successfully used in practice, including testing soldiers for Syphilis in WWII [50, 60, 143]. In computer science terms, the group testing problem estimates the number of ones in long bit-vector using queries that answer whether a subset of coordinates contains all zeros or any ones. The left half of Figure 1.4 depicts group testing.

Independent Set Queries

Continuing our quest to understand pairwise problems, we adapt the group testing framework to the graph setting. Graph parameter estimation has a rich history, with results on

approximately counting edges, triangles, and more complicated structures [63, 70, 75, 77].

Previous work on graph parameter estimation has relied only on queries such as learning the degree of a vertex or whether an edge exists in the graph. However, such queries cannot achieve sub-polynomial query costs on families of graphs identified by Feige [70] and Goldreich and Ron [75], since these queries can only obtain *local* information about the graph.

We study the problem of graph parameter estimation using independent set queries (which answer whether a subset of vertices contains an edge or not), and bipartite independent set queries (which answer whether there exists any edge between two subsets of vertices). The right half of Figure 1.4 depicts these queries. Both types of independent set queries naturally generalize an edge existence query, and their non-locality opens the door for sub-polynomial query algorithms for various graph parameter estimation tasks.

Our Contributions

We present two new algorithms to estimate the number of edges in an n-vertex graph using $n^{2/3} \cdot \log^{O(1)} n$ independent set queries or $\log^{O(1)} n$ bipartite independent set queries, respectively. Our bipartite independent set query result is the first to achieve sub-polynomial query complexity (using a natural graph oracle) for certain hard instances. These results appear in the following publication.

> Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. *Edge Estimation with Independent Set Oracles.* In Proc. 9th Innov. in Theor. Comp. Sci. (ITCS), pages 1-21, 2018. [30]

Our randomized, adaptive algorithms combine several techniques that may have other parameter estimation applications, such as edge sparsification, coarse estimation, and importance sampling. Along the way, we also provide surprisingly efficient ways to *exactly* recover an unknown graph by using a number of queries that scales roughly with the number of edges (up to a logarithmic factor). Finally, we demonstrate a striking dichotomy, showing that bipartite independent set queries seem much more powerful than independent set queries.

1.6 Outline of Thesis

The rest of this thesis is organized as follows.

- **Chapter 2** provides background material on metric geometry, similarity search, clustering, and related graph-theoretic topics. The purpose is to set common notation and explain well-known concepts. Readers familiar with these topics may skip this chapter.

- **Chapter 3** presents our work on distributed similarity joins, including new communication upper and lower bounds for Hamming distance. It also contains the results on distance correlations in the hypercube. This joint work with Paul Beame has been published at SODA 2017 [32].

- **Chapter 4** contains new approximate edge-isoperimetric inequalities for Hamming distance. This joint work with David Ellis arose as part of the work in Chapter 3.

- **Chapter 5** describes the research on large-scale, distributed clustering in edit distance, with applications to DNA data storage. This joint work with Konstantin Makarychev, Miklós Z. Rácz, Siena Dumas Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss has been published at NIPS 2017 [133].

- **Chapter 6** provides algorithms for estimating the number of edges using independent set oracles. This joint work with Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, and Makrand Sinha has been published at ITCS 2018 [30].

- **Chapter 7** concludes and contains open directions for future research. The goal is to outline a larger area, into which this work fits, as well as describe a handful of concrete open theoretical questions.

Chapter 2

BACKGROUND ON DISTRIBUTED SIMILARITY SEARCH

As mentioned in the introduction, we assume that (i) complicated objects, such as images, products, or genomic sequences, have been represented by vectors, and (ii) similar vectors correspond to similar objects. This chapter explores the mathematical tools for designing and analyzing algorithms that find similar pairs or groups of vectors. The goal will be to understand methods that cater to the geometry of distances between vectors. We emphasize tools for a distributed setting, where multiple machines process the dataset. This is in contrast to the rich body of work concerning geometric data structures that reside in main memory (see, for example, the excellent book by Har-Peled [83]).

The rest of this chapter is organized as follows. We begin with metric space preliminaries. Then, we define the tasks, such as nearest-neighbor search, similarity joins, and clustering. We mention connections to extremal graph theory. We conclude with concentration inequalities.

2.1 Gentle Overview of Metric Spaces

We first define a metric space then provide several examples. In what follows, e is the base of the natural logarithm, ln, and all logarithms, log, have base two unless otherwise specified.

Definition 1. A *metric space* is a pair $(\mathcal{X}, d_{\mathcal{X}})$, where $\mathcal{X}$ is a set and $d_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ is a nonnegative *distance function* on pairs in $\mathcal{X}$. The function $d_{\mathcal{X}}$ satisfies three properties:

(i) $d_{\mathcal{X}}(x, y) = d_{\mathcal{X}}(y, x)$,

(ii) $d_{\mathcal{X}}(x, y) = 0$ if and only if $x = y$, and

(iii) $d_{\mathcal{X}}(x, y) \leqslant d_{\mathcal{X}}(x, z) + d_{\mathcal{X}}(z, y)$ for all $x, y, z \in \mathcal{X}$.

The third property (known as the *triangle inequality*) differentiates metric spaces from arbitrary notions of distance, and it implies much useful structure.

Examples of Common Metric Spaces

- Hamming distance: $([k]^d, d_{\mathrm{H}})$, where k is a positive integer, $[k]$ denotes the set $\{1, 2, \ldots, k\}$, and $d_{\mathrm{H}}(x, y)$ counts the coordinates that differ in x and y. Usually we will be interested in $k = 2$ and denote the set of vectors by $\{0, 1\}^d$.

- Euclidean distance: $(\mathbb{R}^d, d_{\ell_2})$, where $d_{\ell_2}(x, y) = \|x - y\|_2 \coloneqq \left(\sum_{i=1}^{d} (x_i - y_i)^2\right)^{1/2}$.

- Cosine distance: $(\mathcal{S}^{d-1}, d_{\ell_2})$ is the important special case of Euclidean distance that restricts to unit vectors on the d-dimensional sphere $\mathcal{S}^{d-1} = \{x \in \mathbb{R}^d : \sum_{i=1}^{d} x_i^2 = 1\}$.

- Edit distance: $(\Sigma^d, d_{\mathrm{E}})$, where Σ is a finite alphabet (e.g., $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$) and $d_{\mathrm{E}}(x, y)$ is the minimum number of insertions, deletions, or substitutions of individual characters needed to transform x to y. Edit distance satisfies the triangle inequality because $d_{\mathrm{E}}(x, z)$ edits can transform x to z and then $d_{\mathrm{E}}(z, y)$ edits can go from z to y.[1]

- Set distance: $(\mathcal{P}(\Sigma), d_J)$, where $\mathcal{P}(\Sigma)$ denotes the power set (all subsets) of a finite set Σ, and $d_J(A, B) = 1 - |A \cap B|/|A \cup B|$ is the *Jaccard distance* between sets A and B.

- Shortest path distance: (V, d_G), where G is an undirected, connected graph with vertices V, and $d_G(x, y)$ is the length of the shortest path between x and y in G.

Vectors may be very similar in one metric but very different in another. For example, the binary strings $0101 \cdots 0101$ and $1010 \cdots 1010$ in $\{0, 1\}^d$ have Hamming distance d but edit distance two. On the other hand, some distances may be expressed as others. For binary vectors, Euclidean distance is the square root of Hamming distance, which itself is the shortest path metric on the standard hypercube, that is, the graph with vertices $\{0, 1\}^d$ and edges connecting vectors with Hamming distance one. In Chapter 5, we will transform strings to sets and then to binary vectors to approximate edit distance with Hamming distance.

[1] Other edits (e.g., transpositions or block deletions and replications) might be relevant for certain applications, but we do not consider these variations in this thesis. Edit distance is also sometimes known as *Levenshtein distance*

Besides the geometry, the time to evaluate a distance function has implications for theory and practice. Hamming and Euclidean distances need $O(d)$ operations for d-dimensional vectors. Edit distance takes $O(d^2)$ operations via dynamic programming. Masek and Paterson provide a better algorithm, with $O(d^2/\log^2 d)$ time [116]. Backurs and Indyk show that, assuming a certain complexity-theoretic conjecture, no algorithm can compute the edit distance in time $O(d^{2-\varepsilon})$ for a positive constant ε [24]. We will use a variation by Ukkonen that determines whether $d_{\mathrm{E}}(x, y) \leqslant r$ with only $O(rd)$ operations [145]. Another algorithm by Landau, Myers, and Schmidt, determines this in $O(d + r^2)$ time [105]. However, the $O(rd)$ algorithm is more efficient in practice. Set similarity between A and B takes $O(\max\{|A|, |B|\})$ time, via hashing. The evaluation time of the shortest path distance varies depending on the graph, but we do not use this metric or go into detail here.

For completeness we mention norms on $\mathbb{R}^d$, although they play a minor role in this thesis. For a vector $x \in \mathbb{R}^d$, the ℓ_p norm of x is defined for $p \in [1, \infty)$ as

$$\|x\|_p \coloneqq \left(\sum_{i=1}^{d} |x_i|^p \right)^{1/p}.$$

Although one can extend the definition above to $0 < p < 1$, these functionals do not in general define norms, since they do not always satisfy the triangle inequality. On the other hand, for $p = 0$ and $p = \infty$, there are suitable generalizations. By convention $\|x\|_0$ is defined as the sparsity of x, that is, the number of nonzero elements, and $\|x\|_\infty$ is defined as the maximum magnitude of a coordinate; that is,

$$\|x\|_\infty \coloneqq \max_{i \in [d]} |x_i|.$$

In other words, we have analogous spaces ℓ_0 and ℓ_∞. These norms induce a metric d_{ℓ_p}through

$$d_{\ell_p}(x, y) \coloneqq \|x - y\|_p.$$

Notice that d_{ℓ_1} reduces to Hamming distance on binary vectors, and d_{ℓ_2} is Euclidean distance.

We move on to describe algorithmic tasks for datasets from a metric space.

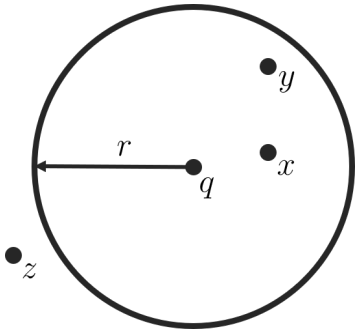


Figure 2.1: Query q has nearest neighbor x and r-near-neighbors x and y. The distance r similarity join of these four points is $\{\{q, x\}, \{x, y\}, \{y, q\}\}$.

2.2 Flavors of Similarity Search

Computational problems fall into two broad categories. In the first, the goal is to support similarity-based queries after preprocessing a set of input points. As a use case, we might want to find a small set of advertisements, restaurants, or job listings that are relevant for a given user. The second category concerns *all-pairs* problems, where the goal is to quickly find all close pairs in a dataset. Clustering is an all-pairs task, where we want to partition a dataset into smaller groups with mutually similar points. This thesis mostly focuses on all-pairs problems, but techniques and results will overlap for both categories.

We define near-neighbor search and similarity joins, then discuss clustering in Section 2.3.

Definition 1 (Near(est) Neighbor Search)**.** Fix a metric space $(\mathcal{X}, d_{\mathcal{X}})$, a threshold r, and a dataset $S \subseteq \mathcal{X}$. After preprocessing S, the *r-near-neighbor* problem for a query $q \in \mathcal{X}$ is to output all $x \in S$ with $d_{\mathcal{X}}(q, x) \leqslant r$. The *nearest-neighbor* problem returns $\mathsf{argmin}_{x \in S}\ d_{\mathcal{X}}(q, x)$. For a parameter $k \geqslant 1$, the *k-nearest-neighbor* problem for a query $q \in \mathcal{X}$ returns a set $T \subseteq S$ with $|T| = k$ such that $d_{\mathcal{X}}(q, x) \leqslant d_{\mathcal{X}}(q, y)$ for any $x \in T$ and $y \in S \setminus T$.

Definition 2 (Similarity Join)**.** The *similarity join* at distance r of a set S in a metric space $(\mathcal{X}, d_{\mathcal{X}})$ is the set of pairs in S with distance at most r, that is,

$$\{\{x, y\} \subseteq S \mid d_{\mathcal{X}}(x, y) \leqslant r, x \neq y\}.$$

Figure 2.1 depicts nearest neighbor, r-near-neighbor, and similarity join for a small example. Approximate versions of these problems allow pairs at distance cr for $c \geqslant 1$. We mention these in Section 2.4. Interestingly, even if we are interested in exact solutions, many techniques for approximate similarity search will be relevant.

The simplest similarity join algorithm compares all pairs $x, y \in S$ and records every pair $\{x, y\}$ with $d_{\mathcal{X}}(x, y) \leqslant r$, where r is the distance threshold. For datasets with $|S| = n$ points, this requires $\binom{n}{2} = \Theta(n^2)$ comparisons.

The following algorithm computes a similarity join by finding the r-near-neighbors of all points in S. First, preprocess S to support lookups "is $y \in S$?". This takes $O(n \log n)$ time using a binary search tree, and queries take $O(\log n)$ time. Then, for every $x \in S$, perform a lookup for all possible points y with distance at most r from x. This yields the similarity join, and the time scales with the the number of possible close points to each $x \in S$.

For Hamming distance, there are at most $\binom{d}{r} + \binom{d}{r-1} + \cdots + \binom{d}{1}$ vectors $y \in \{0,1\}^d$ within distance r of a fixed $x \in \{0,1\}^d$. Notice that this quantity is $O(d^r)$. Therefore, $O(d^r \cdot n)$ lookups suffice to compute a similarity join. When $d^r \ll n$, this may be a significant improvement over the $O(n^2)$ comparison algorithm. A natural question is whether this dependence on d and r is necessary: Do algorithms need time $d^{\Omega(r)} \cdot n$? In Chapter 3, we consider this question in a distributed setting. We show that there is a better algorithm, with communication complexity scaling with $(d/\log n)^{\lceil r/2 \rceil} \cdot n$. We also prove the lower bound that $(d/\log n)^{\Omega(r)} \cdot n$ is the necessary complexity for a large range of d, r, n. Proving similar lower bounds for other computational models remains an enticing research direction.

Similarity Graph

A graph formulation of similarity can unifying. Any graph can be viewed as connecting similar pairs by an edge. More implicitly, we can define a similarity graph in a metric space.

Definition 2. For a metric space $(\mathcal{X}, d_{\mathcal{X}})$ and a threshold $r > 0$, the *distance r similarity graph* is the graph on vertices $\mathcal{X}$ that connects all distinct pairs $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \leqslant r$.

The standard hypercube on $\{0,1\}^d$ is a similarity graph, with 2^d vertices and $d2^{d-1}$ edges, connecting pairs with Hamming distance one. In general, the distance r Hamming distance similarity graph has vertices with degree $\binom{d}{r} + \binom{d}{r-1} + \cdots + \binom{d}{1}$.

Similarity graphs allow us express geometric concepts in terms of graph-theoretic ones. For a fixed distance threshold, the near-neighbors of a vector are exactly its graph-neighborhood, and computing a similarity join is reporting the set of edges induced by the input set.[2] This viewpoint will be an integral part of our similarity join algorithms in Chapter 3.

2.3 Clustering

Clustering means dividing data into groups based on similarity. Elements within the same group should be more similar to each other than to any element of another group. A partition of the dataset into non-empty subsets will serve as our definition of a clustering. Similarity joins can be thought of as a form of clustering. The similarity join corresponds to the subgraph induced by S in the distance r similarity graph. The connected components of this subgraph form a partition of S. Of course, this clustering is most interesting if the data is separated, and the subgraph induced by S is disconnected.

To analyze a clustering algorithm, we need to measure the quality of a clustering. We mention a few examples and refer the reader to other references for more [72, 90, 99, 107]. In Chapter 5, we define a new clustering objective for DNA data storage applications.

If we are clustering vertices of a graph, then we could maximize the number of edges within each cluster and minimize the number of edges between clusters (*e.g,* correlation clustering [28] or spectral clustering [96]). If the dataset is sampled from an underlying distribution, then the goal could be to find a clustering that helps us estimate certain parameters (*e.g.,* determining means and variances of a mixture of Gaussians [37]).

Another example objective is the k-center problem, which is well-defined for any metric space $(\mathcal{X}, d_{\mathcal{X}})$. Given a dataset $S \subseteq \mathcal{X}$, the goal is to quickly find a set of centers $T \subseteq \mathcal{X}$

[2]For a set S of vertices in a graph, the *subgraph induced by* S contains all edges with both endpoints in S.

with the constraint $|T| = k$. The objective is that $\max_{a \in \mathcal{S}} d_{\mathcal{X}}(a, T)$ is minimized over all such sets T. Here we extend the definition of a distance function to sets by $d_{\mathcal{X}}(a, T) = \min_{b \in T} d_{\mathcal{X}}(a, b)$. Gonzalez showed that this problem is NP-hard in general, and he exhibited a simple approximation algorithm [78]. His algorithm greedily finds points in S that are farthest from previously chosen centers, and it runs in time $O(k|S|)$. When k is relatively small compared to $|S| = n$, this works quite well, taking time $\Theta(kn)$. However, when $k = \Omega(n)$, then this algorithm takes quadratic time, $\Theta(n^2)$.

We observe that in Gonzalez's algorithm, clusters can be formed by associating each point in S to its closest center in T. Moreover, this can be viewed as a *top-down* approach, since we are broadly grouping related points based on common criteria. In contrast, clusters could also be formed in a *bottom-up* fashion, where nearby points are iteratively grouped together until clusters emerge. Clustering algorithms are often broadly organized into two categories based on whether they operate in a top-down or bottom-up manner, known as *divisive* versus *agglomerative* methods in the clustering literature [72, 90, 99].

Divisive vs. Agglomerative

Divisive methods seek to optimize a global objective (this includes the above k-center problem, along with the classical k-means problem [113, 139]). Agglomerative algorithms use local rules to iteratively merge similar elements into larger clusters. This dichotomy serves primarily as a qualitative guide, and in many cases algorithms tend to be a hybrid. In our clustering algorithm for DNA data storage (described in Chapter 5), we use an agglomerative method, since it scales well both with a large number of processors and clusters.

As one example of an agglomerative method, we mention *linkage-based* algorithms. Here, the clustering is initialized with every point being its own separate cluster. Then, in each iteration, the two closest clusters are merged. Variations exist depending on the measure of cluster closeness. Some popular examples include single-linkage (distance between the two closest points in two clusters) and average-linkage (mean distance between points in two clusters). In practice, these criteria may lead to very different clusterings.

An analogous dichotomy of divisive versus agglomerative also shows in classical algorithms for finding a Minimum Spanning Tree (MST). For a connected, weighted graph G with n vertices, an *MST* is a connected tree on n vertices that minimizes the sum of its edge weights among all such trees. Kruskal's algorithm greedily builds an MST by iteratively adding an edge of minimum weight, while avoiding cycles. On the other hand, the "Reverse-Delete" algorithm iteratively removes an edge with maximal weight, without disconnecting the graph. Interestingly, both algorithms result in exactly the same MST.

Clustering Real Data: Speed, Filtering, and Outliers

For large datasets, clustering algorithms must be very efficient. An integral component of the running time comes from evaluating a distance function on pairs of points. To optimize an algorithm, we should reduce both the number of pairwise comparisons and the time it takes to perform a comparison. To this end, many *filtering* methods have been proposed, such as Canopy Clustering by McCallum, Nigam, and Ungar [118] and Clustering Using Representatives (CURE), by Guha, Rastogi, and Shim [81]. Another way to filter is to apply a hash function to the data and only compare points in the same hash bucket (we discuss such hashing in Section 2.4). In our clustering algorithm for DNA data storage, we combine many of these innovations (see Chapter 5).

Filtering methods can be very effective when the dataset is comprised of underlying clusters that are separated in the metric space. In this case, we say that the dataset is *clusterable*, and formal definitions of clusterability use a definition of *separation* [2, 27]. The simplest assumption may be that points in different clusters are strictly farther apart than same-cluster points. This can be in terms of a single scale (e.g., distance at most r vs. distance at least $2r$) or a more global criteria (e.g., misclassifying any point leads to a huge increase in the objective value) [23, 26, 114].

In real datasets, assumptions of separability or clusterability may be violated. This could be due to imprecise measurements or to noise in the data collection process. In these cases, it becomes important to understand how the output of a clustering algorithm changes if

adversarial points, or *outliers*, are added to the dataset [2, 27]. From a theoretical point of view, the question is whether algorithms still have good guarantees in the presence of outliers. Fortunately, in many cases, the non-outlier analysis will still hold even in the presence of outliers [2, 27]. For example, there exist algorithms that provide a good approximation for the optimal k-center clustering even with many outliers [115].

2.4 Locality Sensitive Hashing (LSH)

An influential line of work studies *approximate similarity search*, which both relaxes the distance threshold r and forgives a small percentage of false negatives. In this area, a main tool is Locality Sensitive Hashing (LSH), pioneered by Indyk and Motwani [84] and Kushilevitz, Ostrovsky, and Rabani [104]. Informally, an LSH family is a distribution over hash functions such that near points map to the same bucket with a higher probability than far points do. For Hamming space, an example LSH family chooses k random coordinates in $\{1, 2, \ldots, d\}$ and maps a vector to its k-bit projection on these coordinates.

In any similarity join or clustering algorithm, LSH provides a way to reduce the running time through filtering. A modified algorithm would first build several hash tables based on independent hash functions from an LSH family. Then, the algorithm only compares points that have ever been hashed to the same bucket. The accuracy of this method can be improved by using more hash tables. In many cases, hashing the dataset some number of times is much faster than performing a large number of comparisons [84, 142, 154]. We use a modified form of LSH in our clustering algorithm for DNA data storage (see Chapter 5).

A central motivation for LSH is the Approximate Near Neighbor (ANN) problem.

Definition 3. Let $S \subseteq \mathcal{X}$ be a dataset in a metric space $(\mathcal{X}, d_{\mathcal{X}})$. The (c, r)-*ANN problem* for $c > 1$ and $r \in \mathbb{R}_{\geqslant 0}$ is to efficiently pre-process S to quickly answer the following query. Given $q \in \mathcal{X}$, either return $x \in S$ with $d_{\mathcal{X}}(q, x) \leqslant cr$, or report that no such point exists.

The algorithm knows the approximation c and the threshold r ahead of time. Therefore, we focus on data structures parameterized by c, r. A useful primitive is an LSH family.

Definition 4. Let $r_1, r_2 \in \mathbb{R}_{\geqslant 0}$ with $r_1 > r_2$ and $p_1, p_2 \in (0, 1]$ with $p_1 > p_2$ be parameters. A hash family $\mathcal{H}$ of functions $h : \mathcal{X} \to \mathcal{X}$ in a metric space $(\mathcal{X}, d_{\mathcal{X}})$ is (r_1, r_2, p_1, p_2)-*sensitive* if the following two conditions are satisfied:

1. $\mathbf{Pr}[h(x) = h(y)] \geqslant p_1$ for all $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \leqslant r_1$, and
2. $\mathbf{Pr}[h(x) = h(y)] \leqslant p_2$ for all $x, y \in \mathcal{X}$ with $d_{\mathcal{X}}(x, y) \geqslant r_2$,

where the probability is over sampling a uniformly random $h \in \mathcal{H}$.

To solve the (c, r)-ANN problem, it actually suffices to find (r, cr, p_1, p_2)-sensitive hash families [84, 104]. If the dataset has n points, then any efficiently-computable family $\mathcal{H}$ leads to a data structure for the (c, r)-ANN problem with space $n^{1+\rho+o(1)}$ and query time $n^{\rho+o(1)}$ where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$. For Hamming distance, (r, cr, p_1, p_2)-sensitive families are known achieving $\rho = 1/c$. For Euclidean distance, the value is $\rho = 1/c^2$. The time and space bounds come from building n^ρ independent hash tables, each having n points.[3] For details, we refer readers to the well-written references on LSH and applications [83, 84, 104, 134]. LSH has been used successfully in practice (see, for example, [17, 25, 42, 142, 147]). Pagh exhibits an LSH extension for Hamming distance without false negatives [126, 131].

In this thesis, LSH-inspired ideas will come up in two places. For Hamming distance similarity joins in Chapter 3, our algorithms will use an analogous idea of random hashing, and our analysis will also center around the probability of close pair collisions. For edit distance clustering in Chapter 5, a core part of our algorithm will be also be a hash function, but with weaker guarantees than LSH (since actual LSH for edit distance remains unsolved).

2.5 Random Walks and a Conjecture of Erdős-Simonovitz and Sidorenko

Lower bounds for approximate similarity search often appeal to results on random walks in similarity graphs. This is the case in lower bounds for LSH by Motwani, Naor, and Panigrahy [120], with improvements by O'Donnell, Wu, and Zhou [121], and in data structure

[3] The $o(1)$ factors correspond to the space overhead for the points (which is assumed to be $n^{o(1)}$ per point) and the time for computing a hash function (which is also assumed to be $n^{o(1)}$).

lower bounds for ANN by Panigrahy, Talwar, and Wieder [129, 130]. The intuition is that finding near-neighbors is difficult if there are many possible candidates. The number of points within distance r of a query q can be analytically understood by looking at a random walk starting at q and running for $O(cr)$ steps. This boils down to understanding the number of paths of distance r in a graph G. In Chapter 3, we will use similar ideas for proving communication lower bounds for distributed similarity joins under Hamming distance.

In this context, we mention an interesting line of research, concerning copies of a fixed graph H in an overall graph G. In the case when H is a path, this relates to random walks.

Sidorenko's Conjecture

Consider a simple, undirected graph G, and let $V(G)$ and $E(G)$ denote the sets of vertices and edges of G, respectively. For a graph H, we will be interested in the number of "copies" of H in G. Although it may seem peculiar at first, it will make sense to count *homomorphisms* from H to G, which we now define.

Definition 5. A *homomorphism* from H to G is a mapping $f : V(H) \to V(G)$ of the vertices of H to the vertices of G that preserves edges, that is, if uv is an edge in H, then $(f(u), f(v))$ is an edge in G. Let $h_H(G)$ denote the number of homomorphisms from H to G. An *automorphism* of G is a bijective homomorphism from G to itself.

For example, when H is bipartite, we can take any edge uv in G, and map one side of H to u and the other to v, and $h_H(G)$ is at least the number of edges in G.

For a fixed H and varying G, we will be interested in the count $h_H(G)$ of homomorphisms from H to G. To state the bounds precisely, let's fix some notation. Let G be a graph on n vertices, and let H have m edges. We will be interested in lower bounds on relative density $t_H(G)$ of copies of H in G, that is,

$$t_H(G) = \frac{h_H(G)}{n^m}.$$

The term n^m equals the number of choices for m (not necessarily distinct) vertices of G, and $h_H(G)$ is the number of these subsets of vertices that actually correspond to copies of H.

Taking an extremal lens to the quantity, it is natural to ask which graphs G minimize $t_H(G)$ for a fixed H. Parameterizing this question based on the edge-density of G, we can define

$$f(H, \delta) = \min_G t_H(G)$$

where the minimum is over G with edge density of δ. A bit of thinking leads to the conclusion that bipartiteness plays a big role. Indeed, if G is bipartite and H is not, then there will be no copies of H in G! A famous conjecture due independently to Sidorenko [138] and Erdós-Simonovits [67] is that for any bipartite H, the minimum should be achieved by a random δ-dense bipartite graph G. In particular, they conjectured the following, often known as Sidorenko's conjecture, since he formulated in terms of homomorphism densities.

Conjecture 1 ([67, 138])**.** *For any bipartite graph H with m edges and any graph G,*

$$t_H(G) \geqslant t_{K_2}(G)^m,$$

where K_2 is the complete graph on two vertices.

In other words, this conjectures that $f(H, \delta) \geqslant \delta^m$, since $t_{K_2}(G) = \delta$ when G is δ-dense. We sketch why this conjecture would be tight for a random δ-dense graph G, where every edge exists with probability δ. A subset of m distinct vertices in G will be a valid homomorphism of H with probability δ^m. Since we are interested in fixed m, with n growing, a random choice of m vertices in G will almost surely be distinct. Therefore, we expect roughly $\delta^m \cdot n^m$ homomorphisms of H in G, and $t_H(G) \geqslant \delta^m$.

Although the conjecture remains unsolved in full generality, it is known in many special cases, such as when H is any complete bipartite graph, any tree, or any even cycle [138]. The most general techniques have recently followed two, disjoint, lines of research: dependent random choice [55] and entropy-based methods [109]. These prove Conjecture 1 when H has at least one vertex connected to all vertices on the other side. Certain approximate versions are also known [55], as well as extensions to related analytic formulations and graph norms [56]. At the time of writing, the smallest H for which Conjecture 1 is unsolved is when H is the complete bipartite graph on 10 vertices, $K_{5,5}$, minus a ten-cycle.

In Chapter 3, we will prove a theorem related to Conjecture 1 for shortest paths in certain Hamming distance similarity graphs. There will be certain differences, in that we will minimize over vertex-induced subgraphs of G instead of taking an edge-centric approach in terms of H. On the other hand, our results will be natural extensions of the proofs of Conjecture 1 when H is a path [15, 98]. It remains an interesting open direction to understand when results analogous to Conjecture 1 hold for metric-induced graphs and subgraphs.

2.6 Concentration Bounds

For proofs of the following concentration bounds, see the book by Dubhashi and Panconesi [61].

Lemma 2.1 (Hoeffding's inequality)**.** *Let $X_1, \dots, X_r$ be r independent random variables, where $X_i \in [a_i, b_i]$, for $i \in [r]$. Then, for $X = X_1 + \cdots + X_r$ and any $s > 0$, we have $\mathbb{P}[|X - \mathbb{E}[X]| \geqslant s] \leqslant 2\exp\left(-2\,s^2/\sum_{i=1}^{r}(b_i - a_i)^2\right)$.*

Lemma 2.2 (Chernoff-Hoeffding inequality [61, Theorem 1.1])**.** *Let $X_1, \dots, X_r$ be r independent random variables with $0 \leqslant X_i \leqslant 1$, and let $X = \sum_{i=1}^{r} X_i$. For $\mu = \mathbb{E}[X]$, let ℓ and u be real numbers such that $\ell \leqslant \mu \leqslant u$. Then, we have that*

1. *For any $\Delta > 0$, we have $\mathbb{P}[X \leqslant \ell - \Delta] \leqslant \exp\left(-2\Delta^2/r\right)$ and $\mathbb{P}[X \geqslant u + \Delta] \leqslant \exp\left(-2\Delta^2/r\right)$.*
2. *For any $0 \leqslant \delta < 1$, we have $\mathbb{P}[X \leqslant (1-\delta)\mu] \leqslant \exp\left(-\mu\delta^2/2\right)$.*
3. *For any $0 \leqslant \delta \leqslant 1$, we have $\mathbb{P}[X \geqslant (1+\delta)\mu] \leqslant \exp\left(-\mu\delta^2/3\right)$.*

We also need a version of Azuma's inequality that takes into account a rare bad event.

Lemma 2.3 ([61, Theorem 7.1])**.** *Let f be a function of r independent random variables $X_1, \dots, X_r$, such that $0 \leqslant f(X_1, \dots, X_r) \leqslant u$. Let $\mathcal{B}$ be some "bad" event. For $i \in [r]$, assume that*

$$\left|\mathbb{E}\Big[f \;\Big|\; X_1, \dots, X_{i-1}, X_i = a_i, \overline{\mathcal{B}}\Big] - \mathbb{E}\Big[f \;\Big|\; X_1, \dots, X_{i-1}, X_i = a_i', \overline{\mathcal{B}}\Big]\right| \leqslant c_i,$$

for some c_i. Then for any $s > 0$, we have that

$$\mathbb{P}\big[|f - \mathbb{E}[f]| > s + u\,\mathbb{P}[\mathcal{B}]\big] \leqslant 2\exp\big(-2s^2/\sum_{i=1}^{r} c_i^2\big) + 2\mathbb{P}\big[\mathcal{B}\big]\,.$$

Lemma 2.4 ([62]). *Let f be a function of m random variables $Z_1, \ldots, Z_m$ with $f(Z_1, \ldots, Z_m) \leqslant b$ for some $b \in \mathbb{R}_{\geqslant 0}$. For $i \in [m]$, let ξ_i satisfy*

$$|\mathbb{E}[f \mid Z_1, \ldots, Z_{i-1}, Z_i = a_i] - \mathbb{E}[f \mid Z_1, \ldots, Z_{i-1}, Z_i = a_i']| \leqslant \xi_i.$$

Then for any $\eta > 0$, we have that

$$\mathbf{P}\left[|f - \mathbb{E}[f]| > \eta\right] \leqslant \exp\left(-\frac{2\eta^2}{\sum_{i=1}^{m} \xi_i^2}\right)$$

Chapter 3

HAMMING SIMILARITY JOIN

The task of reporting all pairs of similar vectors arises as a core computation in many applications. We study this problem for a distributed model, motivated in part by the prevalence of MapReduce-style algorithms. For all-pairs problems, efficient solutions require some replication of the data among processors, and the communication over the network is a dominant factor in the overall runtime. We focus on one-round algorithms, which must output the correct answer after a single round of communication then local computation. This allows us to prove very strong lower bounds on the amount of data received by some processor in a shared-nothing environment. For example, this shows a lower bound on the necessary local memory for the processors. From a theoretical point of view, our results have novel connections to both similarity search algorithms and to extremal graph theory.

3.1 Introduction

Recall that the similarity join problem for a metric space $(\mathcal{X}, d_{\mathcal{X}})$ is the following: On an input set $S \subseteq \mathcal{X}$ and a distance threshold r, output all distinct x, y in S satisfying $d_{\mathcal{X}}(x, y) \leqslant r$.

Our algorithms and lower bounds are for a *simultaneous* model, where the processors have one round to communicate input points before locally computing and outputting their portion of the set of similar pairs. This model is consistent with previous work [11, 31, 136, 155], and with the fact that each additional round of communication greatly increases the runtime in practice. We assume that the local computation time increases with the maximum number of received vertices, and we focus on minimizing the latter quantity.

If there are p processors and n input points, an ideally balanced load with no replication of data would send precisely n/p points to each processor. However, since every similar *pair*

must end up at some processor, some replication is required, and it will not be possible to achieve this ideal.

We measure the *overhead* of an algorithm as the ratio of the maximum number of points any processor receives to the n/p ideal[1]. To avoid large overhead in the worst case, any adequate algorithm must employ a randomized load balancing strategy.

We exhibit new algorithms that improve on the overhead of the best previous similarity join algorithms from [4, 5, 136] for Hamming distance on $\{0,1\}^d$ with distance thresholds $r > 1$. Our improvements are both quantitative in terms of the overhead bound and more general in terms of the range of set sizes n for which they work. More fundamentally, we show that every randomized similarity join algorithm for Hamming distance r on $\{0,1\}^d$, even one that only weakly approximates the similarity join, requires overhead $d^{\Omega(r)}$. This qualitatively matches the overhead of the best algorithms. Moreover, ours is the first lower bound for similarity join that applies for any $r \neq 1$, for any set size that is $o(2^d)$, or for any algorithm that is only approximately correct.

We develop our improved similarity join algorithm as well as our lower bound by analyzing certain graph-theoretic quantities of a similarity graph. For distance threshold r on $\{0,1\}^d$, the Hamming similarity graph has edges between all pairs of points with Hamming distance at most r. To approximate the similarity join, both endpoints of most edges of this graph must arrive together at some processor. This criterion has a natural interpretation in graph terms. Let $A_1, \ldots, A_p$ be the sets of points that are sent to each of the p processors. Then, the collection of p subgraphs induced by these sets must contain many of the edges in the input set. Since the input consists of an arbitrary set of points, it follows that randomized algorithms correspond to distributions $\mathcal{A}$ on approximate coverings $(A_1, \ldots, A_p)$ of the edges of the similarity graph. We refer to such a p-tuple as an *edge-covering*.

The design of edge-coverings connects the similarity join problem with the edge-isoperimetric question in graph theory. First, our overhead measure is minimized by reducing the number

[1]Overhead resembles the replication rate used in analyzing MapReduce algorithms, but it does not depend on the notion of reducer size [136], and thus it applies more generally than to just MapReduce algorithms.

of input points contained by each set A_i. Second, an edge-covering is improved if each set A_i contains a large number of edges relative to its size. Therefore, we observe that it is good to build each set A_i as a union of randomly chosen edge-isoperimetric sets of suitable size in the similarity graph. In particular, we choose that size to be roughly n/p, the ideal number of input elements sent to a processor. This ensures bounded overhead and avoids imbalanced correlations of the input set with the edge-isoperimetric sets. In fact, this class of algorithms is a generic one that is suitable for use with any edge-transitive similarity graph, such as the graphs defined by the Euclidean or Manhattan distance.

Our general lower bound applies much more broadly than this edge-isoperimetric set construction. It holds for any randomized way of choosing sets $(A_1, \ldots, A_p)$. We show that edge-covering and similarity join are hard even if the input set of points S is itself a random, sub-sampled Hamming ball.

The general intuition for our lower bound proof is that either the total number of elements among the A_i is large, in which case large overhead follows immediately, or the sets A_i have a large density of pairs of points in $\{0,1\}^d$ of distance $\leqslant r$. For this second case we derive our lower bound on the overhead by proving a new connection between the density of different Hamming distances of subsets on $\{0,1\}^d$. We show that for $A \subseteq \{0,1\}^d$, if there is a sufficiently high density of pairs at Hamming distance r in A, then there is also a high density of pairs at much larger Hamming distances. This property is likely to be of independent interest since it can be viewed as analogous to a conjecture of Erdős-Simonovitz and Sidorenko, except for shortest paths in the r-th power of the hypercube (see Section 2.5 for more explanation of the Erdős-Simonovitz-Sidorenko conjecture [67, 138]).

3.2 Related Work

We review two baseline algorithms for the similarity join problem under any metric. The first is a randomized join algorithm and has overhead $O(\sqrt{p})$. The second baseline (known as the "Ball-hashing-2" algorithm in [4]) works by randomly partitioning the points z of the metric space among the processors and sending each input point x to the processors associated with

all z at distance at most $\lceil r/2 \rceil$ from x. In other words, processors are responsible for unions of balls of radius $\lceil r/2 \rceil$. The expected overhead of this algorithm is the size of balls of radius $\lceil r/2 \rceil$ in the metric space, which in the case of the Hamming distance on $\{0,1\}^d$ we denote by $B(d, \lceil r/2 \rceil)$ and is $O(d^{\lceil r/2 \rceil})$.

Recent work [4, 5, 136] demonstrates improvements over the baseline algorithms for $\{0,1\}^d$ with Hamming distance. For distance $r = 1$, Afrati *et al.* [4] present an algorithm (called "Splitting") based on a fixed grid of subcubes. They analyze how subcube dimensionality affects the overall communication. We observe that choosing the dimension to be $\lfloor \log(n/p) \rfloor$ leads to an edge-covering that achieves overhead $O(d/\log(n/p))$. For $r > 1$, in [4, 136] they also provide a variety of related algorithms. They analyze subcubes for larger distances, but this is not as effective as the Ball-hashing-2 algorithm in terms of overall communication. Their Ball-hashing-1 algorithm sends input points to processors based on balls of radius r centered at input points. However, the Ball-hashing-1 communication scales with $B(d, r)$, worse than the Ball-hashing-2 algorithm.

The best previous algorithm for $r > 1$ is the AnchorPoints from [5], which improves upon [4]. The basic building block of their algorithm is a *covering code* $\mathbf{C}$ of radius r, defined as a set of points $\mathbf{C}$ in $\{0,1\}^d$ with the property that every $x \in \{0,1\}^d$ is within distance r of some element of $\mathbf{C}$. After randomly partitioning the elements z of the code $\mathbf{C}$ among the processors, the algorithm sends each input point x to every processor associated with an element of the covering code that is at distance at most $\lceil 3r/2 \rceil$ from x. In other words, processors are responsible for unions of balls of radius $\lceil 3r/2 \rceil$. By using covering codes of density at most $O(1/B(d, r))$, the work in [5] improves on the overhead of the Ball-hashing-2 algorithm by roughly a $2^{\lceil r/2 \rceil}$ factor. By basing our construction on nearly edge-isoperimetric sets and larger Hamming balls, our algorithm improves the previous bound by factor of roughly $(\frac{1}{2}\log_d(n/p))^{\lceil r/2 \rceil}$.

In Section 2.4, we have discussed Locality Sensitive Hashing (LSH) and its role in similarity search algorithms. In a MapReduce setting, LSH buckets correspond to groups of points sent together. Hashing a vector in $\{0,1\}^d$ by choosing k coordinates corresponds to partitioning

$\{0,1\}^d$ into subcubes of dimension k and determining the subcube to which the vector belongs. Achieving a balanced load requires that each hash bucket contain a bounded number of vectors. In contrast, all-pairs similarity search algorithms based on LSH ignore the sizes of each bucket. Instead they focus on the total work done by the algorithm, and, over many rounds, maintain a counter for each vector and simply abort after comparing it to too many far vectors [84, 127]. In fact, as we show, the buckets maintained by LSH algorithms for distances $r > 1$ do not lead to the best load balance.

The approximate similarity join problem we study requires a sharp distance threshold – only producing pairs with distance at most r; false positives can be easily filtered before being output. Therefore, our notion of approximation only refers to the fraction of false negatives. This differs from problems such as Approximate Near Neighbor (see Section 2.4).

There are well-known lower bounds for LSH, such as [21, 120, 121], but these only apply for $r = \Omega(d)$. Indeed, they employ analytic techniques, such as the Bonami-Beckner inequality, that do not capture the small distances that are the focus of our work.

Panigrahy, Talwar, and Wieder [129, 130] prove cell-probe lower bounds for randomized algorithms solving the approximate near neighbor search problem (recent improvements appear in [20]). Their approach resembles ours in that they consider (a generalization of) the edge expansion of the similarity graph for a metric space (i.e., the graph with edges connecting every pair of points with distance at most r), though the specific application and techniques are different.

Recently, there has been work on proving strong conditional lower bounds (assuming the strong exponential time hypothesis [91]) on the time complexity of computing all-pairs near-neighbors [6, 13, 14]. However, these techniques currently only work when $r = \Omega(d)$, and they do not imply anything for similarity joins or near-neighbor search with $r = o(d)$. Handling this regime is an interesting direction for future work.

There are also other kinds of algorithms for similarity search. In particular, there is a substantial body of work on the design and analysis of data structures that first preprocess the data and then answer nearest-neighbor and range queries. Most of the analysis considers

time complexity and space usage for serial algorithms. Therefore, the majority of these results are orthogonal to the study of distributed similarity joins and edge coverings (cf. [83]).

For metrics other than Hamming distance, objects resembling edge-coverings underlie state-of-the-art algorithms for similarity joins on datasets of real vectors: Working in Euclidean space, Aiger, Kaplan, and Sharir [11] demonstrate an efficient edge-covering using randomly shifted and rotated grids. In the ℓ_∞ metric, Lenhof and Smid [106] provide serial and shared-memory algorithms using a data-dependent partition of space into ℓ_∞ hypercubes. For angular distance, LSH schemes use random half-spaces or spherical caps as randomized edge-coverings [17, 48, 84, 142, 147].

3.3 Similarity Graphs, Edge-Coverings, and Overhead

For a distance threshold $r \in [d]$, let $\Gamma_{\leqslant r}$ denote the graph with vertices $\{0,1\}^d$ and edges connecting u, v whenever $d_H(u,v) \leqslant r$, with self-loop edges for $u = v$. Notice that the self-loops differentiate this from Definition 2 of a similarity graph. However, allowing self-loops will be convenient for later calculations and notation. This regular graph has degree $B(d,r) \coloneqq \binom{d}{r} + \binom{d}{r-1} + \ldots + d + 1$. For a subset A in $\{0,1\}^d$, let $\mathsf{E}_{\leqslant r}(A)$ denote the set of edges in $\Gamma_{\leqslant r}$ with both endpoints in A. Similarly, define Γ_r as the graph with edges connecting vertices with distance exactly r, and define $\mathsf{E}_r(A)$ as the set of edges in Γ_r with both endpoints in A. It will be convenient to define notation for the average density of edges in the induced subgraphs $\Gamma_{\leqslant r}[A]$ and $\Gamma_r[A]$, so we define $\mathsf{e}_{\leqslant r}(A) \coloneqq \frac{|\mathsf{E}_{\leqslant r}(A)|}{|A|}$ and $\mathsf{e}_r(A) \coloneqq \frac{|\mathsf{E}_r(A)|}{|A|}$.

Similarity Join Algorithms and Overhead

We study a broad, natural class of one-round, p-processor similarity join algorithms. Each point x in the input set S is sent to some non-empty subset of processors $P(x) \subseteq [p]$. The algorithm then uses local operations at each destination processor to report all similar pairs among the elements it receives from S. We allow the mapping P to depend on the (approximate) size n of the set S but not S itself. The goal is to minimize the maximum number of elements received by any processor, while ensuring that (almost) every similar

pair in S goes to at least one processor.

The ideal load balancing would be a load of n/p elements per processor, but we show that this is not always possible. Randomization is essential in defining P, since for any deterministic mapping P any set $S \subseteq P^{-1}(i)$ for any $i \in [p]$ will yield a load of $|S|$. It is convenient to use an equivalent and convenient perspective by considering the distribution $(P^{-1}(1), \ldots, P^{-1}(p))$ of tuples of sets produced by the algorithm. This emphasizes the combinatorial aspects of the problem, and is formalized in the following definition in terms of randomized edge-coverings.

Definition 6. A *randomized* (p, δ)*-edge covering* for the n-subsets of $\Gamma_{\leqslant r}$ is a distribution $\mathcal{A}$ on p-tuples $(A_1, \ldots, A_p)$ of subsets of $\{0,1\}^d$ such that for every subset $S \subseteq \{0,1\}^d$ of size n,

$$\mathop{\mathbb{E}}_{(A_1,\ldots,A_p)\sim\mathcal{A}} \left[|\bigcup_i \mathsf{E}_{\leqslant r}(A_i \cap S)| \right] \geqslant \delta \cdot |\mathsf{E}_{\leqslant r}(S)|.$$

An analogous definition of a (p, δ)-edge covering for n-subsets of Γ_r has E_r replacing $\mathsf{E}_{\leqslant r}$.

As a way to capture the maximum load of randomized algorithms, our primary complexity measure will be the expectation of the maximum ratio of $|A_i \cap S|$ to $|S|/p$.

Definition 7. Let $\mathcal{A}$ be a distribution on p-tuples $(A_1, \ldots, A_p)$ of subsets of $\{0,1\}^d$. Let $S \subseteq \{0,1\}^d$ with $p \leqslant |S|$. Define the *overhead* of $\mathcal{A}$ on set S,

$$\mathsf{overhead}(\mathcal{A}, S) = \mathop{\mathbb{E}}_{(A_1,\ldots,A_p)\sim\mathcal{A}} \left[\max_{i\in[p]} |A_i \cap S| \cdot \frac{p}{|S|} \right],$$

where the expectation is over the randomness of $\mathcal{A}$. The *n-overhead* of $\mathcal{A}$ is defined as

$$\mathsf{overhead}_n(\mathcal{A}) = \max_{S\subseteq\{0,1\}^d,\ |S|=n} \mathsf{overhead}(\mathcal{A}, S)$$

Note that defining the overhead in terms of the *maximum* value of $|A_i \cap S|$ is critical, since a trivial algorithm that sends S entirely to a random processor achieves average load exactly $|S|/p$.

Any randomized (p, δ)-edge covering $\mathcal{A}$ yields a natural one-round, p-processor, data-parallel algorithm for computing a δ-approximate similarity join. First, choose $(A_1, \ldots, A_p) \sim$

$\mathcal{A}$ using shared randomness. Then, send x to processor i whenever $x \in A_i$. In other words, on input S, processor i receives $A_i \cap S$ for $i \in [p]$. Over the random choices of the algorithm, this strategy outputs in expectation at least a δ fraction of all pairs in S with Hamming distance at most r.

These *local* algorithms capture a very broad class of natural algorithms for similarity joins including the ones discussed in the introduction. For example, in the ball-hashing-2 algorithm, the sets $A_1, \ldots, A_p$ are unions over a random partition of Hamming balls of radius $\lceil r/2 \rceil$.

A Universal All-pairs Algorithm.

As another example of such algorithms we present the other baseline mentioned in the introduction which works for any notion of similarity and yields overhead $O(\sqrt{p})$.

Proposition 3.1. *There is a randomized 1-round local algorithm for any similarity measure that has expected per-processor load $O(|S|/\sqrt{p})$ on all sets S with $p \leqslant |S|$.*

Proof. Assume without loss of generality that $p = \binom{q}{2}$ for integer q. Choose a random mapping $h : \{0,1\}^d \to [q]$. Now send $x \in \{0,1\}^d$ to processors indexed by the pair $\{h(x), i\}$ for every $i \in [q]$ not equal to $h(x)$. Every pair of inputs in S will be seen by some processor. The per-processor load is almost surely $O(|S|/q)$. □

This yields a randomized $(p, 1)$-edge-covering of $\{0,1\}^d$ with overhead $O(\sqrt{p})$ for all set sizes. In Section 3.6.4 we note that for very small sets S, this bound essentially cannot be improved, even on $\Gamma_{\leqslant r}$. For a limited number of processors, this algorithm is still useful in practice, even for large datasets. However, many important cases involve sets S that are a much larger portion of $\{0,1\}^d$. In fact, some previous work has often focused on the case that the input set size is completely at the other extreme – a constant fraction of all possible vectors. We consider a wide range of input sizes and not just these extremes.

Main Complexity Measure

We prove bounds on the following quantity based on overhead.

Definition 8 (Main Complexity Measure). Let $f_{\leqslant r}(m, n, p, \delta)$ be the minimum of $\mathsf{overhead}_n(\mathcal{A})$ over all randomized (p, δ)-edge coverings for n-subsets of $\Gamma_{\leqslant r}$. Analogously define $f_r(n, d, p, \delta)$ for Γ_r.

Lower bounds on $f_{\leqslant r}(m, n, p, \delta)$ are lower bounds on the amount of overhead required for any randomized p-processor 1-round local algorithm that approximately solves similarity join on the binary cube with Hamming distance threshold r for subsets $S \subseteq \{0,1\}^d$ of size n. Upper bounds on $f_{\leqslant r}(m, n, p, \delta)$ and $f_r(n, d, p, \delta)$ will be particularly interesting when $\delta = 1$, which implies no error, or when δ is sufficiently close to 1, which implies the probability of missing any similar pair in S is extremely small.

Comparison to MapReduce Measures.

In MapReduce algorithms, there is a collection of individual tasks called *reducers* that are randomly assigned to one of the p processors. Previous similarity join algorithms (as well as our edge-isoperimetric algorithm) define reducers for each subset $R_1, \ldots, R_K$ of $\{0,1\}^d$ for some $K \gg p$. Then, during a shuffling phase, the sets $A_1, \ldots, A_p$ are formed randomly as unions of some of the $\{R_j\}$. The key complexity measure analyzed in the previous work [4, 5, 136] is the *replication rate*, defined as the average number of reducers to which each point is sent. We claim that overhead is a better measure.

The replication rate is a useful measure only when the reducers are small enough that they can be mapped randomly and uniformly to yield tasks that are nicely load-balanced across the processors. Our measure, overhead, captures a much wider class of algorithms than replication rate can. Moreover, in the regime where replication rate is a useful measure, overhead essentially equals replication rate. The main advantage of overhead concerns very large reducers. For example, if one reducer was responsible for essentially the whole input, the replication rate would be one, but this algorithm would behave badly in practice and has the worst possible overhead of p. Therefore, overhead both captures the lower bounds using replication rate and defines a measure of load balancing for large reducers.

3.4 Our Results

Theorem 3.2. *Let $0 < \delta \leqslant 1$ and r, d, n, p be such that $r \leqslant d$, $p \leqslant n \leqslant 2^d$, and $\gamma = \log_n p$.*

(a) For all integers k with $\lceil r/2 \rceil \leqslant k \leqslant (1-\gamma)\log_d n$ there is a MapReduce algorithm witnessing the bound

$$f_{\leqslant r}(m, n, p, \delta) \leqslant \max\left\{6 \cdot 2^{\lceil r/2 \rceil} \ln\left(\frac{1}{1-\delta}\right) \cdot \left(\frac{d}{k}\right)^{\lceil r/2 \rceil}, \quad 9 \log p\right\}$$

The algorithm has reducer size $B(d, k)$, and with probability $1 - 2^{-d}$, it maps each input to at most

$$7 \cdot \max\left\{d \ln 2, \quad 6 \cdot 2^{\lceil r/2 \rceil} \ln\left(\frac{1}{1-\delta}\right) \cdot \left(\frac{d}{k}\right)^{\lceil r/2 \rceil}\right\}$$

processors. Moreover, with $\delta = 1 - 1/n^3$, the edge-covering of the input is error-free with probability at least $1 - 1/n$.

(b) There is constant $c_\delta > 0$ such that for every r, n, d, p, δ with $r \leqslant \sqrt{d/2}$, $n \geqslant d^{r \log d}$, and $\delta \geqslant 4/\sqrt{d}$,

$$f_{\leqslant r}(m, n, p, \delta) \geqslant (c_\delta / r)^r \cdot (d / \log_d^2 n)^{\gamma r/2}.$$

Further, if $\gamma r \leqslant 1$ then

$$f_{\leqslant r}(m, n, p, \delta) \geqslant (c_\delta / r)^r \cdot (d / \log_d n)^{\gamma r}.$$

(c) The bounds given in (a) and (b) also hold with $f_{\leqslant r}(m, n, p, \delta)$ replaced by $f_r(n, d, p, \delta)$.

Observe that our lower bound in (b) (in its range of applicability) is qualitatively very similar to the upper bound in (a) with maximal choice of k, especially as the number of processors approaches the set size. The only previous lower bound on replication rate or overhead for similarity join [136] used the edge-isoperimetric bound for $r = 1$ to derive a lower bound of $(\log n)/\log q$ where q is an upper bound on the reducer size for input sets of size n that are dense subsets of a subcubes; however, for q as large as n/p, the regime in which our bounds apply, this only yields a nearly trivial lower bound of $1/(1-\gamma)$.

We outline the upper bound improvements compared to previous algorithms [4, 136, 5]. Our upper bound with maximal k comes from an algorithm with overhead $O(2d/\log_m(n/p))^{\lceil r/2 \rceil}$.

This improves on the Ball-hashing-2 algorithm [4], which has overhead $(2d/r)^{\lceil r/2 \rceil}$ using reducers of size $B(d, \lceil r/2 \rceil)$. Our algorithm also improves over the AnchorPoints algorithm in [5], which has overhead $O(B(d, \lceil 3r/2 \rceil)/B(d, r))$, approximately $(d/r)^{\lceil r/2 \rceil}$, using reducers of size $B(d, \lceil 3r/2 \rceil)$. We finally note that the subcube algorithm [4, 136] has good overhead when $r = 1$ and is not improved by our results, but it is not competitive for $r > 1$.

Our algorithm is best for large k with reducers of size $B(d, k) \approx n/p$. We improve the overhead by up to a $(\frac{1}{2}\log_d(n/p))^{\lceil r/2 \rceil}$ factor, taking advantage of larger reducers to obtain better bounds. For a comparison using a natural range of parameters, consider $d = (\log \sqrt{n})^2$ and $p = 2^{\sqrt{d}} = \sqrt{n}$. Our overhead for even r scales like $(d \log d)^{r/4}$, which improves over the best previous algorithm by a factor of roughly $(d/r^2 \log d)^{r/4}$.

Our algorithm is nearly-optimal for $n = 2^{\Theta(d)}$ and $p = n^\gamma$ for constant $\gamma < 1$. In this case, choosing $\delta = 1 - 1/2^{3d}$ we obtain an algorithm that is almost certainly correct and has overhead only $2^{O(r)} d \log^{\lceil r/2 \rceil} d$, since $\log p \leqslant d$. This is a significant improvement over the best previous algorithm [5] which has overhead and replication rate $(d/r)^{\lceil r/2 \rceil}$ in this regime.

We remark that it is possible to improve our upper bound by a factor of roughly $\log(n/p)$ when the distance threshold r is odd. This is due to the fact that Hamming balls actually have fewer edges, asymptotically, than another set. The better set of binary vectors is a hybrid between a subcube and a Hamming ball). In particular, it is the product of a subcube on a coordinates and a Hamming ball on the other $d - a$ coordinates of the appropriate radius (depending on a and the set size). For example, for a value $\xi \in [0, 1]$, and set size n/p, we could take a subcube on $a = \xi \log(n/p)$ coordinates, and a Hamming ball on the remaining coordinates of radius roughly $(1 - xi) \log_d(n/p)$. This hybrid set provides a smooth transition from distance one (where subcubes are optimal) to larger odd distances.

3.5 Randomized Edge-Coverings using Edge-Isoperimetric Sets

In this section, we describe our randomized edge-covering for $\Gamma_{\leqslant r}$ with overhead achieving the upper bound in Theorem 3.2. This randomized edge-covering defines a one-round, data-parallel protocol for reporting all pairs in $S \subseteq \{0,1\}^d$ with Hamming distance at most r. It thus also provides a MapReduce algorithm for Hamming similarity joins.

We begin with our general protocol for arbitrary edge-transitive similarity graphs, since it is easy to state and explains the intuition. We then specialize this general protocol to Hamming distance and provide the specific analysis in this case.

General Edge-Transitive Graph Covering

We build a randomized (p, δ)-edge-covering for G using unions of random translates of a fixed set $U^* \subseteq V$. Recall that an edge-isoperimetric set of size s is any set $U^*(s)$ of vertices in G that maximizes the number of edges among all induced subgraphs with s vertices. We will use translates of $U^* \coloneqq U^*(s)$ for $s \leqslant n/p$. Let π be a graph automorphism on G. Define $\pi(U^*)$ to be the set of vertices $\pi(x)$ for each $x \in U^*$. By edge-transitivity, $\pi(U^*)$ contains the same number of edges as U^* does. In other words, edge-transitivity enables the designation of a canonical maximal set as a function of s.

To construct the (p, δ)-edge covering, we choose a parameter L and choose Lp automorphisms $\pi_1, \ldots, \pi_{Lp}$ of G uniformly at random, and define

$$A_i = \bigcup_{j \,\equiv\, i \bmod p} \pi_j(U^*).$$

With Lp large enough compared to the ratio of the number of edges in G to that of U^*, the sets A_i cover a δ-fraction of edges in S with high probability. As we show, the fact that $|U^*| \leqslant n/p$ ensures a bounded overhead via Bernstein's concentration inequality.

When interpreted as a one-round MapReduce algorithm, the construction above admits good bounds on both the maximum reducer size and the maximum number of times a vertex is replicated by the mappers.

Lemma 3.3. *Let U^* be a subset of vertices in $G = (V, E)$. The MapReduce algorithm that chooses Lp uniformly random automorphisms $\pi_1, \ldots, \pi_{Lp}$ and maps $x \in V$ to the reducer assigned $\pi_j(U^*)$ whenever $x \in \pi_j(U^*)$ has reducer size $|U^*|$, and with probability $1 - |V|^{-1}$, it maps each vertex to at most $7 \cdot \max\{\ln |V|,\ Lp|U^*|/|V|\}$ reducers.*

Proof. Any $x \in V$ is contained in $\pi_j(U^*)$ with probability $|U^*|/|V|$. The expected number of indices j such that $x \in \pi_j(U^*)$ is $Lp|U^*|/|V|$. By independence of the automorphisms, a standard Chernoff bound implies that x is mapped to more than $7 \cdot \max\{\ln |V|,\ Lp|U^*|/|V|\}$ reducers with probability at most $|V|^{-2}$. Taking a union bound over V finishes the proof. □

We now analyze the above algorithm for $\Gamma_{\leqslant r}$.

Edge-Isoperimetry for Hamming Distance

Currently, the optimal edge-isoperimetric set U^* in $\Gamma_{\leqslant r}$ is unknown for $r > 1$. Classical results show that (unions of) subcubes are optimal for $r = 1$ [34, 85, 88, 112]. The main result of Chapter 4 shows that Hamming balls are nearly-optimal for even distances $r \geqslant 2$. For odd distances, Hamming balls contain a large number of edges, but another set (a product of a subcube and a ball) actually has more edges, by a multiplicative factor logarithmic in the set size. For simplicity, we will not make the even/odd distinction and build an edge-covering out of random translates of Hamming balls of radius k with $B(d, k) \approx n/p$.

We now prove a lower bound on the number of edges in a Hamming ball, for all $r \geqslant 1$, which suffices to prove the upper bound for our algorithm. More precisely, we will lower bound the average degree in $\Gamma_{\leqslant r}$ of a ball of radius k, that is, the value $\mathsf{e}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^d, k))$.

Proposition 3.4. *For $k \in [d]$ with $\lceil r/2 \rceil \leqslant k$, we have*

$$\mathsf{e}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^n, k)) \geqslant \frac{1}{2} \cdot \binom{k}{\lceil r/2 \rceil} \cdot B(d - k, \lfloor r/2 \rfloor)$$

Proof. Observe that $\mathsf{e}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^n, k)) \geqslant \frac{1}{2} \min_{x \in \mathsf{Ball}_{\mathrm{H}}(0^n, k)} |\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x, r)|$.

We first argue that the minimum occurs when $|x| = k$. Indeed, let $|x'| < k$ and $|x| = |x'| + 1$ such that $x \oplus x'$ contains a single non-zero coordinate j.

CLAIM: $|\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x', r)| \geqslant |\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x, r))|$.

First note that for any $z \in \{0,1\}^d$, $\mathsf{Ball}_{\mathrm{H}}(z, r) = z \oplus \mathsf{Ball}_{\mathrm{H}}(0^d, r)$. Let $y \in \mathsf{Ball}_{\mathrm{H}}(0^d, r)$ have $y_j = 0$ and let y' be the same as y except that bit $y_j = 1$. Now $x' \oplus y = x \oplus y'$ and $x \oplus y = x' \oplus y'$. Therefore, (i) the number of such y for which $x' \oplus y \in \mathsf{Ball}_{\mathrm{H}}(0^n, k)$ but $x \oplus y \notin \mathsf{Ball}_{\mathrm{H}}(0^n, k)$ is equal to (ii) the number of such y' for which $x \oplus y' \in \mathsf{Ball}_{\mathrm{H}}(0^n, k)$ but $x' \oplus y' \notin \mathsf{Ball}_{\mathrm{H}}(0^n, k)$. Observe that (i) counts $\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x', r) \setminus (\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x, r))$ and (ii) counts a (strict) superset of $\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x, r) \setminus (\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x', r))$ since for any $z' \in \mathsf{Ball}_{\mathrm{H}}(0^d, r)$ with $z'_j = 1$, the element z equal to z' except that $z_j = 0$ is also a member of $\mathsf{Ball}_{\mathrm{H}}(0^d, r)$. (The strictness follows because for those $y \in \mathsf{Ball}_{\mathrm{H}}(0^d, r)$ with $|y| = k$ have $y' \notin \mathsf{Ball}_{\mathrm{H}}(0^d, r)$.) This proves the claim which implies that the minimum occurs when $|x| = k$.

Now fix some $x \in \mathsf{Ball}_{\mathrm{H}}(0^n, k)$ with $|x| = k$. We count vectors z in $\mathsf{Ball}_{\mathrm{H}}(0^n, k) \cap \mathsf{Ball}_{\mathrm{H}}(x, r)$. The vector z may be obtained from x by flipping i bits from one to zero, and j bits from zero to one, as long as $i \leqslant k$, $i + j \leqslant r$, and $j \leqslant i$. Therefore, by summing over (i, j) that meet these conditions, the number of vectors z is at least $\sum_{i \leqslant \min\{r,k\}} \sum_{j \leqslant \min\{r-i,i\}} \binom{k}{i}\binom{d-k}{j}$. This is a lower bound on the minimum, and thus average, degree of $\mathsf{Ball}_{\mathrm{H}}(0^n, k)$ in $\Gamma_{\leqslant r}$. □

3.5.1 Proof of Theorem 3.2(a)

Now we prove the upper bound in our main theorem by exhibiting a randomized (p, δ)-edge-covering for sets of size n in $\Gamma_{\leqslant r}$ achieving the claimed overhead.

For $p \leqslant (d/\log_d n)^{r/2}$ we can achieve the bound simply using the universal algorithm. Our construction for $p > (d/\log_d n)^{r/2}$ consists of a partition of a random set of Hamming balls. We denote the radius of these balls by k, which will be carefully chosen to achieve efficient coverage and ensure proper load balancing. Intuitively, if k is too small, then each ball has too few edges and we will need too many balls to cover all the edges in $\Gamma_{\leqslant r}$. On the other hand, if k is too large, then we risk having a large intersection $|S \cap A_i|$ on input S. We will see that

the value of k that achieves the best overhead bounds is such that $B(d,k) \approx |S|/p = n/p$, but we consider all values of k for which the algorithm makes sense.

Randomized Edge-Covering Construction using Hamming Balls

Let n be the input set size and $k \geqslant \lceil r/2 \rceil$ to be any integer such that $B(d,k) \leqslant n/p$. Observe that by our assumption $p > (d/\log_d n)^{r/2}$, we have that $k \leqslant d/2 - \Omega(\sqrt{rd \log d})$.

The number of Hamming balls in the covering will be proportional to the ratio of the number of edges in $\Gamma_{\leqslant r}$ and in $\mathsf{Ball}_{\mathrm{H}}(0^d, k)$. Let this ratio ℓ_k be

$$\ell_k \coloneqq \frac{|\mathsf{E}_{\leqslant r}(\{0,1\}^d)|}{|\mathsf{E}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^d,k))|}.$$

Define a random distribution $\mathcal{A}_k$ of p-tuples of subsets of $\{0,1\}^d$ with $(A_1,\ldots,A_p) \sim \mathcal{A}_k$ chosen as follows: Let $x_1,\ldots,x_{Lp} \subseteq \{0,1\}^d$ be a uniformly random sequence of vectors where $L = \ln(1/(1-\delta)) \cdot \lceil \ell_k/p \rceil$. Define the random sets A_i as

$$A_i = \bigcup_{j \,\equiv\, i \bmod p} \mathsf{Ball}_{\mathrm{H}}(x_i, k).$$

We begin by bounding ℓ_k, the ratio of the number of edges in $\Gamma_{\leqslant r}$ to that in $\mathsf{Ball}_{\mathrm{H}}(0^d,k)$. This bound on ℓ_k (and thus Lp) combines with Lemma 3.3 to immediately provide the claimed upper bounds in Theorem 3.2 on the reducer size and the number of times that a vector is replicated.

Proposition 3.5. *Fix $k \in [d]$ with $\lceil r/2 \rceil \leqslant k \leqslant d/2 - \lceil r/2 \rceil$. Then,*

$$\frac{|\mathsf{E}_{\leqslant r}(\{0,1\}^d)|}{|\mathsf{E}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^d,k))|} < \left(\frac{2d}{k}\right)^{\lceil r/2 \rceil} \cdot \frac{2^{d+1}}{B(d,k)}.$$

Proof. The definition of $\mathsf{E}_{\leqslant r}(\{0,1\}^d)$ and the lower bound on $\mathsf{e}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^d,k))$ in Proposition 3.4 imply that

$$\ell_k = \frac{|\mathsf{E}_{\leqslant r}(\{0,1\}^d)|}{|\mathsf{E}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^d,k))|} \leqslant \frac{B(d,r)2^d}{\binom{k}{\lceil r/2 \rceil} \cdot B(d-k, \lfloor r/2 \rfloor) \cdot B(d,k)}.$$

We invoke Proposition 3.18 from the appendix and use the assumption $d/2 \geqslant 2\lceil r/2 \rceil$ to bound $B(d,r)/B(d,\lfloor r/2 \rfloor) \leqslant \frac{5}{4} d^{\lceil r/2 \rceil}/r^{\lfloor r/2 \rfloor}$. Then, combining this with the simple bound $\binom{k}{j} \geqslant (k/j)^j$ we obtain that the upper bound on ℓ_k is at most

$$\frac{5}{4} \cdot \frac{(\lceil r/2 \rceil)^{\lceil r/2 \rceil}}{r^{\lfloor r/2 \rfloor}} \cdot \frac{B(d,\lfloor r/2 \rfloor)}{B(d-k,\lfloor r/2 \rfloor)} \cdot \left(\frac{d}{k}\right)^{\lceil r/2 \rceil} \cdot \frac{2^d}{B(d,k)} \quad < \quad \frac{d^{\lfloor r/2 \rfloor}}{(d-k)^{\lfloor r/2 \rfloor}} \cdot \left(\frac{d}{k}\right)^{\lceil r/2 \rceil} \cdot \frac{2^{d+1}}{B(d,k)}.$$

We used that $(\lceil r/2 \rceil)^{\lceil r/2 \rceil}/r^{\lfloor r/2 \rfloor}$ is at most 1 with r even and achieves its maximum value of 1.35 for r odd at $r = 5$. Finally, since $d - k - \lfloor r/2 \rfloor \geqslant d/2$, this is at most $(2d/k)^{\lceil r/2 \rceil} \cdot 2^{d+1}/B(d,k)$. □

The upper bound on the overhead in Theorem 3.2 follows from the following two lemmas, and the fact that $B(d,k) \leqslant d^k$, which implies that $B(d,k)$ for $k \leqslant (1-\gamma)\log_d n = \log_d(n/p)$ is at most n/p.

Lemma 3.6. *For all $S \subseteq \{0,1\}^d$ with $|S| = n$, for $k \geqslant \lceil r/2 \rceil$ and $B(d,k) \leqslant n/p$, we have*

$$\mathbb{E}[\mathsf{overhead}(\mathcal{A}_k, S)] \leqslant \max\{6\ln(1/(1-\delta)) \cdot (2d/k)^{\lceil r/2 \rceil}, 9\log p\}.$$

Moreover, the bound on the overhead holds almost surely.

Proof. Let $S \subseteq \{0,1\}^d$ with $|S| = n$. We will upper bound $\mathbb{E}[\max_i |S \cap A_i|]$ by proving an upper bound on $\mathbb{E}[|S \cap A_i|]$ for each $i \in [p]$, where the latter expectation is over the distribution on A_i induced by $\mathcal{A}_k$. After doing so, we union bound over $[p]$ to guarantee the bound on the expected max with high probability.

Fix $i \in [p]$. The set A_i is the union of $\mathsf{Ball}_{\mathrm{H}}(x_j,k)$ for L vectors x_j chosen uniformly and independently from $\{0,1\}^d$. Define the random variable $Z_j = |S \cap \mathsf{Ball}_{\mathrm{H}}(x_j,k)|$ for $j \in [L]$. Then, defining $Z = \sum_j Z_j$, we have $|S \cap A_i| \leqslant \sum_j Z_j = Z$. We first upper bound $\mathbb{E}[Z] \geqslant \mathbb{E}[|S \cap A_i|]$. Observe that for a uniformly random x_j from $\{0,1\}^d$, every element of $\{0,1\}^d$ is equally likely to be in $\mathsf{Ball}_{\mathrm{H}}(x_j,k)$. Therefore every $j \in [L]$,

$$\mathbb{E}[Z_j] = \mathbb{E}[|S \cap \mathsf{Ball}_{\mathrm{H}}(x_j,k)|] = \sum_{y \in S} \mathbf{Pr}[y \in \mathsf{Ball}_{\mathrm{H}}(x_j,k)] = \sum_{y \in S} \frac{B(d,k)}{2^d} = \frac{n \cdot B(d,k)}{2^d}.$$

Thus, recalling $L = \ln(1/(1-\delta)) \cdot \lceil \ell_k/p \rceil$, and using the upper bound on ℓ_k (the ratio of the number of edges in $\mathsf{Ball}_{\mathrm{H}}(0^d, k)$ and in $\Gamma_{\leqslant r}$) from Proposition 3.5, we compute

$$\mathbb{E}[Z] \;=\; L \cdot \frac{n \cdot B(d,k)}{2^d} \;=\; \ln(1/(1-\delta)) \cdot \lceil \ell_k/p \rceil \cdot \frac{n \cdot B(d,k)}{2^d} \;<\; 3\ln(1/(1-\delta)) \cdot (2d/k)^{\lceil r/2 \rceil} \cdot \frac{n}{p}.$$

For concentration, we use Bernstein's inequality (e.g., Thm. 3.6 in [53]), which applies since $0 \leqslant Z_j \leqslant B(d,k)$ and $\mathbb{E}[Z_j^2] \leqslant B(d,k)\mathbb{E}[Z_j]$. Therefore, for any $\lambda \geqslant \mathbb{E}[Z]$,

$$\mathbf{Pr}\left[Z > 2\lambda\right] \leqslant e^{-\frac{1}{2}\lambda^2/(B(d,k)\mathbb{E}[Z] + \lambda B(d,k)/3)} \leqslant e^{-\frac{3}{8}\lambda/B(d,k)}.$$

For every $\lambda = \lambda(\theta) \geqslant \max\{\mathbb{E}[Z], 4B(d,k)(\log p + \theta)\}$ the probability that $|S \cap A_i| \geqslant 2\lambda$ is at most $e^{-\theta}/p^{3/2}$. By the union bound over $[p]$, the probability that $\max_i |S \cap A_i|$ exceeds 2λ is at most $e^{-\theta}/p^{1/2}$. In particular, we have an upper bound on $\mathsf{overhead}(\mathcal{A}_k, S)$ of 2λ. □

We conclude with the correctness argument.

Lemma 3.7. *For any n, $\mathcal{A}_k$ forms a (p,δ)-edge-covering of the n-subsets of $\{0,1\}^d$. Further, if δ is chosen to be $1 - 1/n^3$, then $(A_1, \ldots, A_p) \sim \mathcal{A}_k$ covers all edges of any fixed input set of size n with probability at least $1 - 1/n$.*

Proof. Fix $S \in \{0,1\}^d$ with $|S| = n$. Since $\Gamma_{\leqslant r}$ is edge-transitive, for $x_j \in \{0,1\}^d$ chosen uniformly at random, each pair $\{u,v\} \in S$ of distance at most r satisfies $\{u,v\} \subseteq \mathsf{Ball}_{\mathrm{H}}(x_j, k)$ with probability

$$1/\ell_k = |\mathsf{E}_{\leqslant r}(\mathsf{Ball}_{\mathrm{H}}(0^d, k))|/|\mathsf{E}_{\leqslant r}(\{0,1\}^d)|.$$

Therefore, the probability that a fixed $\{u,v\}$ is not covered by $(A_1, \ldots, A_p) \in \mathcal{A}_k$ is at most

$$(1 - 1/\ell_k)^{Lp} \leqslant (1 - 1/\ell_k)^{\ln(1/(1-\delta))\ell_k} \leqslant e^{-\ln(1/(1-\delta))} = 1 - \delta.$$

Each edge of $\mathsf{E}_{\leqslant r}(S)$ is covered with probability at least δ, and $(A_1, \ldots, A_p) \sim \mathcal{A}_k$ covers in expectation at least a δ fraction of $\mathsf{E}_{\leqslant r}(S)$.

There are at most n^2 pairs in S so with $\delta = 1 - 1/n^3$ (in which case $\mathcal{A}_k$ chooses $3p\lceil \ell_k/p \rceil \ln n$ uniformly random balls of radius k) a union bound implies that the probability that $(A_1, \ldots, A_p) \sim \mathcal{A}_k$ covers all pairs in S is at least $1 - 1/n$. □

3.6 The Lower Bound

We now prove the lower bound of Theorem 3.2(b). Let $\mathcal{A}$ be any randomized (p, δ)-edge cover of the n-subsets of $\{0,1\}^d$ for $\Gamma_{\leqslant r}$.

We will use a variant of Yao's minimax principle by exhibiting a "hard" distribution on subsets S of $\{0,1\}^d$ of size n. The usual form of the argument would then be to show that the distribution has the property that the expected overhead of any fixed $(A_1, \ldots, A_p)$ in the support of $\mathcal{A}$ is large and conclude that the expected overhead for $\mathcal{A}$ on some element of the support of the hard distribution is large.

However, this does not quite work. Since $\mathcal{A}$ produces a large fraction of the similar pairs of S *only in expectation*, its support may contain $(A_1, \ldots, A_p)$ that achieve low overhead but without producing many similar pairs. Instead, we show that (1) the expected overhead under the hard distribution must be large for any tuple $(A_1, \ldots, A_p)$ such that $|\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i)|$ is at least a $\delta/2$ fraction of $\mathsf{E}_{\leqslant r}(\Gamma_{\leqslant r})$ and that (2) any $(A_1, \ldots, A_p)$ that fails to have this property does badly at covering subsets S chosen according to this distribution and hence such tuples must only be a small fraction of the probability mass of $\mathcal{A}$.

The hard input distribution $\mathcal{D}_{n,d}$ over sets $S \subseteq \{0,1\}^d$ of size n will choose S to be a random sub-sampled Hamming ball of suitable size so that S has relatively high density within that ball.

Definition 9. Let $d \in \mathbb{Z}^+$ and $r \in [d]$ be positive integers. Given $n \in [2^{d-1}]$, let $R = R(n, d, r) \in \mathbb{Z}^+$ denote the unique positive integer such that r divides R and

$$B(d, R - r) < n \leqslant B(d, R).$$

Define distribution $\mathcal{D}_{n,d}$ on subsets S of $\{0,1\}^d$ of size n by first choosing $x \in \{0,1\}^d$ uniformly at random, then choosing a random subset of n elements of $\mathsf{Ball}_{\mathrm{H}}(x, R)$.

Using the fact that $\mathcal{D}_{n,d}$ samples edges uniformly from $\Gamma_{\leqslant r}$ we have the following simple property of potential covers.

Proposition 3.8. *If* $(A_1, \ldots, A_p)$ *satisfies* $|\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i)| \leqslant \alpha \cdot |\mathsf{E}_{\leqslant r}(\Gamma_{\leqslant r})|$ *for some* $\alpha \in [0,1]$, *then*

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}_{n,d}} \left|\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i \cap S)\right| \;\leqslant\; \alpha \cdot |\mathsf{E}_{\leqslant r}(S)|.$$

Proof. Observe that $\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i \cap S) = \left(\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i)\right) \cap \mathsf{E}_{\leqslant r}(S)$. For S chosen according to $\mathcal{D}_{n,d}$, the set $\mathsf{E}_{\leqslant r}(S)$ contains each edge of $\Gamma_{\leqslant r}$ with equal probability, so the claim follows by linearity of expectation. □

Observing that $f_{\leqslant r}(m,n,p,\delta)$ is precisely

$$f_{\leqslant r}(m,n,p,\delta) = \min_{\mathcal{A}} \max_{S\subseteq\{0,1\}^d:\ |S|=n} \mathsf{overhead}(\mathcal{A}, S),$$

Yao's minimax principle [151] applied to distribution $\mathcal{D}_{n,d}$ yields the following.

Lemma 3.9. *Let* $\delta \in [0,1]$ *and* A_δ^{good} *be the set of tuples* $(A_1, \ldots, A_p)$ *of subsets of* $\{0,1\}^d$ *such that* $|\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i)| \geqslant \frac{\delta}{2} \cdot |\mathsf{E}_{\leqslant r}(\Gamma_{\leqslant r})|$. *Then,*

$$f_{\leqslant r}(m,n,p,\delta) \geqslant \frac{\delta}{2} \cdot \max_{(A_1,\ldots,A_p)\in A_\delta^{good}} \mathop{\mathbb{E}}_{S\sim\mathcal{D}_{m,n}} \left[\max_i |A_i \cap S| \cdot \frac{p}{n}\right].$$

Proof. Let $\mathcal{A}$ be a randomized (p,δ)-edge cover for the n subsets of $\Gamma_{\leqslant r}$. Therefore the expectation over $(A_1, \ldots, A_p) \sim \mathcal{A}$ and $S \sim \mathcal{D}_{n,d}$ of $|\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i \cap S)|$ must be at least $\delta|\mathsf{E}_{\leqslant r}(S)|$, since the $\delta|\mathsf{E}_{\leqslant r}(S)|$ bound holds for every choice of S. By Proposition 3.8 for $(A_1, \ldots, A_p) \notin A_\delta^{good}$ we have $\mathop{\mathbb{E}}_{S\sim\mathcal{D}_{n,d}} |\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i \cap S)| \leqslant \delta|\mathsf{E}_{\leqslant r}(S)|/2$. Since the most this quantity can be is $|\mathsf{E}_{\leqslant r}(S)|$, by Markov's inequality, the probability that $\mathcal{A}$ produces $(A_1, \ldots, A_p)$ that is in A_δ^{good} is at least $\delta/2$ and since all quantities are non-negative, the lower bound follows. □

For the remainder of this section we fix $(A_1, \ldots, A_p)$ in A_δ^{good}, so that

$$|\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i)| \geqslant \frac{\delta}{2}|\mathsf{E}_{\leqslant r}(\Gamma_{\leqslant r})| = \delta B(d,r)2^{d-2}. \tag{3.1}$$

We prove that $\mathop{\mathbb{E}}_{S\sim\mathcal{D}_{m,n}} \max_i |A_i \cap S|$ must be large. Note that for sufficiently small r, and sufficiently large δ, equation (3.1) implies a lower bound on the total number of edges of Γ_r covered by the A_i.

Proposition 3.10. *Suppose that $r \leqslant \sqrt{d/2}$ and $\delta \geqslant 4/\sqrt{d}$. If $(A_1, \ldots, A_p) \in A_\delta^{good}$, then*

$$|\bigcup_{i\in[p]} \mathsf{E}_r(A_i)| \geqslant \frac{\delta}{4} \cdot \binom{d}{r} \cdot 2^{d-1}$$

Proof. There are $B(d, r-1)2^{d-1}$ edges of $\Gamma_{\leqslant r}$ that are not in Γ_r. For $r \leqslant \sqrt{d/2}$, we see that

$$\frac{\binom{d}{r-1}}{\binom{d}{r}} = \frac{r}{d-r+1} < \frac{1}{\sqrt{d}+1}$$

and hence $B(d, r-1) \leqslant B(d, r)/\sqrt{d} \leqslant \frac{\delta}{4} \cdot B(d, r)$. Combining with (3.1), we have that at least $\frac{\delta}{4}B(d, r)2^{d-1} \geqslant \frac{\delta}{4}\binom{d}{r}2^{d-1}$ edges of Γ_r are contained in $\bigcup_{i\in[p]} \mathsf{E}_r(A_i)$. □

Next, we reduce the problem to reasoning about covers that only include necessary elements. Intuitively, it suffices to cover each vertex with fewer sets than its degree.

Definition 10. A tuple $(A_1, \ldots, A_p)$ is *r-pruned* if for every $x \in \{0,1\}^d$,

$$|\{i : \ x \in A_i\}| \leqslant B(d, r) - 1.$$

Lemma 3.11. *For any $(A_1, \ldots, A_p)$ there exists a r-pruned $(A'_1, \ldots, A'_p)$ such that $A'_i \subseteq A_i$ for all $i \in [p]$ and $\bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A'_i) = \bigcup_{i\in[p]} \mathsf{E}_{\leqslant r}(A_i)$.*

Proof. Define $\mathsf{w}(x) = |\{i \in [p] \ : \ x \in A_i\}|$. For $i \in [p]$, say that a set A_i is *pivotal with respect to x* if there exists an edge $\{x, y\}$ in $\Gamma_{\leqslant r}$ with $x \neq y$ such that $\{x, y\} \subseteq A_i$ while $\{x, y\} \subsetneq A_j$ for all $j < i$. For each $x \in \{0,1\}^d$, remove x from all but the pivotal sets for x. Let $(A'_1, \ldots, A'_p)$ be the resulting tuple of sets. Clearly, $\bigcup_i \mathsf{E}_{\leqslant r}(A'_i) = \bigcup_i \mathsf{E}_{\leqslant r}(A_i)$ by construction. For each x, the number of pivotal sets for is at most the size of the open neighborhood of x in $\Gamma_{\leqslant r}$, which is $B(d, r) - 1$. Therefore, $w(x) \leqslant B(d, r) - 1$ for all x, and hence, $(A'_1, \ldots, A'_p)$ is r-pruned. □

With these basic preliminaries out of the way we describe the overall proof strategy.

Proof Strategy

Lemma 3.11 allows us to assume without loss of generality that $(A_1, \ldots, A_p)$ is r-pruned since pruning yields the same set of similar pairs covered and does not increase overhead.

Our argument capitalizes on the following tradeoff. We have two cases: For the first case, if $\sum_{i\in[p]} |A_i|$ is much larger than 2^d then the algorithm will replicate vertices many times, implying a large overhead. For the second case, if $\sum_{i\in[p]} |A_i|$ is closer to 2^d, then we will prove that the average edge density of induced edges of $\Gamma_{\leqslant r}$ in the sets A_i must be large. In other words, in the second case, the weighted average of $\mathsf{e}_{\leqslant r}(A_i)$ is large. By Proposition 3.10, this means that the weighted average of $\mathsf{e}_r(A_i)$ is also large.

Using a key combinatorial lemma which shows that sets A with large $\mathsf{e}_r(A)$ must also have large $\mathsf{e}_{\leqslant R}(A)$ where R is the distance defined by distribution $\mathcal{D}_{n,d}$ and hence, in the second case, the sum of the $|\mathsf{E}_{\leqslant R}(A_i)|$ is large.

Finally, it is not hard to see that $|\mathsf{E}_{\leqslant R}(A_i)|$ being large is equivalent to saying that the expected size of $|A_i \cap \mathsf{Ball}_{\mathrm{H}}(x, R)|$ is large for x a randomly chosen element of A_i and it is not hard to see that a similar property also applies to high density random subsets S of $\mathsf{Ball}_{\mathrm{H}}(x, R)$; like elements chosen according to $\mathcal{D}_{n,d}$.

This roughly suggests that a set S chosen from $\mathcal{D}_{n,d}$ will end up producing a large overhead at some set A_i that contains the center x of the ball $\mathsf{Ball}_{\mathrm{H}}(x, R)$ used to define S. However, as x varies, which of these sets A_i are possible also varies and the choice of x and the set index i are correlated so choosing just one A_i would not let the rough argument succeed. Instead, using the fact that, without loss of generality, no element of $\{0,1\}^d$ needs to appear in too many sets A_i, we show that we can eliminate the bias by lower bounding the maximum overhead by the average overhead over all sets A_i with $x \in A_i$.

We begin by proving the last part of this strategy; i.e., we first relate the expected maximum size of $A_i \cap S$ to the number of pairs with distance at most R in A_i.

Lemma 3.12. *Let $d \in \mathbb{Z}^+$, $r \in [d]$, $n \in [2^{d-1}]$ and $R = R(n, d, r) = kr \leqslant d/2$ for integer*

$k \geqslant 2$. *Then for any r-pruned $\{A_1, \ldots, A_p\}$ with each $A_i \subseteq \{0,1\}^d$,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}_{m,n}} \left[\max_{i \in [p]} |A_i \cap S| \right] \;>\; \frac{B(d, R-r)}{2^{d-1} B(d,R)(B(d,r)-1)} \sum_{i \in [p]} |\mathsf{E}_{\leqslant R}(A_i)| \geqslant \frac{R^{(r)} r!}{d^{2r}} \sum_{i \in [p]} |\mathsf{E}_{\leqslant R}(A_i)|/2^d.$$

Proof. The distribution $\mathcal{D}_{n,d}$ first chooses a center $x \in \{0,1\}^d$ uniformly at random and then chooses $S \subseteq \mathsf{Ball}_{\mathrm{H}}(x, R)$ uniformly at random from all subsets of size n. Let $\mathcal{D}^x_{n,d}$ denote the conditional distribution of S given a fixed choice of x. Then, since $\{A_1, \ldots, A_p\}$, for all $x \in \{0,1\}^d$, we have

$$\mathsf{w}(x) = |\{i \in [p] \;:\; x \in A_i\}| \leqslant B(d,r) - 1.$$

We can restrict our choice of maximum to sets containing the center x and lower bound the maximum over such sets by their average to obtain

$$\begin{aligned}
\mathop{\mathbb{E}}_{S \sim \mathcal{D}_{m,n}} \left[\max_{i \in [p]} |A_i \cap S| \right] &= \frac{1}{2^d} \sum_{x \in \{0,1\}^d} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\max_{i \in [p]} |A_i \cap S| \right] \\
&\geqslant \frac{1}{2^d} \sum_{x \in \bigcup_i A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\max_{i \in [p]} |A_i \cap S| \right] \\
&\geqslant \frac{1}{2^d} \sum_{x \in \bigcup_i A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\max_{i:\; x \in A_i} |A_i \cap S| \right] \\
&\geqslant \frac{1}{2^d} \sum_{x \in \bigcup_i A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\sum_{i:\; x \in A_i} \frac{|A_i \cap S|}{B(d,r)-1} \right] \qquad \text{since } 0 < w(x) \leqslant B(d,r)-1 \\
&= \frac{1}{2^d} \sum_{x \in \bigcup_i A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\sum_{i \in [p]} \mathbb{1}_{x \in A_i} \cdot \frac{|A_i \cap S|}{B(d,r)-1} \right] \\
&= \frac{1}{2^d} \sum_{x \in \bigcup_i A_i} \sum_{i \in [p]} \mathbb{1}_{x \in A_i} \cdot \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\frac{|A_i \cap S|}{B(d,r)-1} \right] \\
&= \frac{1}{2^d (B(d,r)-1)} \sum_{i \in [p]} \sum_{x \in A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[|A_i \cap S| \right]. \qquad (3.2)
\end{aligned}$$

Now for each $i \in [p]$ and $x \in A_i$, we have $S \subseteq \mathsf{Ball}_{\mathrm{H}}(x, R)$; therefore

$$\begin{aligned}\sum_{x \in A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} [|A_i \cap S|] &= \sum_{x \in A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} \left[\sum_{y \in \mathsf{Ball}_{\mathrm{H}}(x,R) \cap A_i} \mathbb{1}_{y \in S} \right] \\ &= \sum_{x \in A_i} \sum_{y \in \mathsf{Ball}_{\mathrm{H}}(x,R) \cap A_i} \mathop{\mathbb{E}}_{S \sim \mathcal{D}^x_{n,d}} [\mathbb{1}_{y \in S}] \\ &= \sum_{x \in A_i} \sum_{y \in \mathsf{Ball}_{\mathrm{H}}(x,R) \cap A_i} \frac{n}{B(d,R)} \\ &= \frac{n}{B(d,R)} \cdot 2|\mathsf{E}_{\leqslant R}(A_i)|\end{aligned}$$

since the double summation counts each pair $\{x, y\}$ in A_i of Hamming distance at most R exactly twice. Now, by construction of distribution $\mathcal{D}_{n,d}$, n is larger than $B(d, R-r)$, so inserting this bound in (3.2) yields that $\mathop{\mathbb{E}}_{S \sim \mathcal{D}_{m,n}} [\max_i |A_i \cap S|]$ is larger than

$$\frac{B(d, R-r)}{2^{d-1} B(d,R)(B(d,r)-1)} \sum_{i \in [p]} |\mathsf{E}_{\leqslant R}(A_i)|$$

as required. Now, $B(d,r) \leqslant d^r/r!$ and, since $R > r$, Proposition 3.18 proved in the appendix shows that $B(d,R)/B(d,R-r) \leqslant (d+1)d^{r-1}/R^{(r)}$ which is at most $2d^r/R^{(r)}$. Together these yield the final claimed bound. □

To apply Lemma 3.12, we need a lower bound on $|\mathsf{E}_{\leqslant R}(A_i)|$. We obtain such a lower bound by proving a purely graph-theoretic result. This result relates different distances in subsets of the hypercube. More precisely, we prove that subsets of $\{0,1\}^d$ with sufficiently large density of Hamming distance r pairs (i.e., sufficiently dense induced subgraphs of Γ_r) also have a relatively large density of pairs at Hamming distance at most R when R is a multiple of r. This means that we can reduce the problem of lower bounding $|\mathsf{E}_{\leqslant R}(A_i)|$ to that of lower bounding $\mathsf{e}_r(A_i)$. We give this lemma and its proof in the next subsection.

3.6.1 Relating distance r density and distance R density on the hypercube

Lemma 3.13. *Fix $b \in \{0, 1, \ldots, \lfloor r/2 \rfloor\}$, and assume that r divides R with $r < R$. Assume that $A \subseteq \{0,1\}^d$ satisfies $\mathsf{e}_r(A) \geqslant 4\binom{R-r}{b+1}\binom{d}{r-b-1}$. Then*

$$\mathsf{e}_{\leqslant R}(A) \geqslant \frac{\mathsf{e}_r(A)^{R/r}}{4^{R/r} \cdot R! \cdot d^{bR/r}}.$$

This lemma is nearly tight; examples include Hamming balls of varying radii and subcubes of varying dimensionality. Some lower bound on $\mathsf{e}_r(A)$ is necessary for this to hold since unions of well-separated balls of radius r have many pairs at distance up to r, but there are no pairs precisely at distances between $2r$ and R.

To prove Lemma 3.13, we define a certain class of paths of total length R/r in the induced subgraph $\Gamma_r[A]$ and prove that this set of paths is large. The start and end of each such path yields a pair vertices of A at distance at most R in the hypercube. We then show that any pair of vertices in the hypercube can be connected by relatively few such paths, yielding a large lower bound on the total number of pairs of vertices in A of distance at most R.

Definition 3. Fix $b \in \{0, 1, \ldots, \lfloor r/2 \rfloor\}$, and assume that r divides R with $r < R$. An (R, b)-*path* is a sequence $(v_0, v_1, \ldots, v_{R/r}) \in \{0,1\}^{d \times (R/r+1)}$ with $d_H(v_{j-1}, v_j) = r$ and $d_H(v_0, v_j) \geqslant j(r - 2b)$ for every $j \in [R/r]$. Let $\pi_{R,b}(A)$ denote the number of (R, b)-paths with all vectors in $A \subseteq \{0,1\}^d$.

Note that an (R, b)-path generalizes a Γ_r shortest path, which is an $(R, 0)$-path. Lemma 3.13 is the immediate consequence of the following two lemmas.

Lemma 3.14. *Fix $b \in \{0, 1, \ldots, \lfloor r/2 \rfloor\}$, and assume that r divides R with $r < R$. Let $A \subseteq \{0,1\}^d$ be a subset and define $N = |A|$ and $M = |\mathsf{E}_r(A)|$. If $\frac{M}{N} \geqslant 4\binom{R-r}{b+1}\binom{d}{r-b-1}$, then*

$$\pi_{R,b}(A) \geqslant N \left(\frac{M}{4N}\right)^{R/r}.$$

Lemma 3.15. *Fix $b \in \{0, 1, \ldots, \lfloor r/2 \rfloor\}$, and assume that r divides R with $r < R$. For any $A \subseteq \{0,1\}^d$,*

$$|\mathsf{E}_{\leqslant R}(A)| \geqslant \frac{\pi_{R,b}(A)}{R! \cdot d^{bR/r}}.$$

More precisely, from Lemmas 3.14 and 3.15 we have the following, which yields Lemma 3.13,

$$\mathsf{e}_{\leqslant R}(A) \;=\; \frac{|\mathsf{E}_{\leqslant R}(A)|}{|A|} \;\geqslant\; \frac{\pi_{R,b}(A)}{R!\cdot d^{Rb/r}\cdot|A|} \;\geqslant\; \frac{e_r(A)^{R/r}}{4^{R/r}\cdot R!\cdot d^{Rb/r}}.$$

We begin by proving Lemma 3.14; its proof is similar to proofs of lower bounds on the number of walks in a graph given by Alon and Rusza [15] and Katz and Tao [98].

Proof of Lemma 3.14. We prove the bound by induction on N. For $N \leqslant 2$, the bound holds trivially. Let δ^* denote the minimum degree in the subgraph of Γ_r induced by A. We first consider the case with $\delta^* \geqslant M/(2N)$. We will count (R,b)-paths $(v_0,\dots,v_{R/r})$ by lower bounding the number of choices for v_{j+1} given the path up to v_j. We will argue that

$$\pi_{R,b}(A) \;\geqslant\; \delta^* N \prod_{j=1}^{R/r-1}\left[\delta^* - \binom{jr}{b+1}\binom{d}{r-b-1}\right]. \tag{3.3}$$

There are $\delta^* N$ possibilities for v_0. For $j = 1,2,\dots,R/r-1$, we say that a vertex v_{j+1} is *bad* if

$$d_H(v_0,v_{j+1}) < d_H(v_0,v_j) + (r-2b).$$

We must exclude the bad neighbors of v_j when choosing v_{j+1}. Each neighbor v_{j+1} of v_j in Γ_r differs from v_j in exactly r coordinates. Crucially, for v_{j+1} to be bad, at least $b+1$ of the differing coordinates must be in the support of $v_0 \oplus v_j$ because each such coordinate used in this step reduces the distance of v_{j+1} from v_0 by exactly 2. Since $|v_0 \oplus v_j| = d_H(v_0,v_j) \leqslant jr$, and the other $r-b-1$ coordinates are arbitrary, there are at most $\binom{jr}{b+1}\binom{d}{r-b-1}$ bad neighbors of v_j in Γ_r. We have assumed that δ^* satisfies

$$\delta^* \geqslant \frac{M}{2N} \geqslant 2\binom{R-r}{b+1}\binom{d}{r-b-1} \geqslant 2\binom{jr}{b+1}\binom{d}{r-b-1}.$$

Since each term in the product in (3.3) is at least $\delta^*/2$, we conclude $\pi_{R,b}(A) \geqslant N\left(\frac{M}{4N}\right)^{R/r}$.

Now assume that the minimum degree δ^* is less than $M/(2N)$. Remove a vertex with degree δ^*. The resulting graph on $N-1$ vertices has larger average degree. By induction, the number of (R,b)-paths on $N-1$ vertices is at least

$$(N-1)\left(\frac{M-\delta^*}{4(N-1)}\right)^{R/r}.$$

We claim this is at least $N(M/(4N))^{R/r}$. Rearranging both sides, we need to show

$$\left(\frac{M-\delta^*}{M}\right)^{(R/r)/((R/r)-1)} \geqslant \frac{N-1}{N}.$$

By the assumptions $\delta^* < M/(2N)$ and $R/r \geqslant 2$, we have, as desired,

$$\left(1-\frac{\delta^*}{M}\right)^{(R/r)/((R/r)-1)} > \left(1-\frac{1}{2N}\right)^{(R/r)/((R/r)-1)} \geqslant \left(1-\frac{1}{2N}\right)^2 > 1-\frac{1}{N}.$$

□

We now finish the proof of Lemma 3.13 by proving Lemma 3.15 which says that the number of pairs of distance $\leqslant R$ in A is relatively large compared to the number of (R,b)-paths in A.

Proof of Lemma 3.15. Consider a pair $(u,v) \in A \times A$ with $d_H(u,v) \leqslant R$. We claim that the number of (R,b)-paths between u and v is at most $R! \cdot d^{bR/r}$. Such a path is specified by a sequence of edges in Γ_r, and once we fix u and v, this sequence is specified by

1. a size R multiset M containing coordinates in $[d]$ differing between adjacent vertices,

2. a partition Π of M into R/r sets of size r.

We claim that there are at most $d^{bR/r}$ choices for M and $R!$ choices for Π. We simply upper bound the count for Π by the number of permutations of R elements. Moving on to M, let $k = d_H(u,v)$ and recall that the (R,b)-path definition requires $k \geqslant R - 2bR/r$. Observe that M contains the k elements in the support of $u \oplus v$. Additionally, M contains $(R-k)$ elements that come in identical pairs. That is, they form a partition into $(R-k)/2$ pairs of identical elements. Once $u \oplus v$ is fixed, these pairs determine M, and there are at most $d^{(R-k)/2} \leqslant d^{bR/r}$ choices for the pairs. □

3.6.2 Proof of Theorem 3.2(b)

As described in our overall proof strategy, our lower bound involves a tradeoff between the average number of times $t = \sum_{i\in[p]} |A_i|/2^d$ that elements of $\{0,1\}^d$ are covered by

$\{A_1, \ldots, A_p\}$ and the edge density of the individual A_i, and hence the size of the intersections of sets $S \sim \mathcal{D}_{n,d}$, which are similar to Hamming balls in structure. The function g that we define below helps us capture this overhead tradeoff.

Definition 11. Define $g : [1, \infty) \to \mathbb{R}^+$ by $\quad g(\tau) = \min\limits_{\substack{(A_1,\ldots,A_p) \in A_\delta^{good} \\ \sum_i |A_i| \leqslant \tau \cdot 2^d}} \mathop{\mathbb{E}}\limits_{S \sim \mathcal{D}_{m,n}} \left[\max\limits_i |A_i \cap S| \right] \cdot \dfrac{p}{n}$

In particular, our overhead tradeoff is encapsulated in the following lemma.

Lemma 3.16. *Let $(A_1, \ldots, A_p) \in A_\delta^{good}$ and let $t = 2^{-d} \sum_i |A_i|$. Then,*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}_{m,n}} [\, \max_i |A_i \cap S| \,] \;\geqslant\; \max\{t, g(t)\} \cdot \frac{n}{p}$$

Proof. We first prove the result for $f_{\leqslant r}(m, n, p, \delta)$. The bound in terms of $g(t)$ follows immediately from the definition of g. For the bound in terms of t, observe that, since each $x \in \{0,1\}^d$ has $\mathbf{Pr}_S[x \in S] = n2^{-d}$, by linearity of expectation we have

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}_{m,n}} [\, \max_i |A_i \cap S| \,] \;\geqslant\; \mathop{\mathbb{E}}_{S \sim \mathcal{D}_{m,n}} \left[\frac{1}{p} \sum_{i \in [p]} |A_i \cap S| \right] \;=\; \frac{1}{p} \sum_{i \in [p]} |A_i| \cdot \frac{n}{2^d} \;=\; t \cdot \frac{n}{p}.$$

□

We prove Theorem 3.2(b) by bounding $\max\{t, g(t)\}$. A lower bound on $\max\{t, g(t)\}$ is a value t^* such that either $t \geqslant t^*$ or $g(t) \geqslant t^*$ for any t. Therefore, we must prove $g(t) \geqslant t^*$ whenever $t \leqslant t^*$. To prove this, we will use the fact that g is a decreasing function and show that the value t^* is such that $g(t) \geqslant t$ whenever $t \leqslant t^*$. Indeed, this implies $g(t) \geqslant g(t^*) \geqslant t^*$ for all $t \leqslant t^*$.

Proof of Theorem 3.2(b). We first state the value of a threshold t^* such that $g(t) \geqslant t$ for all $t \leqslant t^*$ and then argue that it implies Theorem 3.2(b).

Lemma 3.17. *Let n, d, r, p be as in the statement of Theorem 3.2(b) and $R = R(n, d, r)$ be as in Definition 9. Define*

$$\beta = \begin{cases} \gamma/2 & \text{if } \gamma r > 1 \\ \gamma & \text{if } \gamma r \leqslant 1. \end{cases}$$

Then, $g(t) \geqslant t$ *whenever* $1 \leqslant t \leqslant t^*$ *for* $\quad t^* = t^*(n,d,p,r) \;:=\; \dfrac{\delta d^{\beta r - 2r^2/R}}{64 r^r R^{\gamma r}}.$

By construction, R is $\Theta(\log_d n)$ and the assumption that $n \geqslant d^{r \log d}$ implies that $d^{2r^2/R}$ is at most c^r for some constant c. Plugging these values into the formula for t^*, the two bounds of Theorem 3.2(b) follow immediately by combining Lemma 3.9 and Lemma 3.16.

The lower bound proof for $f_r(n,d,p,\delta)$ is almost identical, but it uses a variant definition of g and Lemma 3.16 that replaces the condition $(A_1,\ldots,A_p) \in A_\delta^{good}$ with the condition that the conclusion of Proposition 3.10 holds, which is the only place that membership in A_δ^{good} is used in the proof of Lemma 3.16. □

It only remains to prove Lemma 3.17.

Proof of Lemma 3.17. Let $t \leqslant t^*$ and $(A_1,\ldots,A_p) \in A_\delta^{good}$ be a tuple satisfying $\sum_{i\in[p]} |A_i| = t \cdot 2^d$ and $g(t) = \mathop{\mathbb{E}}_{S\sim\mathcal{D}_{n,d}} [\max_i |A_i \cap S|] \cdot \frac{p}{n}$. Since Lemma 3.12 gives us a lower bound on $g(t)$ in terms of $\sum_i |\mathsf{E}_{\leqslant R}(A_i)|$, our goal is to lower bound the latter quantity, assuming $t \leqslant t^*$. We rewrite this quantity in terms of the distribution μ over $[p]$ with $\mu(i) = \frac{|A_i|}{t2^d}$ as

$$\frac{1}{2^d}\sum_{i\in[p]} |\mathsf{E}_{\leqslant R}(A_i)| \;=\; \frac{1}{2^d}\sum_{i\in[p]} \mathsf{e}_{\leqslant R}(A_i)|A_i| \;=\; t \cdot \mathop{\mathbb{E}}_{i\sim\mu}[\mathsf{e}_{\leqslant R}(A_i)]. \tag{3.4}$$

We apply Lemma 3.13 to lower bound the RHS in (3.4). Let $I \subseteq [p]$ be the indices i such that A_i satisfies $\mathsf{e}_r(A_i) \geqslant 4\binom{R-r}{b+1}\binom{d}{r-b-1}$ for $b = \lfloor \gamma r/2 \rfloor$. Observe that with this choice of b, we have $\beta r \leqslant b+1$ since either $\beta r = \gamma r \leqslant 1 \leqslant b+1$ or $\beta r = \gamma r/2 \leqslant \lfloor \gamma r/2 \rfloor + 1 = b+1$. Using Jensen's inequality for $z \mapsto z^{R/r}$, we obtain

$$\mathop{\mathbb{E}}_{i\sim\mu}[\mathsf{e}_{\leqslant R}(A_i)] \;\geqslant\; \sum_{i\in I} \mu(i) \cdot \frac{\mathsf{e}_r(A_i)^{R/r}}{4^{R/r}\cdot R!\cdot d^{bR/r}} \;\geqslant\; \left(\sum_{i\in I} \mu(i)\mathsf{e}_r(A_i)\right)^{R/r} \cdot \frac{1}{4^{R/r}\cdot R!\cdot d^{bR/r}}. \tag{3.5}$$

Now, since $\mu(i)\mathsf{e}_r(A_i) = \frac{|A_i|e_r(A_i)}{t2^d} = \frac{|\mathsf{E}_r(A_i)|}{t2^d}$, and $(A_1,\ldots,A_p) \in A_\delta^{good}$, Proposition 3.10 implies that

$$\sum_{i\in[p]} \mu(i)\mathsf{e}_r(A_i) \geqslant \frac{\delta}{8t}\binom{d}{r}.$$

Therefore, to lower bound $\mathop{\mathbb{E}}_{i\sim\mu}[\mathsf{e}_{\leqslant R}(A_i)]$, it suffices to upper bound $\sum_{i\notin I}\mu(i)\mathsf{e}_r(A_i)$, which is at most $4\binom{R-r}{b+1}\binom{d}{r-b-1}$ by definition. Now, since $t\leqslant t^*$, $\binom{d}{r}\geqslant(d/r)^r$ and $\beta r\leqslant\min\{b+1,\gamma r\}$, we have

$$\frac{\delta}{16t}\binom{d}{r}\geqslant\frac{\delta}{16t^*}\binom{d}{r}\geqslant 4R^{\gamma r}d^{r-\beta r+2r^2/R}>4R^{b+1}d^{r-b-1}\geqslant\sum_{i\notin I}\mu(i)\mathsf{e}_r(A_i).$$

Therefore,

$$\sum_{i\in I}\mu(i)\mathsf{e}_r(A_i)\geqslant\frac{\delta}{16t}\binom{d}{r}\geqslant 4R^{\gamma r}d^{r-\beta r+2r^2/R},$$

and from (3.5) we have

$$\mathop{\mathbb{E}}_{i\sim\mu}[\mathsf{e}_{\leqslant R}(A_i)]\ \geqslant\ \left(4R^{\gamma r}d^{r-\beta r+2r^2/R}\right)^{R/r}\cdot\frac{1}{4^{R/r}\cdot R!\cdot d^{bR/r}}\ \geqslant\ \frac{R^{\gamma R}d^{R-(\beta R+bR/r)+2r}}{R!}\tag{3.6}$$

Now if $\gamma r\leqslant 1$ then $b=0$ and $\beta=\gamma$ so $\beta R+bR/r=\gamma R$; alternatively, if $\gamma r>1$, then $b\leqslant\gamma r/2$ and $\beta=\gamma/2$ and we have $\beta R+bR/r\leqslant\gamma R$. Therefore, from (3.6) we have

$$\mathop{\mathbb{E}}_{i\sim\mu}[\mathsf{e}_{\leqslant R}(A_i)]\ \geqslant\ \frac{R^{\gamma R}d^{R-\gamma R+2r}}{R!}.\tag{3.7}$$

Using Lemma 3.12 and plugging in (3.4) and (3.7) and the definition of $\gamma=\log_n p$ into the definition of $g(t)$, we have

$$g(t)\ =\ \frac{\mathop{\mathbb{E}}_{S\sim\mathcal{D}_{n,d}}[\max_i|A_i\cap S|]}{n^{1-\gamma}}\ \geqslant\ t\cdot\frac{R^{(r)}r!}{2d^{2r}\cdot n^{1-\gamma}}\cdot\frac{R^{\gamma R}d^{R-\gamma R+2r}}{R!}.$$

Since $R\geqslant 2r\geqslant 2$, we see that $g(t)\geqslant t\cdot\frac{R^{\gamma R}d^{(1-\gamma)R}}{n^{1-\gamma}}$. Since $n\leqslant B(d,R)\leqslant d^R/R!$, we derive $g(t)\geqslant t$ as required. □

3.6.3 Ball Ratios

Proposition 3.18. *For $r<R\leqslant(d+1)/2$, we have*

$$\frac{B(d,R)}{B(d,R-r)}\leqslant\frac{(d+1)d^{r-1}}{R^{(r)}}.$$

Proof. For $i\geqslant 0$ define

$$b_i:=\frac{\binom{d}{R-i}+\cdots+\binom{d}{R}}{\binom{d}{R-i}}.$$

Both $B(d,R)$ and $B(d,R-r)$ contain the common terms $\binom{d}{0}+\cdots+\binom{d}{R-r} \geqslant \binom{d}{R-r}$ and so

$$\frac{B(d,R)}{B(d,R-r)} \leqslant b_r = \frac{\binom{d}{R-r}+\cdots+\binom{d}{R}}{\binom{d}{R-r}}.$$

Observe that by definition $b_0 = 1$ and for $i \geqslant 0$ we have the recurrence

$$b_{i+1} = 1 + \frac{\binom{d}{R-i}}{\binom{d}{R-i-1}} \cdot b_i = 1 + \frac{d-R+i+1}{R-i} \cdot b_i.$$

We prove by induction that for $i \geqslant 1$ we have $2 \leqslant b_i \leqslant (d+1)d^{i-1}/R^{(i)}$. For the base case, since $b_0 = 1$ we have $b_1 = 1 + \frac{d-R+2}{R-1} b_0 = (d+1)/R$ and since $R \leqslant (d+1)/2$, we have $b_1 \geqslant 2$. For $i \geqslant 1$,

$$\begin{aligned}
b_{i+1} &= 1 + \frac{d-R+i+1}{R-i} \cdot b_i \\
&= 1 + \left(\frac{d}{R-i} - \frac{R-i-1}{R-i}\right) \cdot b_i \\
&\leqslant 1 + \frac{d}{R-i} \cdot b_i - \frac{R-i-1}{R-i} \cdot 2 \qquad \text{since } b_i \geqslant 2 \text{ by the inductive hypothesis} \\
&\leqslant \frac{d}{R-i} \cdot b_i. \qquad \text{since } R > r \geqslant i.
\end{aligned}$$

Now applying the inductive hypothesis we have

$$b_{i+1} \leqslant \frac{d}{R-i} \cdot b_i = \frac{d}{R-i} \cdot (d+1)d^{i-1}/R^{(i)} = (d+1)d^i/R^{(i+1)}$$

as claimed. □

3.6.4 Other Lower Bounds

We sketch two simple, but nontrivial, lower bounds for Hamming distance that achieve a different dependence on $|S|$ than the bounds we obtain using isoperimetric inequalities. For the discussion, let L denote the number of vertices a processor receives, not bits.

For the first hard distribution, pick one random ball of radius $r/2$ and include each vector in this ball in S with probability half. A processor may output at most all pairs among

vertices it receives. On the other hand, all pairs in the input S are within distance r. Thus, we need

$$p\binom{L}{2} \geqslant \binom{B(d, r/2)}{2}.$$

Since $|S| = \Theta(B(d, r/2))$, we can rewrite this as $L \geqslant \Omega(|S|/\sqrt{p})$.

For another hard distribution, consider picking $|S|/B(d, r/2)$ random balls each of radius $r/2$, then subsampling each vector with probably half. Receiving L vertices leads to at most $\binom{L}{2}$ close pairs. The input has $\Theta(|S| \cdot B(d, r/2))$ close pairs. This gives

$$L \geqslant \Omega\left(\sqrt{B(d, r/2)} \cdot \sqrt{|S|/p}\right).$$

3.7 Discussion and Future Work

We provided improved parallel algorithms for similarity joins under Hamming distance. We also proved communication lower bounds for one-round, local algorithms. Qualitatively, we showed that an overhead of $d^{\Theta(r)}$ is necessary and sufficient. Although we stated our technical results for Hamming distance, we gave a template for upper and lower bounds in general edge-transitive similarity graphs. A main algorithmic theme running through our results was to perform upfront data replication so that the processors need only one round of communication. This methodology may lead to improved algorithms for other distributed computation tasks.

Local Computation. We optimized for the maximum number of vertices assigned to any one processor and hence the maximum difficulty of the local computation required; however, our focus on communication leaves unanswered details about the actual computation of the close pairs. After the communication, any local algorithm may be used to compare candidates and output pairs satisfying the distance threshold. For example, processors could simply compare all received pairs. Clearly, by enlarging the groups of points sent to each processor we lose the efficiency of being able to only check pairs of points within small subcubes or balls [4, 5, 136]. However, during the local computation stage, the processor may use a local

partitioning scheme, possibly involving subcubes or balls, to reduce the overall number of comparisons. Overall, we believe that further optimizations are needed to implement an efficient all-pairs similarly search. In the ℓ_2 metric, Aiger, Kaplan, and Sharir design an algorithm for finding all close pairs using an edge-covering consisting of randomly shifted and rotated grids [11]. Moreover, they demonstrate improvements over direct LSH approaches. Our approach also extends to support dynamic and streaming data efficiently: only the processors responsible for a vector need to be notified for its addition or deletion. Regarding randomness used for communication, work in the theoretical computer science community suggests bounded independence often suffices and improves performance [59, 125, 144].

Open Questions. We point out five concrete future directions left open by our work.

1. Can we close the gap between our upper and lower bounds on $f_{\leqslant r}(m, n, p, \delta)$? Currently, there is a gap between the exponents of $\gamma r/2$ versus $r/2$.

2. Our randomized edge-covering needed a random construction to find a family of Hamming balls that cover all edges with high probability. Is it possible to remove the error and decrease the communication by exhibiting an explicit construction?

3. Our lower bound holds for the restricted model where processors must communicate sets of vertices. Can we generalize our result and prove a lower bound on the maximum number of *bits* some processor must receive during the one round of communication?

4. The protocol for arbitrary similarity graphs uses copies of the optimal edge-isoperimetric set. Can we determine this set and analyze the resulting protocol for other metrics? Prime candidates include the ℓ_1 distance and the edit distance over small alphabets.

5. Does our edge-covering methodology lead to improvements in practice?

Chapter 4

APPROXIMATE EDGE-ISOPERIMETRIC INEQUALITIES

In Chapter 3, we have demonstrated that optimal similarity join algorithms should be based on edge-isoperimetric sets in the underlying metric space. Here, we prove new results on the maximum number of pairs of binary vectors in a subset $\mathcal{A} \subseteq \{0,1\}^d$ that have distance at most $r \in [d]$ with $r > 1$. As a consequence, we show that our similarity join algorithms are indeed nearly-optimal among a large class of one-round distributed algorithms.

From a mathematical perspective, we uncover new inequalities for Hamming distance similarity graphs. Such inequalities are analogous to the classical result in Euclidean geometry that ℓ_2 balls have maximal volume among sets in $\mathbb{R}^d$ with a given surface area. Our results are based on careful counting, along with combinatorial shifting arguments [16, 38]. They also provide approximate answers to problems posed by many authors [8, 10, 86, 95].

Finally, we note that our upper bounds on edge-isoperimetric inequalities directly imply MapReduce lower bounds, based on the framework of Afrati *et. al.* [136]. Indeed, their lower bounds for Hamming distance one similarity joins already use the edge-isoperimetric inequality for this case. Our results extend these lower bounds for larger distances, and match known upper bounds up to a constant factor.

4.1 Introduction

Discrete edge-isoperimetric sets maximize the edge-density among induced subgraphs of a given size. For regular graphs, this is equivalent to minimizing the edge-boundary, that is, the edges with exactly one endpoint in the set. Some classical results involve the standard binary hypercube, which has edges connecting pairs in $\{0,1\}^d$ with Hamming distance exactly one. Harper [85], Berstein [34], Hart [88], and Lindsey [112] prove that a subset $\mathcal{A} \subseteq \{0,1\}^d$ induces

a subgraph with at most $(1/2)|\mathcal{A}| \log_2 |\mathcal{A}|$ edges; this is tight for integral $a = \log_2 |\mathcal{A}|$ by taking $\mathcal{A}$ to be a subcube of dimension a, that is, fixing the projection onto $d - a$ coordinates and letting a coordinates vary in $\{0,1\}^a$. A survey by Bezrukov provides background on discrete isoperimetric questions for other graphs [36].

We consider powers of the hypercube, that is, the families of graphs that connect distinct pairs in $\{0,1\}^d$ with Hamming distance at most r. We identify $\{0,1\}^d$ with the power-set $\mathcal{P}([d])$ via the bijection $x \leftrightarrow \{i \in [d] : \ x_i = 1\}$. Recall that for subsets $\mathcal{A} \subseteq \{0,1\}^d$, we let $\mathsf{E}_{\leqslant r}(\mathcal{A})$ denote the set of edges in the subgraph of $\Gamma_{\leqslant r}$ induced by $\mathcal{A}$, and we denote

$$\mathsf{e}_{\leqslant r}(\mathcal{A}) = \frac{|\mathsf{E}_{\leqslant r}(\mathcal{A})|}{|A|}.$$

Note that $\Gamma_{\leqslant r}$ is a regular graph, and it has

$$\mathsf{e}_{\leqslant r}\left(\{0,1\}^d\right) = \frac{1}{2} \cdot \sum_{j=1}^{r} \binom{d}{j}$$

total edges. Thus,

$$\mathsf{e}_{\leqslant r}(\mathcal{A}) \leqslant \frac{1}{2} \cdot \sum_{j=1}^{r} \binom{d}{j}$$

for any $\mathcal{A} \subseteq \{0,1\}^d$. We will be interested in tighter upper bounds on $\mathsf{e}_{\leqslant r}(\mathcal{A})$.

Restricting to k-sets in $\{0,1\}^d$, the $r = 2$ edge-isoperimetric problem has been called the "Kleitman-West Problem" by Harper [86]. The corresponding subgraph of $\Gamma_{\leqslant r}$ connects pairs of k-sets that agree in $k - 1$ elements. Although the edge-isoperimetric problem is well-posed in this case, no general solutions are known, except when $k \leqslant 2$ [8, 10]. Harper attempted to resolve the edge-isoperimetric problem in this case via a continuous relaxation [86]. Unfortunately, his proof only holds for certain very special cases, and he later demoted his result to a conjecture [87]. Ahlswede and Cai also studied the Kleitman-West Problem [8], and they provide a counter example to a natural conjecture by Kleitman about the optimal sets [7]. Ahlswede and Katona also provide related results and connections to other edge-isoperimetric questions [10].

Kahn, Kalai, and Linial [95] ask about exact or even approximate solutions for $r \geqslant 2$. They observe that for very large sets $\mathcal{A}$ with $|\mathcal{A}| = 2^{d-1}$, the only solutions for odd r are

subcubes on $d-1$ coordinates. For even r, the set of vectors with even Hamming weight provides a second optimal solution. Furthermore, they mention that spectral techniques (*i.e.,* considering eigenvalues of the adjacency matrix of $\Gamma_{\leqslant r}$) are insufficient for proving good bounds for the edge-isoperimetric question.

The edge-isoperimetric question for very small subsets of $\Gamma_{\leqslant r}$ has a well-understood solution (*e.g.,* when $\mathcal{A}$ can be contained in a Hamming ball of radius $r/2$). It reduces to understanding subsets of $\{0,1\}^d$ with bounded diameter. The optimal construction differs slightly depending on the parity of r. For even r, take $\mathcal{A}$ to be a subset of a Hamming ball of radius $r/2$. For odd r, take $\mathcal{A}$ to be any subset of a ball of radius $(r+1)/2$ such that the vectors at distance $(r+1)/2$ from the center all share a common element in $[d]$. In both cases, $\mathcal{A}$ contains $\binom{|\mathcal{A}|}{2}$ pairs with distance at most r, and this is clearly the maximum possible. Kleitman has proven that these constructions are optimal [101]. More precisely, he showed that when all pairs in $\mathcal{A}$ have distance at most r, then

- $|\mathcal{A}| \leqslant \sum_{j=0}^{r/2} \binom{d}{j}$, if r is even;
- $|\mathcal{A}| \leqslant \sum_{j=0}^{(r-1)/2} \binom{d}{j} + \binom{d-1}{\frac{r-1}{2}}$, if r is odd.

Ahlsede and Katona [9] prove that Kleitman's bound is equivalent to an earlier result of Katona [97]. We will be interested in understanding how densely packed (in terms of edges) sets can be when their sizes exceed the bounds of Kleitman (and Katona).

A core challenge for analyzing the edge-isoperimetric problem for $\Gamma_{\leqslant r}$ and $r \geqslant 2$ is that solutions are not nested in general. More precisely, there is no ordering on the elements of $\{0,1\}^d$ such that the optimal set of size m consists of the first m elements in the ordering. This can be seen even for $\Gamma_{\leqslant 2}$ with $d=4$ dimensions. Indeed, for $r=2, d=4$, considering only left-compressed down-sets, we see that the optimal set of size five is the Hamming ball of radius one, whereas the optimal set of size seven is the set of vectors with weight at most two supported on the first three coordinates, that is, $\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{1,3\}\}$.

Similar examples show that the Kleitman-West problem also does not have nested solutions in general. Nonetheless, we show that compression-based arguments can lead to nearly-tight bounds on $\mathsf{e}_{\leqslant r}(\mathcal{A})$, approximating the truth up to a constant depending on r.

Our Results

The edge-isoperimetric inequality differs depending on whether the distance r is even or odd. We state our theorems in terms of $\ell = \ell(|\mathcal{A}|) = \min\left\{\left\lceil \frac{2\log|\mathcal{A}|}{\log d - \log\log|\mathcal{A}|}\right\rceil, \lfloor \log|\mathcal{A}|\rfloor\right\}$.

Theorem 4.1. *Let $\mathcal{A} \subset \{0,1\}^d$ be a set and t be an integer with $1 \leqslant t \leqslant \log|\mathcal{A}|$. Then,*

$$\mathsf{e}_{\leqslant 2t}(\mathcal{A}) \;\leqslant\; \left(\frac{8e}{t}\right)^{2t} \cdot (d\cdot\ell)^t.$$

Theorem 4.2. *Let $\mathcal{A} \subset \{0,1\}^d$ be a set and t be an integer with $1 \leqslant t \leqslant \log|\mathcal{A}|$. Then,*

$$\mathsf{e}_{\leqslant 2t+1}(\mathcal{A}) \;\leqslant\; \left(\frac{16e}{2t+1}\right)^{2t+1} \cdot (d\cdot\ell)^t \cdot \log|\mathcal{A}|.$$

Note that the second term in the minimum for ℓ kicks in when $\frac{\log|\mathcal{A}|}{\log d - \log\log|\mathcal{A}|} \geqslant \log|\mathcal{A}|$, or in other words, when $|\mathcal{A}| \geqslant 2^{d/2}$. The theorems are tight up to the values $8e$ and $16e$, as this is witnessed by Hamming balls for even r and by certain products of Hamming balls and subcubes for odd r. We do not optimize these constants, except in the case of $r = 2$, where we prove a tighter bound (Theorem 4.5). Finally, observe that for $r \geqslant 2\log|\mathcal{A}|$, then all pairs could have distance at most $2\log|\mathcal{A}|$, and the trivial bound $\mathsf{e}_{\leqslant r}(\mathcal{A}) \leqslant \binom{|\mathcal{A}|}{2}$ is tight.

Determining the precise isoperimetric sets of all sizes remains a challenging open problem (which seems beyond the reach of our techniques). As mentioned above, even the restriction to k-sets and $r = 2$ is open for $k \geqslant 3$, that is, the Kleitman-West problem remains unsolved.

Notation and Preliminaries

In what follows, e denotes the base of the natural logarithm and all logarithms have based two (*e.g.* , define $\log z := \log_2 z$).

We say $A \subseteq \{0,1\}^d$ is a *down-set* if $x \in A$ implies $y \in A$ whenever $y \subseteq x$. We say A is *left-compressed* if $x \in A$ implies $y \in A$ whenever y satisfies the two conditions (1) $|x| = |y|$ and (2) either $x_1 = 0, y_1 = 1$ or there exists $i, j \in [d]$ with $1 < i < j$ such that $x_1 = y_1, x_2 = y_2, \ldots, x_{i-1} = y_{i-1}$ and $x_i = 0, x_j = 1, y_i = 1, y_j = 0$. Standard compression arguments (see, for example, [8, 16, 38, 87]) imply the following.

Proposition 4.3. *Let d, m be positive integers with $m \leqslant 2^d$. Among subsets of $\{0,1\}^d$ of size m, a left-compressed down-set $\mathcal{A}$ achieves the maximum of $\mathsf{e}_{\leqslant r}(\mathcal{A})$.*

Proof. We start with the down-set claim, then we move on to left-compressed. Define the down-shift operator D_i on sets $B \subseteq \{0,1\}^d$ for a coordinate $i \in [d]$. The set $D_i(B)$ is obtained from B as follows: Replace every $z \in B$ such that both (i) $z_i = 1$, and (ii) the i'th neighbor[1] of z is not in B, with the i'th neighbor of z. It is well-known that the set

$$D(B) = D_1(D_2(\cdots D_d(B)))$$

is a down-set. We show that the set $D_i(B)$ contains at least as many pairs at distance r as the set B. Let $x, y \in B$ with $x_i = y_i = 1$. Let x^i be the vector agreeing with x except with a zero in the ith position (likewise for and y^i). We analyze the vectors x, x^i, y, y^i and argue the set of pairs associated with these vectors either has the same multi-set of distances after applying D_i or has one closer pair. If both x^i and y^i are in B, or if neither are in B, then the claim is true. Otherwise, assume B contains x, y, y^i but not x^i. This implies $D_i(B)$ contains x^i, y, y^i but not x. Therefore, the two distances are swapped, because $d_{\mathrm{H}}(x^i, y) = d_{\mathrm{H}}(x, y^i)$ and $d_{\mathrm{H}}(x^i, y^i) = d_{\mathrm{H}}(x, y)$. Since this holds for all x, x^i, y, y^i, we are done.

We move on to left-compressed, and we use the operator $L_{i,j}$ on sets $B \subseteq \{0,1\}^d$ for coordinates $i < j \in [d]$. The set $L_{i,j}(B)$ is obtained from B as follows. For each $z \in B$, if $z_i = 0$ and $z_j = 1$, then replace z with the vector that is the same as z except with coordinates i and j swapped, unless this vector already exists in B. We will be interested in the set

$$L(B) = L_{1,2}(L_{1,3}(\cdots L_{d-1,d}(B))),$$

[1] Two vectors are *i'th-neighbors* if they differ in coordinate i and are the same elsewhere.

and it is easy to see that it is left-compressed. We first prove that applying $L_{i,j}$ retains the property of being a down-set, and then we show that it does not decrease the number of edges. Without loss of generality, we look at $i = 1$ and $j = 2$. Consider a vector $x \in L_{1,2}(B)$ and any $y \subseteq x$. If $x \in B$ and $L_{1,2}(x) = x$, then the down-set property of B implies $L_{1,2}(y) = y$. Assume $x \notin B$, so that $x = 10x' \in L_{1,2}(B)$ and $01x' \in B$. When $y = 00y'$, then by the down-set property of B, we have that $00y' \in B$, and thus $00y' \in L_{1,2}(B)$. When $y = 10'$, then we know $01y' \in B$, since $01x' \in B$. Therefore, either $10y' \in B$ already, or we have $10y' \notin B$, which implies $10y' \in L_{1,2}(B)$ as desired.

We show that the set $L_{1,2}(B)$ contains at least as many pairs at distance at most r as the set B. Consider a pair x, y with $d_{\mathrm{H}}(x, y) \leqslant r$.

- If $L_{1,2}$ acts non-identically on both or neither of x, y, then the distance is preserved.
- If either x or y leads with two zeros or two ones, then the distance is preserved.
- If $x = 10x'$ and $y = 01y'$, and $01x' \notin B$ and $10y' \notin B$, then $10y' \in L_{1,2}(B)$. The number of edges does not decrease because $d_{\mathrm{H}}(10x', 10y') < d_{\mathrm{H}}(10x', 01y')$.
- If $x = 01x' \in B$ and $y = 01y' \in B$ and $10x' \in B$ and $10y' \notin B$, then $10y' \in L_{1,2}(B)$. Notice that the new pair of distances simply is swapped from the original, or more precisely, $d_{\mathrm{H}}(01x', 10y') = d_{\mathrm{H}}(01y', 10x')$, and $d_{\mathrm{H}}(10x', 10y') = d_{\mathrm{H}}(01x', 10y')$.

Since $L(D(B))$ is a left-compressed down-set, with the same cardinality as B, and no fewer pairs at distance at most r, the proof is complete. □

Proposition 4.4. *Let $\mathcal{A} \subseteq \{0,1\}^d$ be a down-set. For every $x \in \mathcal{A}$, we have $|x| \leqslant \lfloor \log |\mathcal{A}| \rfloor$.*

Proof. Since $x \in \mathcal{A}$, we also have $y \in \mathcal{A}$ for all $y \subseteq x$. The number of such y is $2^{|x|} \leqslant |\mathcal{A}|$. □

Remark 1. Proposition 4.3 and Proposition 4.4 imply $\mathsf{e}_{\leqslant 1}(\mathcal{A}) \leqslant \lfloor \log |\mathcal{A}| \rfloor$. Indeed, for a down-set $\mathcal{A}$, we have $|\mathsf{E}_{\leqslant 1}(\mathcal{A})| = \sum_{x \in \mathcal{A}} |x| \leqslant |\mathcal{A}| \cdot \lfloor \log |\mathcal{A}| \rfloor$. This approximates, up to a factor of two, the optimal bound $\mathsf{e}_{\leqslant 1}(\mathcal{A}) \leqslant (1/2) \cdot \lfloor \log |\mathcal{A}| \rfloor$ mentioned above [34, 85, 88, 112].

4.2 The distance two case

The special case of our theorem for $r = 2$ has a fairly simple proof, leading to a tighter bound.

Theorem 4.5. *Let $\mathcal{A} \subset \{0,1\}^d$ satisfy $1 \leqslant \log|\mathcal{A}| < d$. Then*

$$\mathsf{e}_{\leqslant 2}(\mathcal{A}) \leqslant d \cdot \ell',$$

where $\ell' := \min\left\{\left\lceil \frac{\log|\mathcal{A}|}{\log d - \log\log|\mathcal{A}|}\right\rceil, \lfloor \log|\mathcal{A}| \rfloor\right\}$

Using an observation of Ahlswede and Cai [8], we reduce the problem to bounding the "sum of ranks" of elements in $\mathcal{A}$. We provide a proof for completeness. Define the rank of x as

$$\|x\| := \sum_{j\in[d]} jx_j = \sum_{j\in x} j.$$

Lemma 4.6. *Let $\mathcal{A}$ be a left-compressed down-set. Then,* $|\mathsf{E}_{\leqslant 2}(\mathcal{A})| = \sum_{x\in\mathcal{A}} \|x\|$.

Proof. Notice that $\{x,y\} \in \mathsf{E}_{\leqslant 2}(\mathcal{A})$ implies that either $\|y\| < \|x\|$ or vice versa. We fix $x \in \mathcal{A}$ and count y such that $\|y\| < \|x\|$. Assume that $x \neq \emptyset, \{1\}$, or the bound is trivial. We separate the cases $|y| = |x|$ and $|y| < |x|$. In the first case, we count y of the form $y = x \cup \{i\} \setminus \{j\}$, where $i < j$, $j \in x$ and $i \notin x$. The number of such y is exactly

$$\sum_{j\in x}\Big(j - 1 - |\{i \in x :\ i < j\}|\Big) \ = \ \|x\| - \binom{|x|+1}{2}.$$

For the second case, with $|y| < |x|$, there are $\binom{|x|+1}{2}$ choices for y of the form $y = x \setminus \{i,j\}$ or $y = x \setminus \{i\}$, where $i, j \in x$. Since we have assumed that $\mathcal{A}$ is a left-compressed down-set, the counted pairs in both cases are in $\mathsf{E}_{\leqslant 2}(\mathcal{A})$. Summing over $x \in \mathcal{A}$ completes the proof. □

To obtain Theorem 4.5, it suffices to upper bound $\|x\|$.

Lemma 4.7. *Let $\mathcal{A} \subset \{0,1\}^d$ be a left-compressed down-set with $|\mathcal{A}| \geqslant 2$. For any $x \in \mathcal{A}$,*

$$\|x\| \leqslant d \cdot \ell',$$

where $\ell' = \min\left\{\left\lceil \frac{\log|\mathcal{A}|}{\log d - \log\log|\mathcal{A}|}\right\rceil, \lfloor \log|\mathcal{A}| \rfloor\right\}$

Proof. Proposition 4.4 implies that $|x| \leqslant \lfloor \log |\mathcal{A}| \rfloor$, and thus, $\|x\| \leqslant d|x| \leqslant d\lfloor \log |\mathcal{A}| \rfloor$. Therefore, we may assume that we are in the case when $\ell' = \lceil \frac{\log |\mathcal{A}|}{\log d - \log\log |\mathcal{A}|} \rceil$.

Using that $\mathcal{A}$ is a left-compressed down-set, we will lower bound the number of $y \in \mathcal{A}$ that are guaranteed in $\mathcal{A}$ by the existence of $x \in \mathcal{A}$. To this end, define $\beta' := \left\lfloor \frac{d\ell'}{\log |\mathcal{A}|} \right\rfloor$, and let $x = x' \cup x''$, where $x' \subseteq \{1, \ldots, \beta'\}$ and $x'' \subseteq \{\beta' + 1, \ldots, d\}$ correspond to the integers in x with values at most β' and at least $\beta' + 1$, respectively (with $|x| = |x'| + |x''|$). We will show that

$$\|x\| \leqslant \beta'|x'| + d|x''| \leqslant d\ell'.$$

Notice that if $|x''| = 0$, then $\|x\| = \beta'|x'| = \beta'|x| \leqslant \beta' \log |\mathcal{A}| \leqslant d\ell'$, where the inequalities use Proposition 4.4 and the definition of β'. Thus, assume that $|x'| \leqslant |x| - 1$ and $|x''| \geqslant 1$.

Consider $y \in \{0,1\}^d$ of the form $y = y' \cup y'' \cup y'''$, where $y' \subseteq x'$, $y'' \subseteq x''$, and $y''' \subseteq \{1, \ldots, \beta'\}$. We claim y of this form is in $\mathcal{A}$ whenever $|y'''| \leqslant |x'' \setminus y''|$ and $y''' \cap x' = \emptyset$. Indeed, this follows directly from the left-compressed down-set assumption. To count such $y \in \mathcal{A}$, first define $\varepsilon_x \in [0, 1)$ as the real number satisfying $2^{|x'|} = |\mathcal{A}|^{\varepsilon_x}$. We will show $|x''| \leqslant (1 - \varepsilon_x)\ell'$. Clearly, there are $2^{|x'|} = |\mathcal{A}|^{\varepsilon_x}$ choices for $y' \subseteq x'$. Then, for (y'', y'''), we lower bound their count by

$$\sum_{j=0}^{|x''|} \binom{\beta' - |x'|}{j} \binom{|x''|}{j}$$

where the j^{th} term counts (y'', y''') with $|x'' \setminus y''| = |y'''| = j$ and $y''' \cap x' = \emptyset$. Since the choice of y' is independent of (y'', y'''), we know that the sum above must be at most $|\mathcal{A}|^{1-\varepsilon_x}$, otherwise we would have guaranteed more than $|\mathcal{A}|$ distinct y in $\mathcal{A}$.

Suppose that $|x''| \geqslant \lceil (1 - \varepsilon_x)\ell' \rceil$ and $\varepsilon_x \leqslant 1 - 1/\ell'$. Observe that the facts $\beta' \geqslant \log |\mathcal{A}|$ and $|x'| = \varepsilon_x \log |\mathcal{A}|$ together imply $\beta' - |x'| \geqslant (1 - \varepsilon_x)\beta'$. Then,

$$\sum_{j=0}^{|x''|} \binom{\beta' - |x'|}{j} \binom{|x''|}{j} > \left(\frac{\beta' - |x'|}{(1 - \varepsilon_x)\ell'} \right)^{(1-\varepsilon_x)\ell'} \geqslant \left(\frac{(1 - \varepsilon_x)d\ell'}{(1 - \varepsilon_x)\ell' \log |\mathcal{A}|} \right)^{(1-\varepsilon_x)\ell'} \geqslant |\mathcal{A}|^{1-\varepsilon_x},$$

where the final inequality uses that $\ell' = \lceil \frac{\log |\mathcal{A}|}{\log d - \log\log |\mathcal{A}|} \rceil$, which implies that $(d/\log |\mathcal{A}|)^{\ell'} \geqslant |\mathcal{A}|$. We have already noted that the sum above must be at most $|\mathcal{A}|^{1-\varepsilon_x}$. To avoid contradicting

the size of $\mathcal{A}$, we must have $|x''| \leqslant \lceil (1-\varepsilon_x)\ell' \rceil - 1 < (1-\varepsilon_x)\ell'$, and thus

$$\|x\| \;\leqslant\; \beta'|x'| + d|x''| \;=\; \beta'\varepsilon_x \log|\mathcal{A}| + d|x''| \;\leqslant\; \varepsilon_x d\ell' + (1-\varepsilon_x)d\ell' \;=\; d\ell'.$$

□

We now complete the proof of Theorem 4.5.

Proof of Theorem 4.5. Applying Proposition 4.3, we may assume that $\mathcal{A}$ is a left-compressed down-set. Then, Lemma 4.6 and Lemma 4.7 together imply the desired bound:

$$\mathsf{e}_{\leqslant 2}(\mathcal{A}) = \frac{1}{|\mathcal{A}|} \cdot \sum_{x\in\mathcal{A}} \|x\| \leqslant d \cdot \ell'.$$

□

4.3 *The general case for even* r

In this section, we prove Theorem 4.1.

Proof Overview

We start with some notation. For $(b,a) \in \mathbb{Z}^2_{\geqslant 0}$, let

$$E_{(b,a)}(\mathcal{A}) := \{\{x,y\} \in \mathsf{E}_{\leqslant 2t}(\mathcal{A}) :\; |x \setminus y| = b,\; |y \setminus x| = a\}.$$

and define $\mathsf{e}_{(b,a)}(\mathcal{A}) := |E_{(b,a)}(\mathcal{A})|/|\mathcal{A}|$. Letting

$$\mathcal{U} = \{(b,a) \in \mathbb{Z}^2_{\geqslant 0} :\; b \geqslant a \text{ and } b+a \leqslant 2t\},$$

observe that we can decompose $\mathsf{E}_{\leqslant 2t}(\mathcal{A})$ as a disjoint union

$$\mathsf{E}_{\leqslant 2t}(\mathcal{A}) = \bigcup_{(b,a)\in\mathcal{U}} E_{(b,a)}(\mathcal{A}),$$

and in particular, this implies,

$$\mathsf{e}_{\leqslant 2t}(\mathcal{A}) = \sum_{(b,a)\in\mathcal{U}} \mathsf{e}_{(b,a)}(\mathcal{A}). \tag{4.1}$$

Our strategy will be to prove upper bounds on $\mathsf{e}_{(b,a)}(\mathcal{A})$, then combine these for the overall theorem. We will need a variant of the bound on $|x''|$ from the proof of Lemma 4.7. In what follows, we express our results using integers $\ell := \ell(|\mathcal{A}|)$ and $\beta := \beta(\mathcal{A})$, defined in the next proposition. We also define $\ell_x := |x \cap \{\beta+1, \dots, d\}|$ for $x \in \mathcal{A}$. Intuitively, β is the threshold for "big" elements; ℓ_x is the number of these "large" elements; and, we will show $\ell_x \leqslant \ell$.

Proposition 4.8. *Let $\mathcal{A} \subset \{0,1\}^d$ be a down-set with $|\mathcal{A}| \geqslant 2$. Define*

$$\ell := \min\left\{ \left\lceil \frac{2\log|\mathcal{A}|}{\log d - \log\log|\mathcal{A}|} \right\rceil, \lfloor \log|\mathcal{A}| \rfloor \right\} \quad \beta := \left\lfloor \left(\frac{d}{\log|\mathcal{A}|}\right)^{1/2} \ell \right\rfloor$$

For any $x \in \mathcal{A}$, we have (i) $|x| \cdot \beta \leqslant d\ell$ and (ii) $\beta^2 \leqslant d\ell$ and (iii) $\log^2|\mathcal{A}| \leqslant d\ell$.

Proof. All parts immediately follow from Proposition 4.4, the definitions of ℓ and β, and the facts $\log|\mathcal{A}| < d$ and $\log(d/\log|\mathcal{A}|) \leqslant (d/\log|\mathcal{A}|)$ □

Lemma 4.9. *Let $\mathcal{A} \subset \{0,1\}^d$, $|\mathcal{A}| \geqslant 2$ be a left-compressed down-set. If $x \in \mathcal{A}$, then $\ell_x \leqslant \ell$.*

Proof. Proposition 4.4 implies $|x| \leqslant \lfloor \log|\mathcal{A}| \rfloor$, and clearly $\ell_x \leqslant |x|$, so we may assume that we are in the case when $\ell = \lceil \frac{2\log|\mathcal{A}|}{\log d - \log\log|\mathcal{A}|} \rceil$. Let $x = x' \cup x''$ where $x' \subseteq \{1, \dots, \beta\}$ and $x'' \subseteq \{\beta+1, \dots, d\}$. By definition, $|x''| = \ell_x$, and since $\mathcal{A}$ is a down-set, we know that $x'' \in \mathcal{A}$. Now, since $\mathcal{A}$ is left-compressed, we have that $x'' \in \mathcal{A}$ implies

$$|\mathcal{A}| \geqslant \sum_{j=0}^{\ell_x} \binom{\beta}{j}\binom{\ell_x}{j}.$$

This implies $\ell_x \leqslant \lceil 2\log|\mathcal{A}|/\log(d/\log|\mathcal{A}|) \rceil = \ell$, or else it contradicts the size of $\mathcal{A}$. □

In what follows, let $\mathcal{A} \subseteq \{0,1\}^d$ be a left-compressed down-set with $1 \leqslant \log|\mathcal{A}| < d$. Let ℓ, β be defined as in Proposition 4.8. Recall that $\ell_x = |x \cap \{\beta+1, \dots, d\}|$ equals the number of large elements in $x \in \mathcal{A}$. In our proofs, it will be helpful to order $\{0,1\}^d$ based on ℓ_x. In particular, we upper bound $\mathsf{e}_{(b,a)}(\mathcal{A})$ by partitioning the pairs $\{x,y\} \in \mathsf{E}_{(b,a)}(\mathcal{A})$ into two sets, based on the cases $\ell_y \leqslant \ell_x$ and $\ell_y > \ell_x$. By the definition of $\mathsf{E}_{(b,a)}(\mathcal{A})$, with $b \geqslant a$, we always have $|x| \geqslant |y|$. Ordering based on ℓ_x and ℓ_y enables us to use different arguments in the two cases: when $\ell_y \leqslant \ell_x$, we count pairs based on x, and when $\ell_y > \ell_x$, we count pairs based on y.

The case when $\ell_y \leqslant \ell_x$

Lemma 4.10. *Let b, a be nonnegative integers with $b \geqslant a$ and $1 \leqslant b + a \leqslant 2\log|\mathcal{A}|$.*

- *If $b + a$ is even, then*

$$|\{\{x, y\} \in E_{(b,a)}(\mathcal{A}) : \ \ell_y \leqslant \ell_x\}| \leqslant \left(\frac{4\sqrt{2}e}{b+a}\right)^{(b+a)} \cdot (d \cdot \ell)^{(b+a)/2} \cdot |\mathcal{A}|.$$

- *If $b + a$ is odd, then*

$$|\{\{x, y\} \in E_{(b,a)}(\mathcal{A}) : \ \ell_y \leqslant \ell_x\}| \leqslant \left(\frac{4\sqrt{2}e}{b+a}\right)^{b+a} \cdot (d \cdot \ell)^{(b+a-1)/2} \cdot \log|\mathcal{A}| \cdot |\mathcal{A}|.$$

Proof. Fix $x \in \mathcal{A}$. For each $p \in [a] \cup \{0\}$, we will bound the number of $y \in \{0,1\}^d$ such that $\{x, y\} \in E_{(b,a)}(\mathcal{A})$ and $\ell_y \leqslant \ell_x$ and $|(y \setminus x) \cap \{\beta + 1, \ldots, d\}| = p$. We claim that the number of such y is at most

$$\binom{d - \beta - \ell_x}{p}\binom{\ell_x}{p}\binom{\beta - |x| + \ell_x}{a - p}\binom{|x|}{b - p}. \tag{4.2}$$

Indeed, the first two factors count the ways to replace p elements in x with p new elements that are larger than β, and the final two factors count the ways to replace $b - p$ elements in x with $a - p$ new elements that are at most β.

Recall that Lemma 4.9 implies that $\ell_x \leqslant \ell$. Then, the quantity in (4.2) is at most

$$\binom{d}{p}\binom{\ell}{p}\binom{\beta}{a-p}\binom{|x|}{b-p} \ \leqslant \ \frac{(d\ell)^p \cdot \beta^{a-p}|x|^{b-p}}{(p!)^2 \cdot (a-p)! \cdot (b-p)!} \tag{4.3}$$

We break (4.3) into two cases, based on the parity of $b + a$. For both cases, recall that Proposition 4.8 implies that $\beta|x| \leqslant d\ell$ and $\beta^2 \leqslant d\ell$ and $|x|^2 \leqslant d\ell$.

The case when $b + a$ is even. We bound the numerator of the RHS of (4.3) by

$$(d\ell)^p \cdot \beta^{a-p}|x|^{b-p} \leqslant (d\ell)^p \cdot (d\ell)^{(a-p)/2} \cdot (d\ell)^{(b-p)/2} = (d\ell)^{(b+a)/2}.$$

We lower bound the denominator of the RHS of (4.3) by

$$(p!)^2 \cdot (b-p)! \cdot (a-p)! \;\geqslant\; \frac{p^{2p}(b-p)^{b-p}(a-p)^{a-p}}{e^{b+a}} \;\geqslant\; \left(\frac{b+a}{4e}\right)^{b+a}, \tag{4.4}$$

where the first inequality uses Stirling's inequality. The second inequality follows from the fact that for all $i, j \in \mathbb{R}$ with $i \geqslant 1$ and $j \geqslant 1$, we have $i^i j^j \geqslant \left(\frac{i+j}{2}\right)^{i+j}$, which we now prove by showing that the LHS is minimized with $i = j$. Suppose that $i \leqslant j$ and we fix $i + j = \kappa$, so that the RHS is $(\kappa/2)^\kappa$. Taking logs, we want to minimize $i \ln i + (\kappa - i)\ln(\kappa - i)$. Differentiating with respect to i gives $\ln i - \log(\kappa - i)$, and this is zero at $i = \kappa/2$ and otherwise an increasing function of i in this range (the second derivative is $1/i + 1/(\kappa - i) > 0$).

Summing the above bound on (4.3) over $p \in [a] \cup \{0\}$ and employing (4.4),

$$\begin{aligned}
|\{y \in \mathcal{A} : \{x, y\} \in E_{(b,a)}(\mathcal{A}),\ \ell_y \leqslant \ell_x\}| \;&\leqslant\; \sum_{p=0}^{a} \frac{(d\ell)^{(b+a)/2}}{(p!)^2 \cdot (b-p)! \cdot (a-p)!} \\
&\leqslant\; (a+1) \cdot \frac{(d\ell)^{(b+a)/2}\,(4e)^{(b+a)}}{(b+a)^{b+a}} \\
&\leqslant\; \frac{(d\ell)^{(b+a)/2}\,(4\sqrt{2}e)^{(b+a)}}{(b+a)^{b+a}},
\end{aligned}$$

where the last inequality uses that $(a+1) \leqslant (\sqrt{2})^{b+a}$, leading to the factor $(4\sqrt{2}e)^{(b+a)}$.

The case when $b + a$ is odd. In this case $b \geqslant a + 1 \geqslant p + 1$. We recall that $|x| \leqslant \log|\mathcal{A}|$, and we upper bound the RHS of (4.3) by

$$(d\ell)^p \cdot \beta^{a-p}|x|^{b-p} \leqslant (d\ell)^p \cdot (d\ell)^{(a-p)/2} \cdot (d\ell)^{(b-p-1)/2} \cdot \log|\mathcal{A}| = (d\ell)^{(b+a-1)/2} \cdot \log|\mathcal{A}|.$$

Summing the above bound on (4.3) over $p \in [a] \cup \{0\}$ and employing (4.4),

$$\begin{aligned}
|\{y \in \mathcal{A} : \{x, y\} \in E_{(b,a)}(\mathcal{A}),\ \ell_y \leqslant \ell_x\}| \;&\leqslant\; \sum_{p=0}^{a} \frac{(d\ell)^{(b+a-1)/2} \cdot \log|\mathcal{A}|}{(p!)^2 \cdot (b-p)! \cdot (a-p)!} \\
&\leqslant\; \frac{(d\ell)^{(b+a-1)/2}\,(4\sqrt{2}e)^{(b+a)} \cdot \log|\mathcal{A}|}{(b+a)^{b+a}}
\end{aligned}$$

In both even and odd cases, summing over $x \in \mathcal{A}$ completes the proof. □

The case when $\ell_y > \ell_x$

Lemma 4.11. *Let b, a be nonnegative integers with $b \geqslant a$ and $1 \leqslant b + a \leqslant 2 \log |\mathcal{A}|$.*

- *If $b + a$ is even, then*

$$|\{\{x, y\} \in E_{(b,a)}(\mathcal{A}) : \ \ell_y > \ell_x\}| \leqslant \left(\frac{4\sqrt{2}e}{b+a}\right)^{(b+a)} \cdot (d \cdot \ell)^{(b+a-2)/2} \cdot \ell\beta \cdot |\mathcal{A}|.$$

- *If $b + a$ is odd, then*

$$|\{\{x, y\} \in E_{(b,a)}(\mathcal{A}) : \ \ell_y > \ell_x\}| \leqslant \left(\frac{4\sqrt{2}e}{b+a}\right)^{b+a} \cdot (d \cdot \ell)^{(b+a-1)/2} \cdot \ell \cdot |\mathcal{A}|.$$

Proof. Fix $y \in \mathcal{A}$. For each $p \in [a]$, we will bound the number of $x \in \{0,1\}^d$ such that $\{x, y\} \in E_{(b,a)}(\mathcal{A})$ and $\ell_y > \ell_x$ and $|(x \setminus y) \cap \{\beta + 1, \ldots, d\}| = p - 1$. We claim that the number of such x is at most

$$\binom{d - \beta - \ell_y}{p-1}\binom{\ell_y}{p}\binom{\beta - |x| + \ell_y}{b-p+1}\binom{|y|}{a-p}. \tag{4.5}$$

Indeed, the first two factors count the ways to replace p elements in y with $p-1$ new elements that are larger than β, and the final two factors count the ways to replace $a - p$ elements in y with $b - p + 1$ new elements that are at most β.

Recall that Lemma 4.9 implies that $\ell_y \leqslant \ell$. Then, the quantity in (4.5) is at most

$$\binom{d}{p-1}\binom{\ell}{p}\binom{\beta}{b-p+1}\binom{|y|}{a-p} \ \leqslant \ \frac{(d\ell)^{p-1} \cdot \ell \cdot \beta^{b-p+1} \cdot |y|^{a-p}}{(p-1)! \cdot p! \cdot (b-p+1)! \cdot (a-p)!} \tag{4.6}$$

We break into two cases, based on the parity of $b+a$. For both cases, recall that Proposition 4.8 implies that $\beta|y| \leqslant d\ell$ and $\beta^2 \leqslant d\ell$ and $|y|^2 \leqslant d\ell$.

The case when $b + a$ is even. Notice that $\ell_y > \ell_x$ and $|x| \geqslant |y|$ implies $a \geqslant 1$ and $b + a \geqslant 2$. We upper bound the numerator of the RHS of (4.6) by

$$(d\ell)^{p-1} \cdot \ell \cdot \beta^{b-p+1} \cdot |y|^{a-p} \leqslant (d\ell)^{p-1} \cdot \ell \cdot \beta \cdot (d\ell)^{(b-p)/2} \cdot (d\ell)^{(a-p)/2} = (d\ell)^{(b+a-2)/2} \cdot \ell\beta.$$

Similar to the proof of (4.3), we lower bound the denominator of the RHS of (4.6) by

$$p! \cdot (p-1)! \cdot (b-p+1)! \cdot (a-p)! \;\geqslant\; \left(\frac{b+a}{4e}\right)^{b+a}. \tag{4.7}$$

Summing our bound on (4.6) over $p \in [a]$, employing (4.7), and using that $a \leqslant (\sqrt{2})^{b+a}$,

$$\begin{aligned} |\{x \in \mathcal{A} : \{x,y\} \in E_{(b,a)}(\mathcal{A}),\ \ell_y > \ell_x\}| &\leqslant \sum_{p=1}^{a} \frac{(d\ell)^{(b+a-2)/2} \cdot \ell\beta}{p! \cdot (p-1)! \cdot (b-p+1)! \cdot (a-p)!} \\ &\leqslant \frac{(d\ell)^{(b+a-2)/2} (4\sqrt{2}e)^{(b+a)} \cdot \beta\ell}{(b+a)^{b+a}} \end{aligned}$$

The case when $b+a$ is odd. Notice that $\ell_y > \ell_x$ and $|x| \geqslant |y|$ implies $a \geqslant 1$, and in this case, $b \geqslant a+1 \geqslant p+1$. We upper bound the RHS of (4.6) by

$$(d\ell)^{p-1} \cdot \ell \cdot \beta^{b-p+1} \cdot |y|^{a-p} \leqslant (d\ell)^{p-1} \cdot \ell \cdot (d\ell)^{(b-p+1)/2} \cdot (d\ell)^{(a-p)/2} = (d\ell)^{(b+a-1)/2} \cdot \ell$$

Summing our bound on (4.6) over $p \in [a]$, employing (4.7), and using that $a \leqslant (\sqrt{2})^{b+a}$,

$$\begin{aligned} |\{x \in \mathcal{A} : \{x,y\} \in E_{(b,a)}(\mathcal{A}),\ \ell_y > \ell_x\}| &\leqslant \sum_{p=1}^{a} \frac{(d\ell)^{(b+a-1)/2} \cdot \ell}{p! \cdot (p-1)! \cdot (b-p+1)! \cdot (a-p)!} \\ &\leqslant \frac{(d\ell)^{(b+a-1)/2} (4\sqrt{2}e)^{(b+a)} \cdot \ell}{(b+a)^{b+a}} \end{aligned}$$

In both even and odd cases, summing over $y \in \mathcal{A}$ completes the proof. □

Proof of the even result, Theorem 4.1

Proof of Theorem 4.1. Recall that $\mathcal{U} = \{(b,a) \in \mathbb{Z}^2_{\geqslant 0} :\ b \geqslant a \text{ and } b+a \leqslant 2t\}$. Invoking (4.1) and using Lemma 4.10 and Lemma 4.11, we will upper bound each term in

$$\mathsf{e}_{\leqslant 2t}(\mathcal{A}) = \sum_{(b,a)\in\mathcal{U}} \mathsf{e}_{(b,a)}(\mathcal{A}).$$

For all $(b,a) \in \mathcal{U}$, we claim, using that $2 \leqslant (\sqrt{2})^{2t}$,

$$\mathsf{e}_{(b,a)}(\mathcal{A}) \leqslant 2\left(\frac{4\sqrt{2}e}{2t}\right)^{2t} (d\ell)^t \leqslant \left(\frac{8e}{2t}\right)^{2t} (d\ell)^t. \tag{4.8}$$

Assuming that (4.8) holds, and using that $|\mathcal{U}| \leqslant 2^{2t}$, we have

$$\sum_{(b,a)\in\mathcal{U}} \mathsf{e}_{(b,a)}(\mathcal{A}) \leqslant |\mathcal{U}| \cdot \left(\frac{8e}{2t}\right)^{2t} (d\ell)^t \leqslant \left(\frac{16e}{2t}\right)^{2t} (d\ell)^t$$

which implies the bound in the theorem statement.

To prove (4.8), we will use Proposition 4.8 and the fact that $2t \leqslant 2\log|\mathcal{A}|$. When $b+a$ is even, then combining Lemma 4.10 and Lemma 4.11 (using $\beta\ell \leqslant d\ell$), we have

$$\mathsf{e}_{(b,a)} \leqslant \left(\frac{8e}{b+a}\right)^{(b+a)} \cdot (d\cdot\ell)^{(b+a)/2}.$$

To verify (4.8), it suffices to show that the quantity above increases with $b+a$ (maximized over $\mathcal{U}$ at $b+a=2t$). Indeed, let $k = b+a \geqslant 2$. Then, we argue that

$$\left(\frac{8e}{k-1}\right)^{k-1} \cdot (d\cdot\ell)^{k/2-1/2} \leqslant \left(\frac{8e}{k}\right)^{k} \cdot (d\cdot\ell)^{k/2}.$$

After rearranging, we have

$$\frac{k}{8e}\left(\frac{k}{k-1}\right)^{k-1} \;\leqslant\; \frac{k}{8} \;\leqslant\; (d\ell)^{1/2}, \tag{4.9}$$

where the first inequality uses that $(\frac{k}{k-1})^{k-1} \leqslant e$, and the second inequality uses that $(k/8)^2 \leqslant t^2 \leqslant \log^2|\mathcal{A}| \leqslant d\ell$, which holds by Proposition 4.8(iii).

Similarly, when $b+a$ is odd, Lemma 4.10 and Lemma 4.11 (using $\ell \leqslant \log|\mathcal{A}|$) imply that

$$\mathsf{e}_{(b,a)} \leqslant \left(\frac{8e}{b+a}\right)^{b+a} \cdot (d\cdot\ell)^{(b+a-1)/2} \cdot \log|\mathcal{A}| \leqslant \left(\frac{8e}{2t-1}\right)^{2t-1} \cdot (d\cdot\ell)^{t-1} \cdot \log|\mathcal{A}|,$$

where the second inequality uses that the middle quantity is maximized at $b+a=2t-1$, which follows analogously to the proof of (4.9). To verify (4.8) in this case, we observe that

$$\left(\frac{8e}{2t-1}\right)^{2t-1} \cdot (d\cdot\ell)^{t-1} \cdot \log|\mathcal{A}| \leqslant \left(\frac{8e}{2t}\right)^{2t} (d\ell)^t,$$

using $(\frac{2t}{2t-1})^{2t-1} \leqslant e$ and $t\log|\mathcal{A}| \leqslant \log^2|\mathcal{A}| \leqslant d\ell$, which holds by Proposition 4.8(iii). □

4.4 The general case for odd r

In this section, we prove Theorem 4.2.

Proof of Theorem 4.2. Letting $\mathcal{U}' = \{(b,a) \in \mathbb{Z}^2_{\geqslant 0} : \ b \geqslant a \text{ and } b+a \leqslant 2t+1\}$, observe that

$$\mathsf{e}_{\leqslant 2t+1}(\mathcal{A}) = \sum_{(b,a)\in\mathcal{U}'} \mathsf{e}_{(b,a)}(\mathcal{A}).$$

We will upper bound each term in the above summation. For $(b,a) \in \mathcal{U}'$, we claim that

$$\mathsf{e}_{(b,a)} \leqslant 2\left(\frac{4\sqrt{2}e}{2t+1}\right)^{2t+1} (d\ell)^t \cdot \log|\mathcal{A}| \leqslant \left(\frac{8e}{2t+1}\right)^{2t+1} (d\ell)^t \cdot \log|\mathcal{A}|. \tag{4.10}$$

Assuming that (4.10) holds, and using that $|\mathcal{U}'| \leqslant 2^{2t+1}$, we have

$$\sum_{(b,a)\in\mathcal{U}} \mathsf{e}_{(b,a)} \leqslant |\mathcal{U}'| \cdot \left(\frac{8e}{2t+1}\right)^{2t+1} (d\ell)^t \cdot \log|\mathcal{A}| \ \leqslant \ \left(\frac{16e}{2t+1}\right)^{2t+1} (d\ell)^t \cdot \log|\mathcal{A}|,$$

which establishes the bound in the theorem statement.

We prove (4.10). When $b+a$ is even, then $b+a \leqslant 2t$ and (4.10) follows from (4.8). When $b+a$ is odd, then Lemma 4.10 and Lemma 4.11 (using $\ell \leqslant \log|\mathcal{A}|$) imply that

$$\mathsf{e}_{(b,a)} \leqslant \left(\frac{8e}{b+a}\right)^{b+a} \cdot (d\cdot\ell)^{(b+a-1)/2} \cdot \log|\mathcal{A}| \leqslant \left(\frac{8e}{2t+1}\right)^{2t+1} \cdot (d\cdot\ell)^t \cdot \log|\mathcal{A}|,$$

where we use that the middle quantity increases with $b+a$ (maximized over $\mathcal{U}'$ at $b+a = 2t+1$), analogous to the proof of (4.9). □

4.5 Tightness

Remark 2. For fixed $t \in \mathbb{N}$, Theorem 4.1 is sharp up to the value $8e$, as can be seen by taking $\mathcal{A} = [d]^{(\leqslant k)}$, i.e., a Hamming ball.

Remark 3. For fixed $t \in \mathbb{N}$, Theorem 4.2 is also sharp up to the value $16e$, as can be seen by taking

$$\mathcal{A} = \{x \in [d]^{(\leqslant k)} : \ |x \cap \{k+1,\dots,d\}| \leqslant 1\},$$

where $k = \Theta(\log d)$.

4.6 Conclusion

An obvious open question is to prove exact edge-isoperimetric inequalities for the graphs we consider. It would also be interesting to study other metric-induced graphs on $[k]^d$ for $k \geqslant 3$. Two possible generalizations of our results would be for the families of graphs connecting pairs in $[k]^d$ either with ℓ_1 or Hamming distance at most r. Bollobas and Leader [39] and Clements and Lindstrom [54] have solved the respective distance one cases.

Chapter 5

CLUSTERING IN EDIT DISTANCE FOR DNA STORAGE

Existing digital storage technologies cannot keep up with the modern data explosion. Therefore, researchers have turned to fundamentally different physical media for alternatives. Synthetic DNA has emerged as a promising option for the archival storage of digital data, with theoretical information density and durability of several orders of magnitude more than magnetic tapes [40, 68, 74, 152]. However, significant biochemical and computational improvements are necessary to scale DNA storage systems to read and write exabytes of arbitrary data within hours or even days.

Storing a file in DNA requires several encoding steps (recall Figure 1.1). First, the large digital file is compressed and partitioned into small, sequential blocks. Then, each block is randomized using an independent pseudo-random sequence. After adding address information to each randomized block, an error correcting code is applied to the global set of blocks. Finally, each block is encoded into the $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ alphabet. This process results in a large collection of hundred-character strings, which are then individually synthesized into real DNA and stored in a test tube until the file is ready to be retrieved.

To retrieve the data, the DNA is accessed using next-generation sequencing. This produces an unordered digital file, containing several noisy copies, called *reads*, of each originally synthesized short string, called a *reference*. Unlike in *de novo* genome assembly, these reads are either completely non-overlapping (different references), or completely overlapping (same reference). With current biochemical technologies, each reference strand contains on the order of hundreds of nucleotides. After sequencing, the goal is to recover the unknown references from the observed reads.

The first step, which is the focus of this chapter, is to cluster the reads. Each cluster will

correspond to a single reference, and the cluster will contain the noisy copies derived from this reference. The output of clustering is fed into a separate consensus-finding algorithm, which predicts the most likely reference to have produced each cluster of reads. To find the consensus, systems use algorithms that solve the *Trace Reconstruction* problem for strings with insertions, deletions, and substitutions [123]. Finally, the actual file is recovered from the approximate references using error correcting and addressing information.

As Figure 5.1 shows, datasets typically contain only a handful of reads for each reference. Moreover, the storage and retrieval process introduces significant noise, and each read differs from a reference by insertions, deletions, and/or substitutions. Therefore, the goal is to cluster the reads with respect to edit distance, and the challenge of clustering is to quickly achieve high precision and recall of many small clusters, in the presence of such errors.

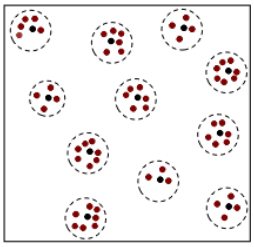


Figure 5.1: DNA storage datasets have many small clusters that are well-separated in edit distance.

Datasets arising from DNA storage have two striking properties. First, the number of clusters grows linearly with the input size, since each cluster typically consists of five to fifteen noisy copies of a reference. Second, the clusters are well-separated as a consequence of the randomization step during the encoding process (*i.e.*, the references are essentially random strings). Therefore, it may be assumed that the clusters have random centers, and in our high-dimensional space, this implies separation. We investigate approximate clustering algorithms for large collections of reads with these properties.

Suitable algorithms must satisfy several criteria. First, they must be distributed, to handle the billions of reads coming from modern sequencing machines. Second, their running time must scale favorably with the number of clusters. In DNA storage datasets, the size of the clusters is fixed and determined by the number of reads needed to recover the data. Thus, the number of clusters k grows linearly with the input size n (i.e., $k = \Omega(n)$). Any methods requiring $\Omega(k \cdot n) = \Omega(n^2)$ time or communication would be too slow for billion-scale datasets. Finally, algorithms must be robust to noise and outliers, and they must find clusters with

large diameters (linear in the dimensionality).

These criteria rule out many clustering methods. Algorithms for k-medians and related objectives are unsuitable because they have running time or communication scaling with $k \cdot n$ [51, 80, 90, 115]. Graph clustering methods, such as correlation clustering [12, 28, 49, 128], require a similarity graph. Constructing this graph is costly, and it is essentially equivalent to our clustering problem, since in DNA storage datasets, the similarity graph has connected components that are precisely the clusters of noisy reads. Linkage-based methods are inherently sequential, and iteratively merging the closest pair of clusters takes quadratic time. Agglomerative methods that are robust to outliers do not extend to versions that are distributed and efficient in terms of time, space, and communication [2, 27].

Turning to approximation algorithms, tools such as metric embeddings [117] and locality sensitive hashing (LSH) [84] trade a small loss in accuracy for a large reduction in running time. However, such tools are not well understood for edit distance [46, 47, 82, 103, 124], even though many methods have been proposed [44, 76, 108, 132, 154]. In particular, no published system has demonstrated the potential to handle billions of reads, and no efficient algorithms have experimental or theoretical results supporting that they would achieve high enough accuracy on DNA storage datasets. This is in stark contrast to set similarity and Hamming distance, which have many positive results [41, 94, 110, 142, 155].

Given these challenges, we ask two questions: (1) Can we design a distributed algorithm that converges in sub-quadratic time for DNA storage datasets? (2) Is it possible to adapt techniques from metric embeddings and LSH to cluster billions of strings in under an hour?

Our Contributions

We present a distributed algorithm that clusters billions of reads arising from DNA data storage. Our agglomerative algorithm utilizes a series of filters to avoid unnecessary distance computations. At a high level, our algorithm iteratively merges clusters based on random representatives. Using a hashing scheme for edit distance, we only compare a small subset of representatives. We also use a light-weight check based on a binary embedding to further filter

pairs. If a pair of representatives passes these two tests, edit distance determines whether the clusters are merged. Theoretically and experimentally, our algorithm satisfies four desirable properties.

- **Scalability:** Our algorithm scales well in both time and space, and in both shared-memory and shared-nothing environments. For example, on an input of n reads, each of P processors needs to hold only $O(n/P)$ reads in local memory.

- **Accuracy:** We measure accuracy as the fraction of clusters with a majority of found members and no false positives. Theoretically, we show that the separation of the underlying clusters implies that our algorithm converges quickly to a correct clustering. Experimentally, a small number of communication rounds achieves 98% accuracy on multiple real datasets, which suffices to retrieve the stored data.

- **Robustness:** When the underlying clusters have sufficient separation, our algorithm is optimally robust to adversarial outliers. We show that such separation is achieved, with high probability, when the noise rate is a small enough constant.

- **Performance:** Our algorithm outperforms the state-of-the-art clustering method for sequencing data, Starcode [157], achieving higher accuracy with a 1000x speedup. Our algorithm quickly recovers clusters with large diameter (e.g., 25), whereas known string similarity search methods perform poorly with distance threshold larger than four [93, 153]. Our algorithm is simple to implement in any distributed framework, and it clusters 5B reads with 99% accuracy in 46 minutes on 24 processors.

Overview of Our Algorithm

Our distributed clustering method iteratively merges clusters with similar representatives, alternating between local clustering and global reshuffling. At the core of our algorithm is a hash family that determines (i) which pairs of representatives to compare, and (ii) how to

repartition the data among the processors. On top of this simple framework, we use a cheap pre-check, based on the Hamming distance between binary signatures, to avoid many edit distance comparisons. Our algorithm achieves high accuracy by leveraging the fact that DNA storage datasets contain clusters that are well-separated in edit distance. In this section, we will define separated clusterings, explain the hash function and the binary signature, and describe the overall algorithm.

Hashing

Algorithms for string similarity search revolve around the simple fact that when two strings $x, y \in \Sigma^m$ have edit distance at most r, then they share a substring of length at least $m/(r+1)$. However, insertions and deletions imply that the matching substrings may appear in different locations. Exact algorithms build inverted indices to find matching substrings, and many optimizations have been proposed to exactly find all close pairs [92, 150, 157]. Since we need only an approximate solution, we design a hash family based on finding matching substrings quickly, without being exhaustive. Informally, for parameters w, ℓ, our hash picks a random "anchor" a of length w, and the hash value for x is the substring of length $w + \ell$ starting at the first occurrence of a in x.

Note that this hash family resembles MinHash [41, 43] with the natural mapping from strings to sets of substrings of length $w+\ell$. Our hash has the benefit of finding long substrings, while only having the overhead of finding shorter anchors (using a linear scan across the string). This reduces computation time in practice, while still leading to effective hashes.

It will be clear that our hash family does not have the full guarantees of LSH. In particular, for worst-case pairs of strings, the collision probability might be very large even for far apart strings. Fortunately, DNA data storage induces a natural average-case problem, based on random cluster centers. In this regime, we prove that our hash does indeed have favorable collision probabilities, and we are able to prove that our algorithm converges in a sub-quadratic number of comparisons. We first describe another optimization, based on a binary embedding, then describe our convergence theorem for the hash family.

Binary Embedding

The q-gram distance is an approximation for edit distance [146]. By now, it is a standard tool in bioinformatics and string similarity search [76, 79, 132, 154]. A q-gram (a.k.a. q-mer) is simply a substring of length q, and the q-gram distance measures the number of different q-grams between two strings. For a string $x \in \Sigma^m$, let the binary signature $\sigma_q(x) \in \{0,1\}^{|\Sigma|^q}$ be the indicator vector for the set q-grams in x. Then, the q-gram distance between x and y equals the Hamming distance $d_H(\sigma_q(x), \sigma_q(y))$. In our case, $|\Sigma| = 4$, and $\sigma_q(x) \in \{0,1\}^{4^q}$.

The utility of the q-gram distance is that the Hamming distance $d_H(\sigma_q(x), \sigma_q(y))$ approximates the edit distance $d_{\mathrm{E}}(x, y)$, yet it is much faster to check $d_H(\sigma_q(x), \sigma_q(y)) \leqslant \theta$ than to verify $d_{\mathrm{E}}(x, y) \leqslant r$. The only drawback of the q-gram distance is that it may not faithfully preserve the separation of clusters, in the worst case. This implies that the q-gram distance by itself is not sufficient for clustering. Therefore, we use binary signatures as a coarse filtering step, but reserve edit distance for ambiguous merging decisions. We provide theoretical bounds on the q-gram distance in Section 5.4.

Main Convergence Theorem

The running time of our algorithm depends primarily on the number of iterations and the total number of comparisons performed. The two types of comparisons are edit distance computations, which take time $O(rm)$ to check distance at most r, and q-gram distance computations, which take time linear in the signature length. To avoid unnecessary comparisons, we partition cluster representatives using our hash function and only compare reads with the same hash value. Therefore, we bound the total number of comparisons by bounding the total number of hash collisions.

For the convergence analysis, we prove that reads in the same cluster will collide with significant probability. Let p be the noise rate, so that the reads differ from the references by edit distance pm in expectation (Definition 13 and Definition 14 describe the precise model).

We prove that after roughly

$$O\left(\max\left\{n^{2p},\ \frac{n}{m^{1/(10p)}}\right\}\cdot\left(1+\frac{\log(s/\varepsilon)}{s}\right)\right)$$

iterations, the found clustering will be $(1-\varepsilon)$ accurate.

To understand this theorem, notice that each iteration involves $O(n)$ comparisons between strings. Therefore, the asymptotic number of comparisons (for fixed s, ε) scales with $\min\{n^{1+2p}, n^2/m^{1/(10p)}\}$. Then, observe that $n^{1+2p} \geqslant n^2/m^{1/(10p)}$ in the minimum whenever the reads are long enough (*i.e.,* m is large enough) and the noise rate is small enough (*i.e.,* p is small enough). This holds, for example, when $m \geqslant n^{10p}$. Thus, for a large range of n, m, p, and ε, our algorithm converges in time proportional to n^{1+2p}, which is sub-quadratic in n, the number of input reads. Since we expect the number of clusters k to be $k = \Omega(n)$, our algorithm outperforms any methods that require time $\Omega(kn) = \Omega(n^2)$ in this regime.

The running time analysis of our algorithm revolves around estimating both the collision probability of our hash function and the overall convergence time to identify the underlying clusters. The main overhead comes from unnecessarily comparing reads that belong to different clusters. Indeed, for pairs of reads inside the same cluster, the total number of comparisons is $O(n)$, since after a comparison, the reads will merge into the same cluster. For reads in different clusters, we show that they collide with probability that is exponentially small in the hash length (since they are nearly-random strings).

In Section 5.6, we experimentally validate our algorithm's running time, convergence, and correctness properties on real and synthetic data.

Outline. The rest of the chapter is organized as follows. We begin, in Section 5.1, by defining the problem statement, clustering accuracy, and our data model. Then, in Section 5.2, we describe our algorithm, hash function, and binary embedding. In Section 5.3, we bound the number of comparisons our algorithm takes for convergence. We analyze the binary embedding in Section 5.4 and outlier robustness in Section 5.5. In Section 5.6, we empirically evaluate our algorithm. We discuss related work in Section 5.7 and conclude in Section 5.8.

5.1 DNA Data Storage Model and Preliminaries

Recall from Chapter 2 that the edit distance $d_{\mathrm{E}}(x, y)$ of $x, y \in \Sigma^*$ equals the minimum number of character insertions, deletions, or substitutions to transform x to y. We fix $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. Also, recall that a clustering $\mathbf{C}$ of $S \subseteq \Sigma^*$ is any partition of S into nonempty subsets.

We introduce the following definition of accuracy, motivated by DNA storage data retrieval.

Definition 12 (Accuracy)**.** Let $\mathbf{C}, \widetilde{\mathbf{C}}$ be clusterings. For $1/2 < \gamma \leqslant 1$ the *accuracy* of $\widetilde{\mathbf{C}}$ with respect to $\mathbf{C}$ is

$$\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}) = \max_{\pi} \frac{1}{|\mathbf{C}|} \sum_{i=1}^{|\mathbf{C}|} \mathbf{1}_{\{\widetilde{C}_{\pi(i)} \subseteq C_i \quad \text{and} \quad |\widetilde{C}_{\pi(i)}| \geqslant \gamma |C_i|\}},$$

where the max is over all injective maps $\pi : \{1, 2, \ldots, |\widetilde{\mathbf{C}}|\} \to \{1, 2, \ldots, \max(|\mathbf{C}|, |\widetilde{\mathbf{C}}|)\}$.

The requirement $\gamma \in (1/2, 1]$ implies $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}) \in [0, 1]$. We think of $\mathbf{C}$ as the underlying clustering and $\widetilde{\mathbf{C}}$ as the output of an algorithm. The accuracy $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}})$ measures the number of clusters in $\widetilde{\mathbf{C}}$ that overlap with some cluster in $\mathbf{C}$ in at least a γ-fraction of elements while containing no false positives. This is a stricter notion than the standard classification error [27, 119]. Notice that our accuracy definition does not require that the clusterings be of the same set. We will compare clusterings of S and $S \cup \mathsf{O}$ for a set of outliers $\mathsf{O} \subseteq \Sigma^*$.

For DNA storage datasets, the underlying clusters have a natural interpretation. During data retrieval, several molecular copies of each original DNA strand (reference) are sent to a DNA sequencer. This produces a small number of noisy reads of each reference. Thus, the reads that correspond to the same reference form a cluster. This interpretation justifies the need for high accuracy: each underlying cluster represents one stored unit of information.

Data Model

To aid in the design and analysis of clustering algorithms for DNA data storage, we introduce the following generative model. First, choose many random centers (representing original references), then perturb each center by insertions, deletions, and substitutions to acquire the elements of the cluster (representing the noisy reads). We model the original references as

random strings because during the encoding process, the file has been randomized using a fixed pseudo-random sequence [123]. We make this model precise, starting with the perturbation.

Definition 13 (p-noisy copy)**.** For $p \in [0,1]$ and $z \in \Sigma^*$, define a p-noisy copy of z by the following process. For each character in z, independently, do one of the following four operations: (i) keep the character unchanged with probability $(1-p)$, (ii) delete it with probability $p/3$, (iii) with probability $p/3$, replace it with a character chosen uniformly at random from Σ, or (iv) with probability $p/3$, keep the character and insert an additional one after it, chosen uniformly at random from Σ.

Our model and analysis can be generalized for separate deletion, insertion, and substitution probabilities $p = p_D + p_I + p_S$, but we use balanced probabilities $p/3$ to simplify the exposition. We note that in real datasets, each of these individual probabilities is between $p/6$ and $2p/3$. Now, we define a noisy cluster. For simplicity, we assume uniform cluster sizes.

Definition 14 (Noisy cluster of size s)**.** We define the distribution $\mathcal{D}_{s,p,m}$ with cluster size s, noise rate $p \in [0,1]$, and dimension m. Sample a cluster $C \sim \mathcal{D}_{s,p,m}$ as follows: choose a center $z \in \Sigma^m$ uniformly at random; then, each of the s elements of C will be an independent p-noisy copy of z.

With our accuracy definition and data model in hand, we define the clustering problem.

Problem Statement: Fix p, m, s, n. Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be a set of $k = n/s$ independent clusters $C_i \sim \mathcal{D}_{s,p,m}$. Given an accuracy parameter $\gamma \in (1/2, 1]$ and an error tolerance $\varepsilon \in [0,1]$, on input set $S = \cup_{i=1}^{k} C_i$, the goal is to quickly find a clustering $\widetilde{\mathbf{C}}$ of S with $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}) \geqslant 1 - \varepsilon$.

Clusters are Well-Separated in Edit Distance

The most important consequence of our data model $\mathcal{D}_{s,p,m}$ is that the clusters will be well-separated in edit distance. Moreover, this reflects the actual separation of clusters in real datasets. Our clustering separability definition differs slightly from known notions [2, 3, 27]

in that we explicitly bound both the diameter of clusters and distance between clusters. We first define cluster diameter and distance, then use these to define separation.

Definition 15. A cluster C has *diameter* at most r if $d_{\mathrm{E}}(x, y) \leqslant r$ for all pairs $x, y \in C$. We define the *distance between two clusters* $C_1, C_2 \subseteq \Sigma^*$ as $d_{\mathrm{E}}(C_1, C_2) = \min_{x \in C_1, y \in C_2} d_{\mathrm{E}}(x, y)$.

Definition 16. A clustering $\{C_1, \ldots, C_k\}$ is (r_1, r_2)-*separated* if C_i has diameter at most r_1 for every $i \in \{1, 2, \ldots, k\}$, while any two different clusters C_i and C_j satisfy $d_{\mathrm{E}}(C_i, C_j) > r_2$.

DNA storage datasets will be separated with $r_2 \gg r_1$. Thus, recovering the clusters corresponds to finding pairs of strings with distance at most r_1. Whenever $r_2 \geqslant 2 \cdot r_1$, our algorithm will be robust to outliers (see Theorem 5.14). We will prove that clusters under $\mathcal{D}_{s,p,m}$ are separated with $r_2 \gg r_1$ as long as the noise p is a small enough constant.

Edit Distance of Random Strings

When recovering the data, DNA storage systems receive clusters that consist of p-noisy copies of the centers, which are random due to the encoding process. We need the clusters to be far apart for our algorithm to perform well. We will prove the following results in this section. First, we show that two reads inside of the same cluster will have edit distance at most $2pm + O(\sqrt{m \log n})$ with high probability, as they are p-noisy copies of the same center. Then we prove that random cluster centers will have edit distance $c_{\mathsf{ind}} \cdot m$ with high-probability, since two independent random strings have expected edit distance $c_{\mathsf{ind}} \cdot m$, for a constant $c_{\mathsf{ind}} > 0$. By the triangle inequality, reads in different clusters will have distance at least $(c_{\mathsf{ind}} - 4p)m - O(\sqrt{m \log n})$, and therefore, clusters will be separated in edit distance whenever m is large enough and $p \ll c_{\mathsf{ind}}/4$ is a small enough constant.

We start by showing that cluster centers are far apart. Let $\mathcal{U}_m$ be the uniform distribution over Σ^m. The cluster centers $c_1, \ldots, c_k$ are i.i.d. from $\mathcal{U}_m$. It is known that there exists a constant $c_{\mathsf{ind}} = c_{\mathsf{ind}}(|\Sigma|) > 0$ such that for $x, y \sim \mathcal{U}_m$,

$$\lim_{m \to \infty} \frac{d_{\mathrm{E}}(x, y)}{m} = c_{\mathrm{ind}}$$

almost surely (this follows from Kingman's ergodic theorem). Surprisingly, determining the precise value of c_{ind} is a challenging open problem, related to calculating the size of a ball under edit distance [29, 73]. When $|\Sigma| = 4$, simulations suggest $c_{\text{ind}} \approx 0.51$, and it is known [73] that $c_{\text{ind}} > 0.338$. In what follows, let c_{ind} be the largest constant satisfying

$$\mathop{\mathbb{E}}_{x,y\sim\mathcal{U}_m}\left[d_{\mathrm{E}}(x,y)\right] \geqslant c_{\mathsf{ind}} \cdot m,$$

for $\Sigma = \{\mathsf{A},\mathsf{C},\mathsf{G},\mathsf{T}\}$. The following lemma implies that random strings will have edit distance that is linear in m with high probability, since their expected distance is at least $c_{\mathsf{ind}} \cdot m$ and at most m.

Lemma 5.1. *For any $\lambda > 0$ we have*

$$\mathop{\mathbf{Pr}}_{x,y\sim\mathcal{U}_m}\left[\left|d_{\mathrm{E}}(x,y) - \mathop{\mathbb{E}}_{x,y\sim\mathcal{U}_m}\left[d_{\mathrm{E}}(x,y)\right]\right| \leqslant \lambda\sqrt{\frac{m}{2}}\right] \geqslant 1 - e^{-\lambda^2}.$$

Proof. Let $X = X_1X_2\cdots X_m$ and $Y = Y_1Y_2\cdots Y_m$ be random strings drawn from $\mathcal{U}_m$, and let Z_i be the pair (X_i, Y_i). Define $f(Z_1,\ldots,Z_m) = d_{\mathrm{E}}(X,Y)$. It is easy to see that

$$\left|\mathbb{E}[f \mid Z_1,\ldots,Z_{i-1},Z_i = (x_i,y_i)] - \mathbb{E}[f \mid Z_1,\ldots,Z_{i-1},Z_i = (x_i',y_i')]\right| \leqslant 1,$$

where x_i, y_i, x_i', y_i' are any four characters in Σ. The statement follows from Lemma 2.4. □

Cluster Diameter and Separation

We now show clusters have small diameter and are well-separated. We first analyze a cluster's radius by bounding the edit distance of p-noisy copies.

Lemma 5.2. *Let x be a p-noisy copy of $c \in \Sigma^m$. Then, for any $\lambda > 0$ we have*

$$\mathbf{Pr}\left[d_{\mathrm{E}}(x,c) \leqslant pm + \lambda\sqrt{3m}\right] \geqslant 1 - e^{-\lambda^2}.$$

Proof. Recall that x was generated from c by going from left to right in c, and at each character, introducing an edit (substitution, insertion, or deletion) with probability p, independently of the other edits. Notice that the number of edits from c to x is binomially distributed, with parameters m and p. Thus, $\mathbb{E}\left[d_{\mathrm{E}}(x,c)\right] = pm$, and then the Chernoff-Hoeffding bound (Lemma 2.2) proves the lemma. □

We tie together the above lemmas to prove that clusters are separated with high probability when m is large enough. In what follows, we think of p as a small constant $p \ll c_{\mathsf{ind}}$, to ensure separation. For high probability bounds, the number of input reads n just needs to satisfy $n < e^{\delta' c_{\mathsf{ind}}^2 m}$ for a small enough constant $\delta' > 0$.

Lemma 5.3. *Consider a clustering* $\mathbf{C} = \{C_1, \ldots, C_k\}$*, where each* $C_i \sim \mathcal{D}_{s,p,m}$*, and let* $n = sk$*. Then* $\mathbf{C}$ *is* (r_1, r_2)*-separated for*

$$r_1 = 2pm + 3\sqrt{m \ln n}$$
$$r_2 = c_{\mathsf{ind}} \cdot m - 4pm - 12\sqrt{m \ln n},$$

with probability at least $1 - 1/n^2$*.*

Proof. Let $\mathcal{E}_1$ be the event that for all $i \in [k]$ and for all $x \in C_i$, we have

$$d_{\mathrm{E}}(x, c_i) \leqslant \frac{r_1}{2},$$

where c_i is the center of C_i. We claim that $\mathcal{E}_1$ holds with probability at least $1 - \frac{1}{2n^2}$. This follows from setting $\lambda = 2\sqrt{\ln n}$ in Lemma 5.2 and a union bound over the $sk = n$ pairs (x, c_i) relevant to the event $\mathcal{E}_1$. When $\mathcal{E}_1$ holds, we have that $d_{\mathrm{E}}(x, x') \leqslant r_1$ for any pair $x, x' \in C_i$. Now, recall that

$$\mathbb{E}\left[d_{\mathrm{E}}(c_i, c_j)\right] \geqslant c_{\mathsf{ind}} \cdot m.$$

Let $\mathcal{E}_2$ be the event that for all pairs of cluster centers c_i and c_j for $i \neq j$ we have

$$d_{\mathrm{E}}(c_i, c_j) > r_2 + 2r_1.$$

When $\mathcal{E}_1 \cap \mathcal{E}_2$ holds, we have that for any two points $x \in C_i$ and $y \in C_j$ with $i \neq j$,

$$d_{\mathrm{E}}(x, y) \geqslant d_{\mathrm{E}}(c_i, c_j) - d_{\mathrm{E}}(x, c_i) - d_{\mathrm{E}}(y, c_j) > r_2 + 2r_1 - 2r_1 = r_2,$$

where the first inequality follows from the triangle inequality. By setting $\lambda = 2\sqrt{\ln n}$ in Lemma 5.1 and a union bound over $\binom{k}{2} \leqslant n^2$ pairs of cluster centers, we have that $\mathcal{E}_2$ holds with probability at least $1 - \frac{1}{2n^2}$. We conclude that $\mathcal{E}_1 \cap \mathcal{E}_2$ holds with probability at least $1 - \frac{1}{n^2}$ by a union bound, and that $\mathbf{C}$ is (r_1, r_2)-separated conditioned on $\mathcal{E}_1 \cap \mathcal{E}_2$. □

5.2 Our Algorithm

We define our distributed clustering algorithm. We start with our hash function, which is more likely to hash same-cluster strings to the same bucket than different-cluster strings. Then, we define binary signatures, which map strings to binary vectors while approximately preserving the pairwise distances. Finally, we provide a description of our overall algorithm.

Hashing for Edit Distance

The family of hash functions $\mathcal{H}_{w,\ell} = \{h_{\pi,\ell} : \Sigma^* \to \Sigma^{w+\ell}\}$ is parametrized by integers w, ℓ, and π, which is a permutation of Σ^w. For $x = x_1 x_2 \cdots x_m$, the value of $h_{\pi,\ell}(x)$ is defined as follows. Find the earliest, with respect to π, occurring w-gram a in x. Let i be the index of the first occurrence of a in x. Then, we define $h_{\pi,\ell}(x) = x_i \cdots x_m$, where we ensure the length of the hash is consistent by only looking for anchors that start at least $w + \ell$ characters from the end of the string. To sample $h_{\pi,\ell}$ from $\mathcal{H}_{w,\ell}$, simply pick a uniformly random permutation $\pi : \Sigma^w \to \Sigma^w$. Notice that computing $h_{\pi,\ell}(x)$ takes $O(m)$ time (when $w + \ell \ll m$ as it typically will be) since it requires only a single pass over the string.

Binary Signature Distance

A q-gram is simply a substring of length q, and the q-gram distance measures the number of different q-grams between two strings. For a string $x \in \Sigma^m$, let the binary signature $\sigma_q(x) \in \{0,1\}^{4^q}$ be the indicator vector for the set q-grams in x. Then, the q-gram distance between x and y equals the Hamming distance $d_H(\sigma_q(x), \sigma_q(y))$.

We show in Section 5.4 that random cluster centers will have q-gram distance at least $2m - O(1)$, with high probability, when $q \geqslant 2\log_4 m$. Additionally, for any two reads x, y, we show that $d_H(\sigma_q(x), \sigma_q(y)) \leqslant 2q \cdot d_{\mathrm{E}}(x, y)$, implying that if x and y are in the same cluster, then their q-gram distance will be at most $8qpm$ (since their edit distance is at most $4pm$ with high probability). Therefore, whenever $p \ll 1/q \approx 1/\log m$, signatures will already separate the clusters. For larger p, we use the pair of thresholds $\theta_{low} < \theta_{high}$ to mitigate false merges.

Algorithm 5.2.1: Clustering DNA Strands

function CLUSTER(S, r, q, w, ℓ, θ_{low}, θ_{high}, comm_steps, local_steps)
  $\widetilde{\mathbf{C}} = S$.
  For $i = 1, 2, \ldots,$ comm_steps:
    Sample $h_{\pi,\ell} \sim \mathcal{H}_{w,\ell}$ and partition clusters by applying $h_{\pi,\ell}$ to representatives.
    For $j = 1, 2, \ldots,$ local_steps:
      Sample $h_{\pi,\ell} \sim \mathcal{H}_{w,\ell}$.
      For $C \in \widetilde{\mathbf{C}}$, sample a representative $x_C \sim C$ and compute $h_{\pi,\ell}(x_C)$.
      For each pair x, y with $h_{\pi,\ell}(x) = h_{\pi,\ell}(y)$:
        If $(d_H(\sigma(x), \sigma(y)) \leqslant \theta_{low})$ or $(d_H(\sigma(x), \sigma(y)) \leqslant \theta_{high}$ and $d_{\mathrm{E}}(x, y) \leqslant r)$:
          Update $\widetilde{\mathbf{C}} = (\widetilde{\mathbf{C}} \setminus \{C_x, C_y\}) \cup \{C_x \cup C_y\}$.
  return $\widetilde{\mathbf{C}}$.
end function

Algorithm Description

We describe our distributed, agglomerative clustering algorithm (displayed in Algorithm 5.2.1). The algorithm ingests the input set $S \subset \Sigma^*$ in parallel, so each core begins with roughly the same number of reads. Signatures $\sigma_q(x)$ are pre-computed and stored for each $x \in S$. The clustering $\widetilde{\mathbf{C}}$ is initialized as singletons. It will be convenient to use the notation x_C for an element $x \in C$, and the notation C_x for the cluster that x belongs to. We abuse notation and use $\widetilde{\mathbf{C}}$ to denote the current global clustering. The algorithm alternates between global communication and local computation.

Communication: One representative x_C is sampled uniformly from each cluster C_x in the current clustering $\widetilde{\mathbf{C}}$, in parallel. Then, using shared randomness among all cores, a hash function $h_{\pi,\ell}$ is sampled from $\mathcal{H}_{w,\ell}$. Using this same hash function for each core, a hash value is computed for each representative x_C for cluster C in the current clustering $\widetilde{\mathbf{C}}$. The

communication round ends by redistributing the clusters randomly using these hash values. In particular, the value $h_{\pi,\ell}(x_c)$ determines which core receives C. The current clustering $\widetilde{\mathbf{C}}$ is thus repartitioned among cores.

Local Computation: The local computation proceeds independently on each core. One local round revolves around one hash function $h_{\pi,\ell} \sim \mathcal{H}_{w,\ell}$. Let $\widetilde{\mathbf{C}}_j$ be the set of clusters that have been distributed to the j^{th} core. During each local clustering step, one uniform representative x_C is sampled for each cluster $C \in \widetilde{\mathbf{C}}_j$. The representatives are bucketed based on $h_{\pi,\ell}(x_c)$. The local clustering uses three parameters, $r, \theta_{low}, \theta_{high}$, set ahead of time. For each pair y, z in a bucket, first the algorithm checks $d_H(\sigma_q(y), \sigma_q(z)) \leqslant \theta_{low}$. If so, the clusters C_y and C_z are merged. Otherwise, the algorithm checks if both $d_H(\sigma_q(y), \sigma_q(z)) \leqslant \theta_{high}$ and $d_{\mathrm{E}}(x, y) \leqslant r$, and merges the clusters C_y and C_z if these two conditions hold. Immediately after a merge, $\widetilde{\mathbf{C}}_j$ is updated, and C_x corresponds to the present cluster containing x. Distributing the clusters among cores during communication implies that no coordination is needed after merges. The local clustering repeats for *local_steps* rounds before moving to the next communication round.

Termination: The algorithm ends after *comm_steps* communication rounds. When the local computation finishes in the final clustering iteration, the algorithm outputs the current clustering $\widetilde{\mathbf{C}} = \bigcup_j \widetilde{\mathbf{C}}_j$ and terminates.

5.3 Algorithm Convergence and Hash Analysis

In this section, we bound the number of comparisons performed throughout the execution of the algorithm. We assume that the accuracy ε and the noise p are approximately known ahead of time (and $p \leqslant p_0$ for an absolute constant $p_0 > 0$). For the theoretical analysis, we prove a convergence theorem for a serial version of the algorithm, which we call CHM for *Clustering using Hashing and Merging*. It slightly differs from the one described in Section 5.2 in two ways (adding strings to multiple buckets, and ignoring binary signatures).

Modified Algorithm Description

CHM maintains a collection of clusters $\widetilde{\mathbf{C}}_t$, which we will call groups to differentiate them from the clusters $\mathbf{C}$ in the underlying clustering. CHM starts with a clustering $\widetilde{\mathbf{C}}_0$ of singletons; that is, on input $S \subseteq \Sigma^*$, every read belongs to its own group, so $\widetilde{\mathbf{C}}_0 = \{\{x\} : x \in S\}$. We will set hash parameters L_A and L_H for the anchor length $w = L_A$ and hash length $\ell = L_H$ as $L_A = \lceil \log_4 m \rceil$, $L = \min(\lfloor \frac{\log_4 m}{16p} \rfloor, \log_4(n/m))$. Then, CHM runs for $T = \frac{\beta_1 4^{L_A} \log(s/\varepsilon)}{m(1-2p)^L}$ iterations. At every iteration $t = 1, 2, \ldots, T$, CHM picks an anchor — a random string $a(t)$ of length w. Then, in each group g it picks a random read $x_g(t)$, which we call the center of group g for this iteration. For every center x, CHM finds all substrings of length $L = L_A + L_H$ with prefix $a(t)$. These substrings are hash values for g. Denote the set of hash values by $H_g(t)$. The algorithm adds g to *all buckets* of a hash table indexed by $h \in H_g(t)$. Then, CHM computes the edit distance between *every pair* of reads in each bucket and merges those groups whose centers x, x' have distance $d_{\mathrm{E}}(x, x') \leqslant r := 4pm$.

Adding g to multiple buckets is one difference in the modified algorithm. In practice, we only add g to the bucket for the earliest occurring anchor. Note that under $\mathcal{D}_{s,p,m}$, we expect reads to hash to $O(1)$ buckets, when L_A is large enough, with high probability.

Also, CHM does not use the binary signatures, since we are interested in the asymptotic number of comparisons. The signatures reduce the time it takes to compare two strings, but they may introduce errors due to the approximation in the edit distance. In Section 5.4, we prove cluster separation under signatures, and explain how signatures fit into this analysis. We empirically demonstrate the benefit of signatures on the running time in Section 5.6.

The following result analyzing the above algorithm is our main theoretical contribution. We assume that the underlying clusters are sampled independently from the distribution $\mathcal{D}_{s,p,m}$ defined earlier.

Theorem 5.4. *There exist absolute constants $p_0 > 0$, $\beta_1, \beta_2, \beta_3 > 0$, such that for every $p \leqslant p_0$, $\varepsilon > 0$, integer $s \geqslant 1$, and sufficiently large n, m ($n > m^2$) the following holds. Let $L_A = \lceil \log_4 m \rceil$, $L = \min(\lfloor \frac{\log_4 m}{16p} \rfloor, \log_4(n/m))$. Then, after $T = \frac{\beta_1 4^{L_A} \log(s/\varepsilon)}{m(1-2p)^L}$ steps, CHM*

recovers $(1-\varepsilon)$ *fraction of all clusters in expectation. The expected number of comparisons performed by* CHM *is upper bounded by*

$$O\Big(\frac{n^2 m(1+\log(s/\varepsilon)/s)}{4^L(1-2p)^L}\Big).$$

If $L = \lfloor \frac{\log m}{16p} \rfloor \leqslant \log_4(n/m)$, *we get the bound*

$$O\left(\frac{n^2(1+\log(s/\varepsilon)/s)}{m^{\beta_2/p}}\right).$$

If $L = \log_4(n/m) \leqslant \lfloor \frac{\log m}{16p} \rfloor$, *we get the bound*

$$O\Big(n^{1+2p} m^{2(1+p)}(1+\log(s/\varepsilon)/s)\Big).$$

Proof Overview

Lemma 5.3 implies that the underlying clusters $\mathbf{C} = \{C_1, \ldots, C_k\}$ will be $(r, 2r)$-separated for $r = 4pm$, and this holds with very high probability when $\mathbf{C}$ is drawn from $\mathcal{D}_{s,p,m}$ and $p \leqslant p_0$ is a small enough constant. For the remainder of this section, we will set $r = 4pm$. We thus assume that the separation holds, and only merge clusters if a pair of strings u, v has distance $d_{\mathrm{E}}(u,v) \leqslant r$. Hence, every intermediate clustering $\widetilde{\mathbf{C}}_t$ at time t will have the property that every group $g \in \widetilde{\mathbf{C}}_t$ is a subset of some C_i in the underlying clustering.

Let us introduce some notation: Denote the set of groups a cluster C_i is split into at the beginning of iteration t by $\mathcal{G}_i(t)$; the set of centers chosen for these groups by $\mathrm{center}_i(t)$; and, the number of these groups/centers by $s_i(t)$. Note that $\mathcal{G}_i(0)$ is a set of singletons $\mathcal{G}_i(0) = \{\{u\} : u \in C_i\}$; $\mathrm{center}_i(0) = C_i$; and $s_i(0) = |C_i| = s$. Let T be the total number of steps performed by CHM ; and $\mathcal{G}_i(T)$, $\mathrm{center}_i(T)$, and $s_i(T)$ be the set of groups, the set of centers, and the number of centers (respectively) in the cluster C_i when CHM terminates.

We need to show that the expected number of i such that $s_i(T) = 1$ is $(1-\varepsilon)n$ and then give a bound on the expected number of comparisons.

To analyze CHM, we will obtain lower and upper bounds on the probability that two distinct centers u and v end up in the same bucket during the t^{th} iteration. For any two

strings $u, v \in \Sigma^*$, let $S_{uv} \subseteq \Sigma^L$ be the set of all common substrings in u and v of length L, and let $A_{uv} \subseteq \Sigma^{L_A}$ be the set of length L_A prefixes of strings in S_{uv}. That is,

$$\begin{aligned} S_{uv} &= \{x \in \Sigma^L : x \text{ is a substring of } u, \text{ and } x \text{ is a substring of } v\}; \\ A_{uv} &= \{x_1 \cdots x_{L_A} \in \Sigma^{L_A} : x \in S_{uv}\}. \end{aligned}$$

Note that the sets A_{uv} and S_{uv} and the quantities $|A_{uv}|$ and $|S_{uv}|$ are random variables, due to the distribution of $\mathbf{C}$ sampled from $\mathcal{D}_{s,p,m}$.

Suppose that at iteration t, strands u and v are chosen as centers of distinct groups g_1 and g_2. Then, u and v are placed into the same bucket of the hash table if and only if the anchor $a(t)$ belongs to the set A_{uv}. This happens with probability $|A_{uv}|/4^{L_A}$, since CHM chooses the anchor $a(t) \in \Sigma^{L_A}$ uniformly at random and $|\Sigma| = 4$. The expected number of buckets that contain both u and v at step t is $|S_{uv}|/4^{L_A}$. Recall that in CHM a string may be placed not in one but multiple different buckets if the anchor string occurs in it more than once.

In what follows, we prove four lemmas. The first and second lemmas analyzes the expectation and concentration of the quantities $|S_{uv}|$ and $|A_{uv}|$. The third provides correctness for our algorithm, showing that a $(1-\varepsilon)$ fraction of clusters are completely recovered. The fourth lemma bounds the number of comparisons. In the final subsection, we put these lemmas together to prove the theorem.

We would like to show now that the random variables $|S_{uv}|$ and $|A_{uv}|$ are concentrated around the mean with high probability We will show in Lemma 5.6 that is true for $|S_{uv}|$ and $|A_{uv}|$ when $u, v \in C_i$. However, that is not true when $u \in C_i$, $v \in C_j$. To overcome this problem, we consider a sum of many random variables $|S_{uv}|$. We fix a cluster C_i and vertex $u \in C_i$ and bound the sum $\sum_{v \notin C_i} |S_{uv}|$, where the precise statement appears in Lemma 5.6.

We first estimate the expectation of $|S_{uv}|$ and $|A_{uv}|$.

Lemma 5.5. *Suppose $L_A = \lceil \log_4 6m \rceil$ and L is set as in the theorem statement. Then, over the randomness of $\mathbf{C}$ chosen from $\mathcal{D}_{s,p,m}$,*

(a) If $u, v \in C_i$ for some i, then

$$\mathbb{E}|S_{uv}| \geqslant (m - L + 1)(1 - 2p)^L,$$

and

$$\mathbb{E}|A_{uv}| \geqslant \frac{1}{6}(m - L + 1)(1 - 2p)^L.$$

(b) If $u \in C_i$, $v \in C_j$ for $i \neq j$, then $\mathbb{E}|S_{uv}| \leqslant m^2 4^{-L}$.

Proof. (a) If u and v belong to the same true cluster C_i, then they are p-noisy copies of the same reference $c \in \Sigma^m$, and c contains $(m - L + 1)$ substrings of length L. Each of these substrings is present in both u and v with probability $(1-p)^L \times (1-p)^L = (1-p)^{2L}$. Hence, the expected number of common substrings satisfies $\mathbb{E}|S_{uv}| \geqslant (1-p)^{2L}(m - L + 1)$.

We now estimate $\mathbb{E}|A_{uv}|$. The reference c contains $(m - L + 1)$ prefixes of length L_A of substrings of length L. Let Z_a be the number of occurrences of a fixed "anchor" a (i.e., a substring of length L_A) in c that starts at least L characters before the end of c. The expectation of Z_a equals $\mathbb{E}[Z_a] = (m - L + 1)4^{-L_A}$, we denote this expectation by μ. An easy computation shows that $\mathbf{Pr}[Z_a \geqslant 1] \geqslant \mu/6$. Thus, the expected number of distinct anchors in c is at least $\frac{\mu}{6} \times 4^{L_A} = (m - L + 1)/6$. Each of these anchors gets copied to u and v without errors together with the next $L_H = L - L_A$ characters in w with probability at least $(1-2p)^L$. Hence, $\mathbb{E}|A_{uv}| \geqslant (m - L + 1)(1 - 2p)^{L_A}/6$.

(b) If $i \neq j$, then each u and v contains $m - L + 1 \leqslant m$ substrings of length L. The number of pairs of strings of length L – one from u and one from v – is at most m^2. The probability that two particular substrings are the same is 4^{-L} (as strings u and v are independent random strings in the alphabet Σ). Hence the expected number of common substrings of length L is at most $m^2 4^{-L}$. □

Lemma 5.6. *There exists an absolute constant β such that the following inequalities hold if $L < m/2$ and $L_A \geqslant \log_4 6m$.*

(a) If $u, v \in C_i$ for some i, then

$$\mathbf{Pr}\left[|S_{uv}| \geqslant \frac{1}{10}\, m(1-2p)^L\right] \geqslant 1 - \exp\left(-\frac{\beta m(1-4p)^L}{L^2}\right);$$

and

$$\mathbf{Pr}\left[|A_{uv}| \geqslant \frac{1}{60}\, m(1-2p)^L\right] \geqslant 1 - \exp\left(-\frac{\beta m(1-4p)^L}{L^2}\right);$$

(b) If $u \in C_i$, $v \in C_j$ for some $i \neq j$, then

$$\mathbf{Pr}\left[\sum_{v \notin C_i} |S_{uv}| \leqslant \frac{2nm^2}{4^L}\right] \geqslant 1 - m^2 \exp\Big(-\frac{\beta n}{4^L}\Big).$$

Proof. The proof of Lemma 5.6 is similar to the proof of Lemma 5.5. We estimate the expectation as in Lemma 5.5 and then apply a concentration inequality. The only minor complication is that not all random variables in the sum we get are independent. Thus, in part (a), we use the Bounded Difference inequality (Lemma 2.4); and in part (b), we use the Chernoff-Hoeffding bound (Lemma 2.2), but we break the sum into m^2 sums of independent random variables. □

5.3.1 Single Cluster Dynamics

We analyze the evolution of a single cluster C_i from the underlying true clustering throughout the execution of CHM. To obtain the desired bounds, we use that the probability that any two groups in C_i are merged at step t is at least $\alpha = m(1-2p)^L/(60 \cdot 4^{L_A})$. Note, however, that the events $\{g_1$ and g_2 are merged at step $t\}$ and $\{g_2$ and g_3 are merged at step $t\}$ are not independent.

Lemma 5.7. *The following bounds hold for $\alpha = m(1-2p)^L/(60 \cdot 4^{L_A})$.*

(a) $\mathbf{Pr}[s_i(T) > 1] \leqslant (s-1)(1-\alpha)^T$.

(b) $\mathbb{E}\left[\sum_{t=0}^{T-1} s_i(t)\right] \leqslant T + \frac{s-1}{\alpha}$.

Proof. (a) Choose a designated vertex u in C_i. For every v the probability that u and v are in two distinct groups after t steps of CHM is at most $(1-\alpha)^t$ since at every step the probability that the group containing u and group containing v are merged is at least α. Thus, the expected number of vertices that are not in the group containing u is at most $(s-1)(1-\alpha)^t$. This is also a bound on the number of groups not containing u. Hence, $\mathbb{E}[s_i(t)] \leqslant 1+(s-1)(1-\alpha)^t$, and, consequently, $\mathbf{Pr}[s_i(t) > 1] \leqslant \mathbb{E}[s_i(t)-1] \leqslant (s-1)(1-\alpha)^t$.

(b) Using the bound $\mathbb{E}[s_i(t)] \leqslant 1+(s-1)(1-\alpha)^t$, we get

$$\sum_{t=0}^{T-1}(1+(s-1)(1-\alpha)^t) \leqslant T + (s-1)/\alpha.$$

□

5.3.2 Number of Comparisons

Lemma 5.8. *The expected number of different-cluster (external) comparisons performed by* CHM *in* $T = \frac{\beta_1 4^{L_A}\log(s/\varepsilon)}{m(1-2p)^L}$ *steps is upper bounded by*

$$O\Big(\frac{n^2 m(1+\log(s/\varepsilon)/s)}{4^L(1-2p)^L}\Big).$$

Proof. Fix a cluster C_i. We estimate the expected number of external comparisons between the strands in C_i and the strands outside of C_i. Consider a step t of CHM . If u is chosen as a center of a group in C_i, then the expected number of external comparisons between u and strands v outside of C_i is upper bounded by $4^{-L_A}\sum_{v\notin C_i}|S_{uv}|$. As we discussed earlier $4^{-L_A}|S_{uv}|$ is the expected number of buckets in which both u and v are placed at iteration t if u and v are centers. Here, we use a conservative estimate: instead of counting only strands v that are centers of groups outside of C_i we count all v's outside of C_i. Summing over all centers $u \in \text{center}_i(t)$, we upper bound the expected number of comparisons between the centers in C_i and the other centers:

$$\begin{aligned} 4^{-L_A}\mathbb{E}\left[\sum_{u\in\text{center}_i(t)}\sum_{v\notin C_i}|S_{uv}|\right] &\leqslant 4^{-L_A}\mathbb{E}\Big[|\text{center}_i(t)|\max_{u\in C_i}\sum_{v\notin C_i}|S_{uv}|\Big] &(5.1)\\ &= 4^{-L_A}\mathbb{E}\Big[s_i(t)\max_{u\in C_i}\sum_{v\notin C_i}|S_{uv}|\Big]. &(5.2)\end{aligned}$$

Let $\mathcal{E}_u$ be the event that $\sum_{v \notin C_i} |S_{uv}| \leqslant 2nm^2/4^L$, and let $\mathcal{E}_{C_i} = \bigcap_{u \in C_i} \mathcal{E}_u$ be the event that

$$\max_{u \in C_i} \sum_{v \notin C_i} |S_{uv}| \leqslant \frac{2nm^2}{4^L}.$$

Let I_i be the indicator of the event $\mathcal{E}_{C_i}$. By Lemma 5.6, $\mathbf{Pr}(\neg \mathcal{E}_u) \leqslant m^2 \exp(-\beta n/4^L)$. Using the union bound, we get $\mathbb{E}[1 - I_i] = \mathbf{Pr}(\neg \mathcal{E}_{C_i}) \leqslant sm^2 \exp(-\beta n/4^L) \leqslant 4^{-L}/s$, where the last inequality follows from the bound $L \leqslant \log_4(n/m)$ for sufficiently large m. We now consider two cases: $I_i = 1$, then we bound the expected number of comparisons using (5.2); and $I_i = 0$, then we bound the number of comparisons by sn (this is the maximum possible number of comparisons). We get the following bound on the number of comparisons between centers in C_i and all other centers at iteration t:

$$\begin{aligned} \frac{1}{4^{L_A}} \mathbb{E}\left[I_i \cdot s_i(t) \cdot \sum_{v \notin C_i} |S_{uv}| \right] + \mathbb{E}\left[(1 - I_i) sn\right] &\leqslant \frac{1}{4^{L_A}} \mathbb{E}\left[I_i \cdot s_i(t) \cdot \frac{2nm^2}{4^L} \right] + \mathbb{E}\left[(1 - I_i) sn\right] \\ &\leqslant \frac{2nm^2}{4^{L+L_A}} \mathbb{E}[s_i(t)] + \frac{n}{4^L}. \end{aligned}$$

We now sum up this bound over all steps $t = 0, \ldots, T-1$ of CHM. By Lemma 5.7,

$$\mathbb{E}\left[\sum_{t=0}^{T-1} |s_i(t)| \right] \leqslant T + s/\alpha,$$

where we recall that $\alpha = m(1-2p)^L/(60 \cdot 4^{L_A})$. Hence, the expected number of external comparisons for a cluster C_i is upper bounded by

$$\frac{2nm^2}{4^{L+L_A}} \cdot \left(T + \frac{60s \cdot 4^{L_A}}{m(1-2p)^L} \right) + \frac{nT}{4^L}, \quad \text{which is } \; O\left(\frac{Tnm^2}{4^{L+L_A}} + \frac{snm}{4^L(1-2p)^L} \right).$$

Summing up this bound over all k clusters $\{C_i\}$, and plugging in the value for T, we get the bound on the number of comparisons of

$$O\left(\frac{Tn^2m^2}{s4^{L+L_A}} + \frac{n^2m}{4^L(1-2p)^L} \right) = O\left(\frac{n^2m(1 + \log(s/\varepsilon)/s)}{4^L(1-2p)^L} \right).$$

□

5.3.3 Putting it all together

We now complete the proof of Theorem 5.4.

Proof of Theorem 5.4. We first argue that the algorithm CHM is correct, in the sense that it recovers at least an ε fraction of all clusters. Then, we bound the total number of comparisons made by the algorithm.

Correctness. We first show that the expected number of clusters completely recovered after T steps of CHM is $(1-\varepsilon)$. Consider a cluster C_i and $u, v \in C_i$. Suppose that u and v are centers of two different groups in C_i. If CHM places u and v in the same bucket of the hash table, which happens with probability $|A_{uv}|/4^{L_A}$, then the groups corresponding u and v are merged, since all clusters are separated for our setting of the distance threshold $r = 4pm$ (recall this holds with high probability when $p < p_0$ is small enough). We say that C_i is *good* if for all $u, v \in C_i$ we have $|A_{uv}| \geqslant \frac{m}{60}(1-2p)^L$, and otherwise, we say that C_i is *bad*.

Lemma 5.6 asserts that the fraction of bad clusters is at most $\exp(-\beta m(1-4p)^L/L^2)$. Observe that the hypothesis of the theorem sets $L \leqslant \frac{\log m}{16p}$, which implies

$$\begin{aligned}(1-4p)^L = \exp(-L\log(1/(1-4p))) &\geqslant \exp\left(-\frac{\log m}{16p}\cdot\log(1/(1-4p))\right)\\ &\geqslant \exp\left(-\frac{1}{2}\log m\right)\\ &= m^{-1/2},\end{aligned}$$

where we used that for $p \leqslant 1/8$, we have $\frac{1}{16p}\cdot\log(1/(1-4p)) \leqslant 1/2$. Hence, the fraction of bad clusters is upper bounded by $\exp(-\beta m^{1/2}/L^2)$, which is much less than ε for a sufficiently large m. Thus, we can ignore bad clusters without reducing accuracy significantly.

Consider a good cluster C_i. At every iteration t, for any two centers $u, v \in \text{center}_i(t)$, the probability that CHM puts u and v into the same bucket is $|A_{uv}|4^{-L_A}$, which at least

$$\alpha = \frac{m(1-2p)^L}{60\cdot 4^{L_A}}.$$

Therefore, using Lemma 5.7, we obtain the following bound on the expected number of non-recovered clusters:

$$s(1-\alpha)^T = s\Big(1 - \frac{m(1-2p)^L}{60 \cdot 4^{L_A}}\Big)^T \leqslant \varepsilon/2.$$

The last inequality holds, since

$$T = \frac{\beta_1 4^{L_A} \log(s/\varepsilon)}{m(1-2p)^L},$$

for some absolute constant β_1 that is large enough.

Number of Comparisons. We now upper bound the total number of comparisons. We divide all strand comparisons into two types: internal and external. Recall that we say that a comparison of u and v is internal if u and u belong to the same true cluster C_i; we say that it is external, if u and v belong to two distinct clusters C_i and C_j. The total number of internal comparisons for C_i is bounded by $|C_i| - 1 = s - 1$, since after $s-1$ comparisons, all elements of C_i must be merged into one group. Thus, the total number of internal comparisons is $k(s-1) < n$. Lemma 5.8 gives a bound on the number of external comparisons of

$$O\left(\frac{n^2 m(1 + \log(s/\varepsilon)/s)}{4^L(1-2p)^L}\right).$$

This bound together with the bound on the number of internal comparisons (at most n) gives us the desired result and completes the proof of the theorem. □

5.4 Clusters are Well-Separated Under Binary Signatures

Recall that for a string $x \in \Sigma^m$ with $|\Sigma| = 4$, the signature $\sigma_q(x) \in \{0,1\}^{4^q}$ is the indicator vector for the set q-grams in x, and, the q-gram distance between x and y equals the Hamming distance $d_H(\sigma_q(x), \sigma_q(y))$. In this section, we analyze how well the q-gram distance approximates the edit distance, both for clusters with random centers and for pairs of strings within the same cluster. The goal will be to show that when the noise rate p is small enough, then the clusters remain separated in q-gram distance.

Binary signatures are an important innovation of our empirical algorithm presented in Section 5.2. We did not use them as part of our theoretical algorithm, CHM, which is analyzed in Theorem 5.4. The reason for this discrepancy is two-fold. First, although the signatures reduce the time to compare two strings, they do not reduce the number of comparisons, which is the focus of theorem. Second, we will see shortly that the signatures only work when the noise rate p satisfies $p \ll O(1/\log m)$, due to the approximation guarantees (otherwise, clusters may not be separated). We leave open the interesting question of developing an embedding of edit distance into Hamming distance that works for constant noise rate for random strings (we discuss in Section 5.7.1 some limitations for worst-case strings).

5.4.1 Signature Distance of Random Strings

We start by lower bounding the expected distance for random strings.

Lemma 5.9. *Let q be a natural number satisfying $q \geqslant \lceil \log_4 3m \rceil$ and $q \leqslant m$. Then,*

$$\mathop{\mathbb{E}}_{x,y \sim \mathcal{U}_m} [d_H(\sigma_q(x), \sigma_q(y))] \geqslant 2(m-q+1) - \frac{3(m-q+1)^2}{4^q}.$$

In particular, for $q = \lceil \log_4 3m \rceil$ and m large enough, $d_H(\sigma_q(c_i), \sigma_q(c_i)) \geqslant m - O(\log m)$ in expectation for distinct cluster centers $c_i, c_j \sim \mathcal{U}_m$.

Proof. The number of q-grams in a string $x \in \Sigma^m$ is exactly $m-q+1$. Let X, Y denote the multi-set of q-grams in x and y, respectively, where $|X| = |Y| = (m-q+1)$. Observe that

$$d_H(\sigma_q(x), \sigma_q(y)) \geqslant |\sigma_q(x)| + |\sigma_q(y)| - 2|X \cap Y|.$$

By linearity of expectation, it suffices to prove that the following two bounds hold:

$$\mathbb{E}\Big[|X \cap Y|\Big] \leqslant \frac{|X||Y|}{4^q} = \frac{(m-q+1)^2}{4^q}. \tag{5.3}$$

$$\mathbb{E}\Big[|\sigma_q(x)| + |\sigma_q(y)|\Big] \geqslant 2(m-q+1) - \frac{2(m-q+1)^2}{4^q}. \tag{5.4}$$

We start with (5.3). Each q-gram in the multi-set X appears in Y with probability at most $|Y|/4^q$, since there are $|Y|$ elements of Y and each of the 4^q possible q-grams is equally likely. Summing over the $|X|$ q-grams in X gives the bound on $\mathbb{E}\Big[|X \cap Y|\Big]$.

We move on to (5.4). We wish to show $\mathbb{E}\,|\sigma_q(x)| \geqslant |X| - \frac{|X|^2}{4^q}$. We claim that each q-gram in X appears at most $1 + |X|/4^q$ times in X, in expectation. Decompose the string x as $x = x_1 x_2 \cdots x_m$, and let $x[i,j] = x_i \cdots x_j$ for $i \leqslant j$ denote the substring of x for this range. By translation and symmetry, it suffices to show that for any $t \in \{1, \ldots, m-q\}$ we have

$$\mathbf{Pr}\left[x[1,q] = x[t+1, t+q]\right] \leqslant \frac{1}{4^q}. \tag{5.5}$$

When $t \geqslant q$, these substrings do not overlap, and the probability of equality is exactly 4^{-q}. Assume $t < q$, and observe that $x[1,q] = x[t+1, t+q]$ is only possible if t divides q, due to the implied periodicity. That is, when t does not divide q, the probability of equality is zero. When t divides q, notice that $x[1,q] = x[t+1, t+q]$ implies that we must have

$$x[it+1, (i+1)t] = x[jt+1, (j+1)t]$$

for all $i, j \in \{0, 1, \ldots, q/t\}$. This happens with probability $(4^{-t})^{q/t} = 4^{-q}$, establishing (5.5). We conclude that (5.4) holds by summing over all possible q-grams in x and all values $t \in \{1, \ldots, m-q\}$.

Finally, for distinct random centers $c_i, c_j \sim \mathcal{U}_m$, the lower bound on the expectation of $d_H(\sigma_q(c_i), \sigma_q(c_i))$ follows by plugging in $q = \lceil \log_4 3m \rceil$. □

Remark 4. We make two brief comments about Lemma 5.9. First, an obvious question is whether it is possible to do better by counting the q-grams instead of defining the signatures in terms of indicator vectors. When $q = \lceil 2 \log_4 m \rceil$, we expect each q-gram to appear at most once, so in this case, the vast majority of counts will be binary anyways. The second comment is that the proof heavily relies on the fact that the strings are random. Indeed, there are worst-case examples where the q-gram distance can be zero, even when the edit distance between x and y is very large. Let α, β be arbitrary strings on disjoint alphabets with length at least q. Then, consider the concatenated strings $x = \alpha\beta\alpha$ and $y = \beta\alpha\beta$, and notice that $\sigma_q(x) = \sigma_q(y)$, yet $d_{\mathrm{E}}(x, y) = \Omega(|x|)$.

Lemma 5.10. *Let $\mathcal{U}_m$ be the uniform distribution over Σ^m. Let q be a natural number satisfying $q \geqslant \lceil 2\log_4 m \rceil$ and $q \leqslant m/2$. Then, for any $\lambda > 0$, we have*

$$\Pr_{x,y\sim\mathcal{U}_m}\left[\left|d_H(\sigma_q(x),\sigma_q(y)) - \mathbb{E}[d_H(\sigma_q(x),\sigma_q(y))]\right| \leqslant \lambda q\sqrt{m}\right] \geqslant 1 - e^{-2\lambda^2}.$$

Proof. Let $X = X_1X_2\cdots X_m$ and $Y = Y_1Y_2\cdots Y_m$ be random strings drawn from $\mathcal{U}_m$, and let Z_i be the pair (X_i, Y_i). Define $f(Z_1,\ldots,Z_m) = d_H(\sigma_q(X),\sigma_q(Y))$. Since σ_q is defined in terms of q-grams, it is easy to see that

$$|\mathbb{E}[f \mid Z_1,\ldots,Z_{i-1},Z_i=(x_i,y_i)] - \mathbb{E}[f \mid Z_1,\ldots,Z_{i-1},Z_i=(x_i',y_i')]| \leqslant q,$$

where x_i, y_i, x_i', y_i' are any four characters in Σ^m. Therefore, the statement follows from Lemma 2.4. □

5.4.2 Cluster Analysis for Signatures

We are interested in bounding the maximum difference between the edit distance of the original strands and the Hamming distance of the signatures.

Lemma 5.11. *Let q be a natural number satisfying $0 \leqslant q \leqslant m$. For any $x, y \in \Sigma^*$,*

$$d_H(\sigma_q(x),\sigma_q(y)) \leqslant \min\{2q\cdot d_{\mathrm{E}}(x,y),\ |x|+|y|-2q+2\}$$

Proof. The upper bound of $|x|+|y|-2q+2$ holds simply because $\sigma_q(x)$ contains at most $|x|-q+1$ non-zero coordinates (likewise for y). To show $d_H(\sigma_q(x),\sigma_q(y)) \leqslant 2q\cdot d_{\mathrm{E}}(x,y)$, we will analyze the case of a single edit, and prove $d_H(\sigma_q(x),\sigma_q(z)) \leqslant 2q$ when $d_{\mathrm{E}}(x,z)=1$. Then, we observe that the triangle inequality implies the upper bound for all edit distances. Let z be any string in Σ^* with $d_{\mathrm{E}}(x,z)=1$. Let X and Z be the set of q-grams in x and z, respectively. Notice that $d_H(\sigma_q(x),\sigma_q(z))$ equals the size of the symmetric difference $|X\triangle Z|$. We claim $|X\setminus Z| \leqslant q$, since the single edit between x and z can cause at most q elements of X to be absent in Z. Similarly, we claim $|Z\setminus X| \leqslant q$, since the single edit between x and z can cause at most q elements to be in Z that are not present in X. Thus,

$$d_H(\sigma_q(x),\sigma_q(z)) = |X\triangle Z| \leqslant |X\setminus Z| + |Z\setminus X| \leqslant 2q.$$

□

Lemma 5.12. *Let x be a p-noisy copy of $c \in \Sigma^m$. Then, for any $\lambda > 0$ we have*

$$\mathbf{Pr}\left[d_H(\sigma_q(x), \sigma_q(c)) \leqslant 2q\left(pm + \lambda\sqrt{3m}\right)\right] \geqslant 1 - e^{-\lambda^2}.$$

Proof. This follows directly from Lemma 5.2 and Lemma 5.11. □

Lemma 5.13. *Let $q = \lceil 2\log_4 m \rceil$. Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be a random clustering with $C_i \sim \mathcal{D}_{s,p,m}$. Then, with probability $1 - 1/n^2$, we have that*

1. *for any $i = 1, 2, \ldots, k$ and any $x, y \in C_i$,*

$$d_H(\sigma_q(x), \sigma_q(y)) \leqslant 4q\left(pm + 3\sqrt{m \ln n}\right) =: r'_1.$$

2. *for $x \in C_i$ and $y \in C_j$ with $i \neq j$,*

$$d_H(\sigma_q(x), \sigma_q(y)) \geqslant 2m - 2q\left(4pm + 7\sqrt{m \ln n}\right) =: r'_2.$$

Proof. Let $\mathcal{E}'_1$ be the event that for all $i \in [k]$ and for all $x \in C_i$, we have

$$d_H(\sigma_q(x), \sigma_q(c_i)) \leqslant \frac{r'_1}{2},$$

where c_i is the center of C_i. We claim that $\mathcal{E}'_1$ holds with probability at least $1 - \frac{1}{2n^2}$. This follows from setting $\lambda = 2\sqrt{\ln n}$ in Lemma 5.12 and a union bound over the $sk = n$ pairs (x, c_i) relevant to the event $\mathcal{E}'_1$. Notice that when $\mathcal{E}'_1$ holds, we have that $d_H(\sigma_q(x), \sigma_q(x')) \leqslant r'_1$ for any pair $x, x' \in C_i$.

Now, let $\mathcal{E}'_2$ be the event that for all pairs of cluster centers c_i and c_j for $i \neq j$ we have

$$d_H(\sigma_q(c_i), \sigma_q(c_j)) > r'_2 + 2r'_1.$$

When $\mathcal{E}'_1 \cap \mathcal{E}'_2$ holds, we have that for any two points $x \in C_i$ and $y \in C_j$ with $i \neq j$,

$$d_H(\sigma_q(x), \sigma_q(y))) \geqslant d_H(\sigma_q(c_i), \sigma_q(c_j)) - d_H(\sigma_q(x), \sigma_q(c_i)) - d_H(\sigma_q(y), \sigma_q(c_j)) > r'_2,$$

where the first inequality follows from the triangle inequality. By applying Lemma 5.9 and setting $\lambda = 2\sqrt{\ln n}$ in Lemma 5.10 and a union bound over $\binom{k}{2} \leqslant n^2$ pairs of cluster centers, we have that $\mathcal{E}_2'$ holds with probability at least $1 - \frac{1}{2n^2}$.

We conclude that $\mathcal{E}_1' \cap \mathcal{E}_2'$ holds with probability at least $1 - \frac{1}{n^2}$ by a union bound, and that $\mathbf{C}$ satisfies the claimed bounds conditioned on $\mathcal{E}_1' \cap \mathcal{E}_2'$. □

Lemma 5.13 proves that the clusters are separated according to the binary signatures with high probability whenever $p \ll 1/\log m$. This is a stricter requirement than we needed for the edit distance separation, since that tolerated error rate $p = O(1)$. Constructing an efficient binary embedding for $p = \Omega(1)$ would indeed be interesting.

5.4.3 Practical Considerations

In Section 5.6, we mention an optimization for the binary signatures, based on blocking, which empirically improves the approximation quality, while reducing computational overhead. The basic idea is to use a smaller q by breaking the string in a small number of blocks. We will generate the signature for each disjoint block separately, then concatenate the signatures for the separate blocks to generate a single binary vector. As a result of our theoretical bounds above, we use the fact that q simply needs to be logarithmic in the length of the block.

5.5 Outlier Robustness

Our final theoretical result involves bounding the number of incorrect merges caused by potential outliers in the dataset. In real datasets, we expect some number of highly-noisy reads, due to experimental error. Fortunately, such outliers lead to only a minor loss in accuracy for our algorithm, when the clusters are separated.

Theorem 5.14. *Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be an $(r, 2r)$-separated clustering of S. Let O be any set of size $\varepsilon' k$. Fixing the randomness and parameters in CHM with distance threshold r, let $\widetilde{\mathbf{C}}$ be the output on S and $\tilde{\mathbf{C}}'$ be the output on $S \cup \mathsf{O}$. Then, $\mathcal{A}_\gamma(\mathbf{C}, \tilde{\mathbf{C}}') \geqslant \mathcal{A}_\gamma(\mathbf{C}, \tilde{\mathbf{C}}) - \varepsilon'$.*

Proof. For this proof, we analyze a slightly simplified version of our algorithm that does not use signatures for merging. A similar result holds for the original algorithm, but it is more complicated to state (since it depends on $\mathcal{D}_{s,p,m}$), and this complexity does not add new insight. In particular, for this section, we will set $\theta_{low} = -1$. The other parameters of CHM can be arbitrary, except for r which depends on the separation.

Let $\mathbf{C} = \{C_1, \ldots, C_k\}$ be $(r, 2r)$-separated. A point z *touches* a cluster C_i if $d_{\mathrm{E}}(z, x) \leqslant r$ for any $x \in C_i$. For a set O of outliers, say a cluster C_i is *untouched by* O if no point in O touches it. We show that at most $\varepsilon' k$ terms in the sum in $\mathcal{A}_\gamma$ differ for $\widetilde{\mathbf{C}}$ versus $\widetilde{\mathbf{C}}'$. This follows from two simple claims:

(a) If C_i is untouched by O, then the i^{th} term is the same in the sums for both $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}})$ and $\mathcal{A}_\gamma(\mathbf{C}, \widetilde{\mathbf{C}}')$.

(b) Each outlier in O can touch at most one cluster in $\mathbf{C}$.

To see the first of the these claims, notice that if a cluster C_i is untouched, then $d_{\mathrm{E}}(x, z) > r$ for any $z \in \mathsf{O} \cup (S \setminus C_i)$. Therefore, no element of C_i will ever merge with an element outside of C_i, and the output of CHM will be the same with respect to C_i on both inputs S and $S \cup \mathsf{O}$. The second claim follows directly from the $(r, 2r)$-separated property. Indeed, if an outlier z has $d_{\mathrm{E}}(x, z) \leqslant r$ for any point $x \in C_i$, then it must be that $d_{\mathrm{E}}(y, z) > r$ for all $y \in C_j$ with $j \neq i$. Putting these claims together, at most $\varepsilon' k$ terms differ in the sums for $\mathcal{A}_\gamma$ and thus the accuracies differ by at most ε', since each term is at most 1. □

Notice that this is optimal since $\varepsilon' k$ outliers can clearly modify $\varepsilon' k$ clusters. For DNA storage data recovery, if we desire $1 - \varepsilon$ accuracy overall, and we expect at most $\varepsilon' k$ outliers, then we simply need to aim for a clustering with accuracy at least $1 - \varepsilon + \varepsilon'$.

5.6 Experiments

We experimentally evaluate our algorithm on real and synthetic data, measuring accuracy and wall clock time. Table 5.1 describes our datasets. We evaluate accuracy on the real data by

comparing the found clusterings to a gold standard clustering. We construct the gold standard by using the original reference strands, and we group the reads by their most likely reference using an established alignment tool (see Section 5.6.3 for full details). The synthetically generated data resembles real data distributions and properties [123]. We implement our algorithm in C++ using MPI. We run tests on Microsoft Azure virtual machines (size H16mr: 16 cores, 224 GB RAM, RDMA network).

Table 5.1: Datasets. Real data from Organick et. al. [123]. Synthetic data follow Definition 14.

Dataset	# Reads	Avg. Length	Description
3.1M real	3,103,511	150	Movie file stored in DNA
13.2M real	13,256,431	150	Music file stored in DNA
58M real	58,292,299	150	Collection of files (40MB stored in DNA; includes above)
12M real	11,973,538	110	Text file stored in DNA
5.3B synthetic	5,368,709,120	110	Noise $p = 4\%$; cluster size $s = 10$.

5.6.1 Implementation and Parameter Details

For the edit distance threshold, we desire r to be just larger than the cluster diameter. With p noise, we expect the diameter to be at most $4pm$ with high probability. We observe $p \leqslant 4\%$ for real data, and we set $r = 25$, since $4pm = 24$ for $p = 0.04$ and $m = 150$.

For the binary signatures, we observe that choosing larger q separates clusters better, but it also increases overhead, since $\sigma_q(x) \in \{0,1\}^{4^q}$ is very high-dimensional. To remedy this, we used a blocking approach. We partitioned x into blocks of 22 characters and computed σ_3 of each block, concatenating these 64-bit strings for the final signature. On synthetic data, we found that setting $\theta_{low} = 40$ and $\theta_{high} = 60$ leads to very reduced running time while sacrificing negligible accuracy.

For the hashing, we set w, ℓ to encourage collisions of close pairs and discourage collisions of far pairs. Following Theorem 5.4, we set $w = \lceil \log_4(m) \rceil = 4$ and $\ell = 12$, so that

$w + \ell = 16 = \log_4 n$ with $n = 2^{32}$. Since our clusters are very small, we find that we can further filter far pairs by concatenating two independent hashes to define a bucket based on this 64-bit value. Moreover, since we expect very few reads to have the same hash, instead of comparing all pairs in a hash bucket, we sort the reads based on hash value and only compare adjacent elements. For communication, we use only the first 20 bits of the hash value, and we uniformly distribute clusters based on this.

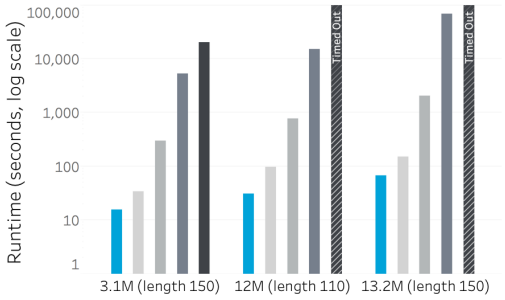
Finally, we conservatively set the number of iterations to 780 total (26 communication rounds, each with 30 local iterations) because this led to 99.9% accuracy on synthetic data (even with $\gamma = 1.0$).

Starcode Parameters Starcode [157] takes a distance threshold $d \in \{1, 2, \ldots, 8\}$ as an input parameter and finds all clusters with radius not exceeding this threshold. We run Starcode for various settings of d, with the intention of understanding how Starcode's accuracy and running time change with this parameter. We use Starcode's sphere clustering "-s" option, since this has performed most accurately on sample data, and we use the "-t" parameter to run Starcode with 16 threads.

5.6.2 Discussion

Figure 5.2 shows that our algorithm outperforms Starcode, the state-of-the-art clustering algorithm for DNA sequences [157], in both accuracy and time. As explained above, we have set our algorithm's parameters based on theoretical estimates. On the other hand, we vary Starcode's distance threshold parameter $d \in \{2, 4, 6, 8\}$. We demonstrate in Figures 5.2a and 5.2b that increasing this distance parameter significantly improves accuracy on real data, but also it also greatly increases Starcode's running time. Both algorithms achieve high accuracy for $\gamma = 0.6$, and the gap between the algorithms widens as γ increases. In Figure 5.2a, we show that our algorithm achieves more than a 1000x speedup over the most accurate setting of Starcode on three real datasets of varying sizes and read lengths. For $d \in \{2, 4, 6\}$, our algorithm has a smaller speedup and a larger improvement in accuracy.

(a) Time Comparison (log scale)



(b) Accuracy Comparison

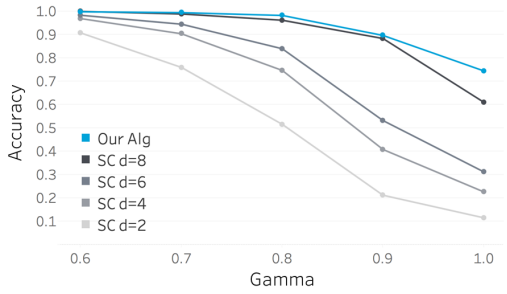


Figure 5.2: Comparison to Starcode. Figure 5.2a plots running times on three real datasets of our algorithm versus four Starcode executions using four distance thresholds $d \in \{2, 4, 6, 8\}$. For the first dataset, with 3.1M real reads, Figure 5.2b plots $\mathcal{A}_\gamma$ for varying $\gamma \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$ of our algorithm versus Starcode. We stopped Starcode if it did not finish within 28 hours. We ran tests on one processor, 16 threads.

Figure 5.3a shows how our algorithm's clustering accuracy increases with the number of communication rounds, where we evaluate $\mathcal{A}_\gamma$ with $\gamma = 0.9$. Clearly, using 26 rounds is quite conservative. Nonetheless, our algorithm took only 46 minutes wall clock time to cluster 5.3B synthetic reads on 24 processors (384 cores). We remark that distributed MapReduce-based algorithms for string similarity joins have been reported to need tens of minutes for only tens of millions of reads [58, 150].

Figure 5.3b demonstrates the effect of binary signatures on runtime. Recall that our algorithm uses signatures in two places: merging clusters when $d_H(\sigma(x), \sigma(y)) \leqslant \theta_{low}$, and filtering pairs when $d_H(\sigma(x), \sigma(y)) > \theta_{high}$. This leads to four natural variants: (i) omitting signatures, (ii) using them for merging, (iii) using them for filtering, or (iv) both. The biggest improvement (20x speedup) comes from using signatures for filtering (comparing (i) vs. (iii)). This occurs because the cheap Hamming distance filter avoids a large number of expensive edit distance computations. Using signatures for merging provides a modest 30%

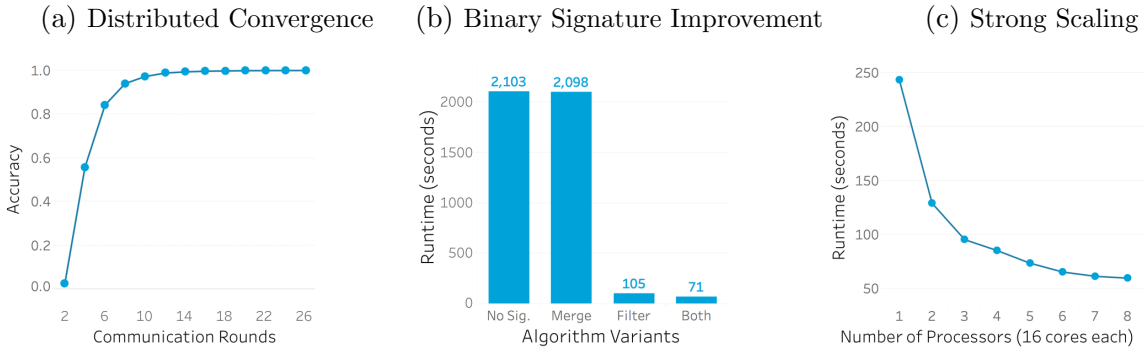


Figure 5.3: Empirical results for our algorithm. Figure 5.3a plots accuracy $\mathcal{A}_{0.9}$ of intermediate clusterings (5.3B synthetic reads, 24 processors). Figure 5.3b shows single-threaded running times for four variants of our algorithm, depending on whether it uses signatures for merging and/or filtering (3.1M real reads; single thread). Figure 5.3c plots times as the number of processors varies from 1 to 8, with 16 cores per processor (58M real reads).

improvement (comparing (iii) vs. (iv)); this gain does not appear between (i) and (ii) because of time it takes to compute the signatures. Overall, the effectiveness of signatures justifies their incorporation into an algorithm that already filters based on hashing.

Figure 5.3c evaluates the scalability of our algorithm on 58M real reads as the number of processors varies from 1 to 8. At first, more processors lead to almost optimal speedups. Then, the communication overhead outweighs the parallelization gain. Achieving perfect scalability requires greater understanding and control of the underlying hardware and is left as future work.

5.6.3 Additional Information about Datasets and Experimental Setup

We use dedicated virtual machines on Microsoft Azure within a single region and virtual network. They machines have Azure size H16mr. The specifications are as follows: Intel E5-2667 V3 3.2 GHz processors, utilizing 224GB DDR4 memory and 2TB SSD-based local

storage. They have a dedicated RDMA backend network enabled by FDR InfiniBand network. The Linux operating system uses image Centos 7.1-HPC.

We implemented out algorithms using C++. We used the MPICH MPI-3 compilers and the Intel MPI 5.2 runtime library. We compiled with the -O2 flag. For MPI, we allocated a single rank per core, and the large communication steps of our algorithm are implemented with non-blocking windows, which support Remote Direct Memory Access (RDMA).

Real Datasets Used for Evaluation

We provide details about the real datasets that we used for experimental evaluation. The data came from a DNA storage system presented by Organick et. al. [123]. We explain the way we have processed their sequencing data to generate our datasets, and we explain the methods we used to produce a gold standard on which we evaluate our algorithm and Starcode.

The data from Organick et. al. [123] is the output of an Illumina NextSeq machine. They prepared and sequenced molecules of synthetic DNA that store encoded data. The details of wetlab preparation and amplification can be found in the paper by Organick et. al. [123].

To generate a gold standard for clustering, we must find the true mapping between references and reads. To do this, we used a standard biological alignment tool, the Burrows-Wheeler Aligner (BWA) [108]. We run BWA with 10 threads using the command

```
bwa mem -t 10 [reference file] [read file]
```

BWA first indexes the references (the expected synthesized DNA) to create a database of references. After the references are indexed, BWA compares the references to all reads in order to find the best mapping between the references and reads. Each reference maps to many reads, which will be the basis for the clusters.

BWA also outputs an alignment between reads and references in the "bam" file format. From this, we extract the reads that aligned successfully. More precisely, each line in a bam file contains several fields, one of which is the read. Another field in the bam file keeps track of the most similar reference for this read, or leaves this field empty if no references are found

to align. For a read that aligns, the file also identifies the best alignment (that minimizes edit distance between the read and reference). We discard lines that do not align to any reference. Since reads of DNA often contain errors, as well as additional nucleotides before and after the expected strand, we used the alignment results to extract the portions of the reads that align to the references. Finally, from the BWA output, we know which reference each read is most similar to. Thus, we use this to create gold standard clusters.

We note that in our experiments, the clustering algorithms do not have access to the references, so the clusters generated using BWA should have much higher quality than anything produced by a clustering algorithm that does not have access to the references. Therefore, the alignments serve as a good gold standard.

The datasets were generated from a single sequencing run, which in total produced 58M reads after processing. The set of references for this run corresponds to about 40MB of data stored in DNA. Additionally, using the original file information, we are able to separate out three smaller datasets, each corresponding to a single encoded file. These represent a movie file, a text file, and a music file, respectively. The sizes of the these files appear in Table 5.1.

We remark that in practice, any true DNA storage system has an additional challenge. The data retrieval pipeline must identify the noisy reads from the sequencer output. The complication arises because the sequencer may append relatively long strings of random characters on either end of each read that it outputs. Since systems cannot align to true references, they must use additional information, such as a known substring, to extract a high-quality subsection of the read for clustering.

Synthetic Datasets Used for Evaluation. We follow Definition 14. The datasets with generated using Python 3.6, using the default random library. We implemented the uniform noise generation with "random.uniform", picking insertion, deletion, or substitution, each with equal probability for 4% total chance of error. We picked insertion and substitution characters uniformly at random in $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$.

Sources of Errors. In DNA data storage systems, errors arise not only from sequencing, but also from synthesis and amplification. This leads to a higher overall error rate than sequencing alone, with less than a ten-fold difference between substitution and insertion/deletion rates. In particular, the majority of reads contain at least one insertion or deletion, and therefore the clustering algorithm must be tailored to edit distance. Finally, although Illumina error rates may be low, future technologies such as Nanopore sequencing will require clustering algorithms that are robust to much higher error rates.

5.6.4 Accuracy Justification

We expound on our definition of accuracy and its relationship to DNA storage. Recall that the goal of clustering is to find the groups of reads that came from the same reference. Then, using these clusters, a method called trace reconstruction is used to determine the most likely reference for each group of noisy reads. The effectiveness of trace reconstruction depends on the number of *traces*. Therefore, larger clusters are clearly better. When the size of the original clusters varies, it is better to have more traces from small clusters, therefore we use this parameter γ to measure the fraction of recovered strings in a cluster.

Trace reconstruction methods do not have a built-in way to handle false positives. Fortunately, it is easy to check false positives because we know the threshold r. In particular, we assume that any clustering methods will check false positives before outputting the clusters.

5.7 Related Work

A longer survey on clustering algorithms and objectives already appears in Section 2.3.

Recent work identifies the difficulty of clustering datasets containing large numbers of small clusters. Betancourt et. al. [35] calls this "microclustering" and proposes a Bayesian non-parametric model for entity resolution datasets. Kobren et. al. [102] calls this "extreme clustering" and studies hierarchical clustering methods. DNA data storage provides a new domain for micro/extreme clustering, with interesting datasets and important consequences [40, 68, 74, 123, 152].

Large-scale, extreme datasets – with billions of elements and hundreds of millions of clusters – are an obstacle for many clustering techniques [51, 80, 90, 115]. We demonstrate that DNA datasets are well-separated, which implies that our algorithm converges quickly to a highly-accurate solution. It would be interesting to determine the minimum requirements for robustness in extreme clustering.

One challenge of clustering for DNA storage comes from the fact that reads are strings with edit errors and a four-character alphabet. Edit distance is regarded as a difficult metric, with known lower bounds in various models [1, 18, 24]. Similarity search algorithms based on MinHash [41, 43] originally aimed to find duplicate webpages or search results, which have much larger natural language alphabets. However, known MinHash optimizations [110, 111] may improve our clustering algorithm.

5.7.1 Metric Embeddings for Edit Distance

We mention a connection to metric embeddings for non-repetitive strings. Charikar and Krauthgamer [47] call $x \in \Sigma^m$ a *t-non-repetitive* string if all of its t-grams are distinct. They provide an embedding into the ℓ_1 metric space with distortion $O(t \log m)$ for the submetric of edit distance corresponding to considering only t-non-repetitive stings. For random strings, we prove in Lemmas 5.10 and 5.11 that the binary signatures $\sigma_q(x)$ provide an embedding into Hamming space with distortion $O(\log m)$ with high probability when $q = 2 \log_4 m$. It is natural to wonder if our binary signatures work in general for $O(\log m)$-non-repetitive strings (since random strings will have this property with high probability). Unfortunately, it is easy to construct pairs of example strings with linear edit distance but whose signatures have only logarithmic Hamming distance ($\approx q$). Therefore, it is an interesting open question to find an efficient metric embedding into ℓ_1 that has distortion $O(1)$ for strings under our random model. We pose a formal question in Section 7.2.

Any such embedding must crucially use that the strings are random since Andoni and Krauthgamer [18] prove that embedding 1-non-repetitive strings into ℓ_1 requires distortion $\Omega(\log m / \log \log m)$. Andoni and Krauthgamer also study a related question, about

approximately computing edit distance under a random model [19].

Chakraborty, Goldenberg, and Koucký explore the question of preserving small edit distances with a binary embedding [46]. This embedding was adapted by Zhang and Zhang [156] for approximate string similarity joins. We leave a thorough comparison to these papers as future work, along with obtaining better theoretical bounds for hashing or embeddings [47, 124] under our data distribution.

5.8 Conclusion

We highlighted a clustering task motivated by DNA data storage. We proposed a new distributed algorithm and hashing scheme for edit distance. Experimentally and theoretically, we demonstrated our algorithm's effectiveness in terms of accuracy, performance, scalability, and robustness.

We plan to release one of our real datasets. We hope our dataset and data model will lead to further research on clustering and similarity search for computational biology or other domains with strings.

For future work, our techniques may also apply to other metrics and to other applications with large numbers of small, well-separated clusters, such as entity resolution or deduplication [52, 66, 89]. Finally, our work motivates a variety of new theoretical questions, such as studying the distortion of embeddings for random strings under our generative model.

Chapter 6

EDGE ESTIMATION WITH INDEPENDENT SET QUERIES

Oracle models provide a way to understand the complexity of data science algorithms. Here, we imagine a very informative, but expensive, oracle that provides a small amount of information about the dataset. An oracle can be based on an experiment (*e.g.,* testing if a group of people contains any infected individuals) or on a computation (*e.g.,* determining the degree of a vertex in a graph). Then, we measure the number of oracle calls needed to accurately estimate some parameter of interest. In the case of graph parameter estimation, we will design randomized approximation algorithms that estimate the number of edges in an n-vertex graph while making only $o(n)$ queries to some oracle. As one interpretation, this asks whether it is possible to estimate this parameter without *learning* the whole graph. We provide several positive results in this direction.

6.1 Introduction

We investigate the problem of estimating the number of edges in a simple, unweighted, undirected graph $G = ([n], E)$, where $[n] \coloneqq \{1, 2, \ldots, n\}$ and $\mathsf{m} = |E|$, using only an oracle that answers independent set queries. For a parameter $\varepsilon > 0$, we wish to output an estimate $\widetilde{\mathsf{m}}$ satisfying $(1-\varepsilon)\mathsf{m} \leqslant \widetilde{\mathsf{m}} \leqslant (1+\varepsilon)\mathsf{m}$ with high probability. We consider randomized, adaptive algorithms with access to one of the two following oracles:

- BIS (Bipartite independent set) oracle: Given disjoint subsets $U, V \subseteq [n]$, a BIS query answers whether U, V satisfy $\mathsf{m}(U, V) = 0$, where $\mathsf{m}(U, V)$ denotes the number of edges with one endpoint in U and the other in V.

- IS (Independent set) oracle: Given a subset $U \subseteq [n]$, an IS query answers whether U

satisfies $\mathsf{m}(U) = 0$, where $\mathsf{m}(U)$ denotes the number of edges with both endpoints in U.

Previous work on graph parameter estimation has primarily focused on *local* queries, such as *degree* queries (which output the degree of a vertex v), *edge existence* queries (which answer whether a pair $\{u, v\}$ forms an edge), or *neighbor* queries (which provide the i^{th} neighbor of a vertex v). However, such queries cannot achieve sub-polynomial query costs on certain graphs identified by Feige [70] and Goldreich and Ron [75]. These queries can only obtain *local* information about the graph. This motivates an investigation of other queries that may enable efficient parameter estimation. The independent set queries described above naturally generalize an edge existence query, and their non-locality opens the door for sub-polynomial query algorithms for various graph parameter estimation tasks.

6.1.1 Motivation and related work

The most relevant motivation for BIS and IS queries comes from the area of sub-linear time algorithms for graph parameter estimation. BIS and IS queries also have interesting connections to the classical area of group testing, to emptiness versus counting questions in computational geometry, and to the complexity of decision versus counting problems.

Graph parameter estimation. Feige [70] showed how to use $O(\sqrt{n}/\varepsilon)$ degree queries to output $\widetilde{\mathsf{m}}$ that satisfies $\mathsf{m} \leqslant \widetilde{\mathsf{m}} \leqslant (2+\varepsilon)\mathsf{m}$, where $\mathsf{m} = |E|$. Moreover, he showed that any algorithm achieving better than a 2-approximation must use a nearly linear number of degree queries. Goldreich and Ron [75] showed that by using both degree and neighbor queries, the approximation improves to $(1-\varepsilon)\mathsf{m} \leqslant \widetilde{\mathsf{m}} \leqslant (1+\varepsilon)\mathsf{m}$ by using $\sqrt{n} \cdot \mathrm{poly}(\log n, 1/\varepsilon)$ queries. It is worth nothing that Feige [70] and Goldreich and Ron [75] have identified certain hard instances showing that these upper bounds cannot be improved, up to polylog factors.

Related work approximates the number of stars [77], the minimum vertex cover [122], the number of triangles [137, 63], and the number of k-cliques [64]. A special case of BIS query

(where one of the bipartition sets is a singleton) has been used for testing k-colorability of graphs [33] and edge estimation [148].

Group testing. A classic estimation problem involves efficiently approximating the number of defective items or infected individuals in a certain collection or population [50, 60, 143]. To query a population, a small group is formed, and all the individuals in the group are tested in one shot. For example, in genome-wide association studies, combined pools of DNA may be tested as a group for certain variants [100]. In group testing, the result of a test often indicates only whether there is at least one infected or defective unit, or if there is none. Such a dichotomous outcome resembles the IS/BIS queries. In the graph setting, group testing suggests testing pairwise interactions between many items or individuals, instead of singular events.

Computational geometry. Certain geometric applications exhibit the phenomenon that emptiness queries have more efficient algorithms than counting queries. For example, in three dimensions, for a set P of n points, half-space counting queries (i.e., what is the size of the set $|P \cap h|$, for a query half-space h), can be answered in $O(n^{2/3})$ time, after near-linear time preprocessing. On the other hand, emptiness queries (i.e., is the set $P \cap h$ empty?) can be answered in $O(\log n)$ time. Aronov and Har-Peled [22] used this to show how to answer approximate counting queries (i.e., estimating $|P \cap h|$), with polylogarithmic emptiness queries.

As another geometric example, consider the task of counting edges in disk intersection graphs using GPUs [71]. For these graphs, IS queries decide if a subset of the disks have any intersection (this can be done using sweeping in $O(n \log n)$ time [45]). Using a GPU, one could quickly draw the disks and check if the sets share a common pixel. In cases like this – when IS and BIS oracles have fast implementations – algorithms exploiting independent set queries may be useful.

Decision versus counting complexity. A generalization of IS and BIS queries has previously appeared in a line of work investigating the relationship between decision and counting problems [140, 141, 57]. Stockmeyer [140, 141] showed how to estimate the number of satisfying assignments for a given circuit with queries to an NP oracle. Ron and Tsur [135] observed that Stockmeyer implicitly provided an algorithm for estimating set cardinality using *subset* queries, where a subset query specifies a subset $X \subseteq \mathcal{U}$ and answers whether $|X \cap S| = 0$ or not. Subset queries generalize IS and BIS queries because S corresponds to the set of edges in the graph and X is an arbitrary subset of pairs of vertices.

Indeed, consider subset queries in the context of estimating the number of edges in a graph. To this end, fix $|S| = \mathsf{m}$ (i.e., the number of edges in the graph) and $|\mathcal{U}| = \binom{n}{2}$ (the number of possible edges). Stockmeyer provided an algorithm using only $O(\log\log \mathsf{m} \cdot \mathrm{poly}(1/\varepsilon))$ subset queries to estimate m within a factor of $(1+\varepsilon)$ with a constant success probability. Note that for a high probability bound, which is what we focus on in this chapter, the algorithm would naively require $O(\log n \cdot \log\log \mathsf{m} \cdot \mathrm{poly}(1/\varepsilon))$ queries to achieve success probability at least $1 - 1/n$. Falahatgar *et. al.* [69] gave an improved algorithm that estimates m up to a factor of $(1+\varepsilon)$ with probability $1-\delta$ using $2\log\log \mathsf{m} + O\left((1/\varepsilon^2)\log(1/\delta)\right)$ subset queries. Nearly matching lower bounds are also known for subset queries [140, 141, 135, 69]. Ron and Tsur [135] also study a restriction of subset queries, called *interval queries*, where they assume that the universe $\mathcal{U}$ is ordered and the subsets must be intervals of elements. We view the independent set queries that we study as another natural restriction of subset queries.

Analogous to Stockmeyer's results, a recent work of Dell and Lapinskas [57] provides a framework that relates edge estimation using BIS and edge existence queries to a question in fine-grained complexity. They study the relationship between decision and counting versions of problems such as 3SUM and Orthogonal Vectors. We first describe their edge estimation result and then explain their connection to fine-grained complexity. They proved that, for a bipartite graph, using $O(\varepsilon^{-2}\log^6 n)$ BIS queries, and $\varepsilon^{-4} n \cdot \mathrm{polylog}(n)$ edge existence queries, one can output a number $\widetilde{\mathsf{m}}$, such that, with probability at least $1 - 1/n^2$, we have $(1-\varepsilon)\mathsf{m} \leqslant \widetilde{\mathsf{m}} \leqslant (1+\varepsilon)\mathsf{m}$.

Dell and Lapinskas [57] used their edge estimation algorithm to obtain approximate counting algorithms for problems in fine-grained complexity. For instance, given an algorithm for 3SUM with runtime T, they obtain an algorithm that estimates the number of YES instances of 3SUM with runtime $O(T\varepsilon^{-2}\log^6 n) + \varepsilon^{-4}n \cdot \text{polylog}(n)$. The relationship is quite simple and natural. The decision version of 3SUM corresponds to checking if there is at least one edge in a certain bipartite graph. The counting version then corresponds to counting the edges in this graph. We note that in their application, the large number $O(n \cdot \text{polylog}(n))$ of edge existence queries does not affect the dominating term in the overall time in their reduction; the larger term in the time is a product of the time to decide 3SUM and the number of BIS queries.

6.1.2 Our results

We describe two new algorithms. In what follows, let $G = ([n], E)$ be a simple graph with $\mathsf{m} = |E|$ edges.

The Bipartite Independence Oracle. We present an algorithm that uses BIS queries and computes an estimate $\widetilde{\mathsf{m}}$ for the number of edges in G, such that $(1-\varepsilon)\mathsf{m} \leqslant \widetilde{\mathsf{m}} \leqslant (1+\varepsilon)\mathsf{m}$. The algorithm performs $O(\varepsilon^{-4}\log^{14} n)$ BIS queries, and succeeds with high probability (see Theorem 6.16 for a precise statement). Since $\text{polylog}(n)$ BIS queries can simulate a degree query (see Section 6.4.4), one can obtain a $(2+\varepsilon)$-approximation of m by using Feige's algorithm [70], which uses degree queries. This gives an algorithm that uses $O\left(\sqrt{n} \cdot \text{polylog}(n)/\text{poly}(\varepsilon)\right)$ BIS queries. Our new algorithm provides significantly better guarantees, in terms of both the approximation and number of BIS queries.

Compared to the result of Dell and Lapinskas [57], our algorithm uses exponentially fewer queries, since we do not spend $n \cdot \text{polylog}(n)$ edge existence queries. In terms of their specific applications, our improvement does not seem to imply anything for fine-grained complexity. We leave open the question of finding problems where a more efficient BIS algorithm would lead to new decision versus counting complexity results.

Query Types	Approximation	# Queries (up to const. factors)	Reference
Edge existence	$1+\varepsilon$	$(n^2/\mathsf{m})\ \mathrm{poly}(\log n, 1/\varepsilon)$	Folklore (see Section 6.5.2)
Degree	$2+\varepsilon$	$\sqrt{n}\log n/\varepsilon$	[70]
Degree + neighbor	$1+\varepsilon$	$\sqrt{n}\ \mathrm{poly}(\log n, 1/\varepsilon)$	[75]
Subset	$1+\varepsilon$	$\mathrm{poly}(\log n, 1/\varepsilon)$	[141, 69]
BIS + edge existence	$1+\varepsilon$	$n\ \mathrm{poly}(\log n, 1/\varepsilon)$	[57]
BIS	$1+\varepsilon$	$\mathrm{poly}(\log n, 1/\varepsilon)$	This Work
IS	$1+\varepsilon$	$\min\left(\sqrt{\mathsf{m}},\ n^2/\mathsf{m}\right)\cdot\mathrm{poly}(\log n, 1/\varepsilon)$	This Work

Table 6.1: Comparison of the best known algorithms using a variety of queries for estimating the number of edges m in a graph with n vertices. The bounds stated are for high probability results, with error probability at most $1/n$. Constant factors are suppressed for readability.

The Ordinary Independence Oracle. We also present a second algorithm, which uses only IS queries and computes a $(1+\varepsilon)$-approximation. The algorithm performs $O(\varepsilon^{-4}\log^5 n + \min(n^2/\mathsf{m}, \sqrt{\mathsf{m}})\cdot\varepsilon^{-2}\log^2 n)$ IS queries (see Theorem 6.25). In particular, the number of IS queries is bounded by $O(\varepsilon^{-4}\log^5 n + \varepsilon^{-2}n^{2/3}\log^2 n)$.

The first term in the minimum (i.e., $\approx n^2/\mathsf{m}$) comes from a folklore algorithm for estimating set cardinality using membership queries (see Section 6.2.2). The second term in the minimum (i.e., $\approx \sqrt{\mathsf{m}}$) is the number of queries used by our new algorithm.

We observe that BIS queries are surprisingly more effective for estimating the number of edges than IS queries. Shedding light on this dichotomy is one of the main contributions of this work.

Comparison with other queries. Table 6.1 summarizes the results for estimating the number of edges in a graph in the context of various query types. Given some of the results in Table 6.1 on edge estimation using other types of queries, a natural question is how well BIS and IS queries can simulate such queries. In Section 6.4.4, we show that $O(\varepsilon^{-2}\log n)$

BIS queries are sufficient to simulate degree queries. On the other hand, we do not know how to simulate a neighbor query (to find a specific neighbor) with few BIS queries, but a random neighbor of a vertex can be found with $O(\log n)$ BIS queries (see [33]). For IS queries, it turns out that estimating the degree of a vertex v up to a constant factor requires at least $\Omega\left(n/\deg\left(v\right)\right)$ IS queries (see Section 6.5.3).

Notation. Throughout, log and ln denotes the logarithm taken in base two and e, respectively. For integers, u, k, let $[k] = \{1, \ldots, k\}$ and $[u : k] = \{u, \ldots, k\}$. The notation $x = \text{polylog}(n)$ means $x = O(\log^c n)$ for some constant $c > 0$. A collection of disjoint sets $U_1, \ldots, U_k$ such that $\bigcup_i U_i = U$, is a *partition* of the set U, into k *parts* (a part U_i might be an empty set). In particular, a (uniformly) *random partition* of U into k parts is chosen by coloring each element of U with a random number in $[k]$ and identifying U_i with the elements colored with i.

Throughout, we use $G = ([n], E)$ to the denote the input graph. The number of edges in G is denoted by $\mathsf{m} = |E|$. For a set $U \subseteq [n]$, let $E\left(U\right) = \{uv \in E \mid u, v \in U\}$ be the set of edges between vertices of U in G. For two disjoint sets $U, V \subseteq [n]$, let $E\left(U, V\right)$ denote the set of edges between U and V: $E\left(U, V\right) = \{uv \in E \mid u \in U, v \in V\}$. Let $\mathsf{m}\left(U\right)$ and $\mathsf{m}\left(U, V\right)$ denote the number of edges in $E\left(U\right)$ and $E\left(U, V\right)$, respectively. We also abuse notation and let $\mathsf{m}(H)$ be the number of edges in a subgraph H (e.g., $\mathsf{m}(G) = \mathsf{m}$).

High probability conventions. Throughout the chapter, the randomized algorithms presented would succeed with high probability; that is, with probability $\geqslant 1 - 1/n^{O(1)}$. Formally, this means the probability of success is $\geqslant 1 - 1/n^c$, for some arbitrary constant $c > 0$. For all these algorithms, the value of c can be increased to any arbitrary value (i.e., improving the probability of success of the algorithm) by increasing the asymptotic running time of the algorithm by a constant factor that depends only on c. For the sake of simplicity of exposition, we do not explicitly keep track of these constants (which are relatively well-behaved).

6.1.3 Overview of the algorithms

The BIS algorithm

Our discussion of the BIS algorithm follows Figure 6.1, which depicts the main components of one level of our recursive algorithm. Our algorithms rely on several building blocks, as described next.

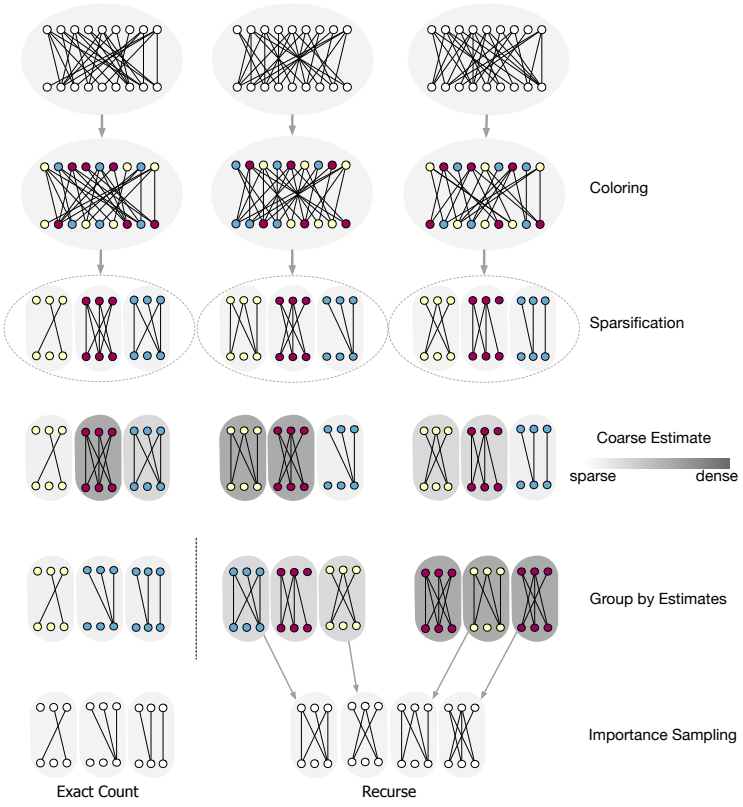


Figure 6.1: A depiction of one level of the BIS algorithm. In the first step, we color the vertices and sparsify the graph by only looking at the edges between vertices of the same color. In the second step, we coarsely estimate the number of edges in each colored subgraph. Next, we group these subgraphs based on their coarse estimates, and we subsample from the groups with a relatively large number of edges. In the final step, we exactly count the edges in the sparse subgraphs, and we recurse on the dense subgraphs.

Exactly count edges. One can exactly count the edges between two subsets of vertices, with a number of queries that scales nearly linearly in the number of such edges. Specifically, a simple deterministic divide and conquer algorithm to compute $\mathsf{m}(U, V)$ using $O(\mathsf{m}(U, V) \log n)$ BIS queries is described below in Lemma 6.8.

Sparsify. The idea is now to sparsify the graph in such a way that the number of remaining edges is a good estimate for the original number of edges (after scaling). Consider sparsifying the graph by coloring the vertices of graph, and only looking at the edges going between certain pairs of color classes (in our algorithm, these pairs are a matching of the color classes). We prove that it suffices to only count the edges between these color classes, and we can ignore the edges with both endpoints inside a single color class.

For $1 \leqslant k \leqslant \lfloor n/2 \rfloor$, let $U_1, \ldots, U_k, V_1, \ldots, V_k$ be a uniformly random partition of $[n]$. Then, we have

$$\mathbb{P}\left[\left| 2k \sum_{i=1}^{k} \mathsf{m}(U_i, V_i) - \mathsf{m} \right| \geqslant ck\sqrt{\mathsf{m}} \log n \right] \leqslant \frac{1}{n^4}, \tag{6.1}$$

where c is some constant. For the proof of this inequality see Section 6.3. Specifically, if we set G_i to be the induced bipartite subgraph on U_i and V_i, then $2k \sum_i \mathsf{m}(G_i)$ is a good estimate for $\mathsf{m}(G)$.

Now the graph is bipartite. The above sparsification method implies that we can assume without loss of generality that the graph is bipartite. Indeed, invoking the lemma with $k = 2$, we see that estimating the number of edges between the two color classes is equivalent to estimating the total number of edges, up to a factor of two. For the rest of the discussion, we will consider colorings that respect the bipartition.

Coarse estimator. We give an algorithm that coarsely estimates the number of edges in a (bipartite) subgraph, up a $O(\log^2 n)$ factor, using only $O(\log^3 n)$ BIS queries.

The subproblems. After coloring the graph, we have reduced the problem to estimating the total number of edges in a collection of (disjoint) bipartite subgraphs. However, certain subgraphs may still have a large number of edges, and it would be too expensive to directly use the exact counting algorithm on them.

Reducing the number of subgraphs in a collection, via importance sampling. Using the coarse estimates we can form $O(\log n)$ groups of bipartite subgraphs, where each group contains subgraphs with a comparable number of edges. For the groups with only a polylogarithmic number of edges, we can exactly count edges using polylog(n) BIS queries via the exact count algorithm mentioned above. For the remaining groups, we subsample a polylogarithmic number of subgraphs from each group. Since the groups contained subgraphs with a similar number of edges, the number of edges in the subsampled subgraphs will be proportional to the total number of edges in the group, up to a scaling factor depending on the factors by which sizes in a given group can vary, and this new estimate is a good approximation to the original quantity, with high probability. This corresponds to the technique of *importance sampling* that is used for variance reduction when estimating a sum of random variables that have comparable magnitudes.

Sparsify and reduce. We use the sparsification algorithm on each graph in our collection. This increases the number of subgraphs while reducing (by roughly a factor of k) the total number of edges in these graphs. The number of edges in the new collection is a reliable estimate for the number in the old collection. We will choose k to be a constant so that every sparsification round reduces the number of edges by a constant factor.

If the number of graphs in the collection becomes too large, then we reduce it in one of two ways. For the sparse subgraphs, we exactly count the number of edges using only polylog(n) queries. For the dense subgraphs, we can apply the above importance sampling technique and retain only polylog(n) subgraphs. Every basic operation in this scheme requires polylog(n) BIS queries, and the number of subgraphs is polylog(n). Therefore, a round can

be implemented using polylog(n) BIS queries. Now, since every round reduces the number of edges by a constant factor, the algorithm terminates after $O(\log n)$ rounds, resulting in the desired estimate for m using only polylog(n) queries in total. Figure 6.1 depicts the main components of one round.

We have glossed over some details regarding the reweighting of intermediate estimates. Recall that both the sparsification and importance sampling steps involve subsampling and rescaling. To handle this, the algorithm will maintain a weight value for each subgraph in the collection (starting with unit weight). Then, these weights will be updated throughout the execution, and they will be used during coarse estimation. For the final estimate, the algorithm will output a weighted sum of the estimates for the remaining subgraphs, in addition to the weighted version of the exactly counted subgraphs. By using these weights to properly rescale estimates and counts, the algorithm will achieve a good estimate for m with high probability.

The IS algorithm

We move on to describe our second algorithm, based on IS queries. As with the BIS algorithm, the main building block for the IS algorithm is an efficient way to exactly count edges using IS queries. The exact counting algorithm works by first breaking the vertices of the graph into independent sets in a greedy fashion, and then grouping these independent sets into larger independent sets using (yet again) a greedy algorithm. The resulting partition of the graph into independent sets has the property that every two sets have an edge between them, and this partition can be computed using a number of queries that is roughly m. This is beneficial, because when working on the induced subgraph on two independent sets, the IS queries can be interpreted as BIS queries. As such, edges between parts of the partition can be counted using the exact counting algorithm, modified to use IS queries. The end result is, that for a given set $U \subseteq [n]$, one can compute $\mathsf{m}(U)$, the number of edges with both endpoints in U, using $O(\mathsf{m}(U) \log n)$ IS queries. This algorithm is described in Section 6.5.1.

Now, we can sparsify the graph to reduce the overall number of IS queries. In contrast to

the BIS queries, we do not know how to design a coarse estimator using only IS queries (see Section 6.5.3). This prohibits us from designing a similar algorithm. Instead, we estimate the number of edges in one shot, by coloring the graph with a large number of colors and estimating the number of edges going between a matching of the color classes.

Using a matching of the color classes for sparsification seems somewhat counter intuitive. An initial sparsification attempt might be to count only the edges going between a single pair of colors. If the total number of colors is $2k$, then we expect to see $\mathsf{m}/\binom{2k}{2}$ edges between this pair. Therefore, we could set k to be large and invoke Lemma 6.20. Scaling by a factor of $\binom{2k}{2}$, we would hope to get an *unbiased* estimator for m.

Unfortunately, a star graph demonstrates that this approach does not work, due to the large variance of this estimator. If we randomly color the vertices of the star graph with $2k$ colors, then out of the $\binom{2k}{2}$ pairs of color classes, only $2k-1$ pairs have any edge going between color classes. So, if we only chose one pair of color classes, then with high probability one of the following two cases occurs: either (i) there is no edge crossing the color pair, or (ii) the number of edges crossing the pair is $\approx \mathsf{m}/2k$. In both cases our estimate after scaling by a factor of $\binom{2k}{2}$ will be far from the truth.

At the other extreme, the vast majority of edges will be present if we look at the edges crossing *all pairs* of color classes. Indeed, the only edges we miss have both endpoints in a color class, and this accounts for only a $1/2k$ fraction of the total number of edges. Thus, this does not achieve any substantial sparsification.

By using a matching of the color classes, we simultaneously get a reliable estimate of the number of edges and a sufficiently sparsified graph (see Lemma 6.7). Let $U_1, \ldots, U_k, V_1, \ldots, V_k$ be a random partition of the vertices into $2k$ color classes. This implies that with high probability, the estimator $2k \sum_{i=1}^{k} \mathsf{m}(U_i, V_i)$ is in the range $\mathsf{m} \pm O(k\sqrt{\mathsf{m}} \log n)$. Hence, as long as we choose k to be less than $\varepsilon\sqrt{\mathsf{m}}/\mathrm{polylog}(n)$, we approximate m up to a factor of $(1 + O(\varepsilon))$. We use geometric search to find such a k efficiently.

To get a bound on the number of IS queries, we claim that we can compute $\sum_{i=1}^{k} \mathsf{m}(U_i, V_i)$ using Lemma 6.20, with a total of $\left(k + \frac{\mathsf{m}}{k}\right) \cdot \mathrm{polylog}(n)$ IS queries. The first term arises

since we have to make at least one query for each of the k color pairs (even if there are no edges between them). For the second term, we pay for both (i) the edges between the color classes and (ii) the total number of edges with both endpoints within a color class (since the number of IS queries in Lemma 6.20 scales with $\mathsf{m}(U \cup V)$). By the sparsification lemma, we know that (i) is bounded by $O(\mathsf{m}/k)$ with high probability and we can prove an analogous statement for (ii). Hence, plugging in a value of $k \approx \varepsilon\sqrt{\mathsf{m}}/\text{polylog}(n)$, the total number of IS queries is bounded by $\sqrt{\mathsf{m}} \cdot \text{polylog}(n)/\varepsilon$.

6.1.4 Outline

The rest of th chapter is organized as follows. We start at Section 6.2 by reviewing some necessary tools – concentration inequalities, importance sampling, and set size estimation via membership queries. In Section 6.3, we prove our sparsification result (Lemma 6.7).

In Section 6.4 we describe the algorithm for edge estimation for the BIS case. Section 6.4.1 describes the exact counting algorithm. In Section 6.4.2, we present the algorithm that uses BIS queries to coarsely estimate the number of edges between two subsets of vertices (Lemma 6.15). We combine these building blocks to construct our edge estimation algorithm using BIS queries in Section 6.4.3.

The case of IS queries is tackled in Section 6.5. In Section 6.5.1, we formally present the algorithms to exactly count edges between two subsets of vertices (Lemma 6.20). In Section 6.5.2, we present our algorithm using IS queries. In Section 6.5.3 we provide some discussion of why the IS case seems to be harder than the BIS case.

We conclude in Section 6.6 and discuss open questions.

6.2 Preliminaries

Here we present some standard tools that we need later on.

6.2.1 Importance sampling

Importance sampling is a technique for estimating a sum of terms. Assume that for each term in the summation, we can cheaply and quickly get an initial, coarse estimate of its value. Furthermore, assume that better estimates are possible but expensive. Importance sampling shows how to sample terms in the summation, then acquire a better estimate *only for the sampled terms*, to get a good estimate for the full summation. In particular, the number of samples is bounded independently of the original number of terms, depending instead on the coarseness of the initial estimates, the probability of success, and the quality of the final output estimate.

Lemma 6.1 (Importance Sampling)**.** *Let $U = \{x_1, \ldots, x_r\}$ be a set of numbers, all contained in the interval $[\alpha/b, \alpha b]$, for $\alpha > 0$ and $b \geqslant 1$. Let $\gamma, \varepsilon > 0$ be parameters. Consider the sum $\Gamma = \sum_{i=1}^{r} x_i$. For an arbitrary $t \geqslant \frac{b^4}{2\varepsilon^2}\left(1 + \ln \frac{1}{\gamma}\right)$, and $i = 1, \ldots, t$, let X_i be a random sample chosen uniformly (and independently) from the set U. Then, the estimate $Y = (r/t)\sum_{i=1}^{t} X_i$ for the value of Γ satisfies $\mathbb{P}[|Y - \Gamma| \geqslant \varepsilon\Gamma] \leqslant \gamma$.*

Proof. Observe that $r(\alpha/b) \leqslant \Gamma \leqslant r\alpha b$, and

$$\mu = \mathbb{E}[Y] = \mathbb{E}\left[(r/t)\sum_{i=1}^{t} X_i\right] = (r/t)\sum_{i=1}^{t} \mathbb{E}[X_i] = \frac{r}{t} \cdot t \cdot \frac{\Gamma}{r} = \Gamma.$$

Furthermore, we have $Z = (t/r)Y = \sum_{i=1}^{t} X_i$, $\mathbb{E}[Z] = (t/r)\Gamma$, and for the length, Δ_i, of the interval containing X_i, we have $\Delta_i^2 = (\alpha b - \alpha/b)^2 \leqslant \alpha^2 b^2$.

Using $r\alpha/b \leqslant \Gamma$, by Lemma 2.1, we have

$$\begin{aligned}
\mathbb{P}\left[|Y - \Gamma| \geqslant \varepsilon\Gamma\right] = \mathbb{P}\left[\left|\frac{t}{r}Y - \frac{t}{r}\Gamma\right| \geqslant \frac{t\varepsilon\Gamma}{r}\right] &\leqslant \mathbb{P}\left[\left|\sum_{i=1}^{t} X_i - \frac{t}{r}\Gamma\right| \geqslant \frac{t\varepsilon r\alpha/b}{r}\right] \\
&= \mathbb{P}\left[\left|\sum_{i=1}^{t} X_i - \mathbb{E}[Z]\right| \geqslant \frac{t\varepsilon\alpha}{b}\right] \\
&\leqslant 2\exp\left(-\frac{2\left(t\varepsilon\alpha/b\right)^2}{\sum_{i=1}^{t}\Delta_i^2}\right) \\
&\leqslant 2\exp\left(-\frac{2\left(t\varepsilon\alpha/b\right)^2}{t\alpha^2 b^2}\right) = 2\exp\left(-\frac{2t\varepsilon^2}{b^4}\right) \leqslant \gamma.
\end{aligned}$$

□

The above lemma enables us to reduce a summation with many numbers into a much shorter summation (while introducing some error, naturally). The list/summation reduction algorithm we need is described next.

Lemma 6.2 (Summation reduction)**.** *Let $(\mathcal{H}_1, w_1, e_1), \ldots, (\mathcal{H}_r, w_r, e_r)$ be given, where $\mathcal{H}_i$'s are some structures, and w_i and e_i are numbers, for $i = 1, \ldots, r$. Every structure $\mathcal{H}_i$ has an associated weight $\overline{\mathsf{w}}(\mathcal{H}_i) \geqslant 0$ (the exact value of $\overline{\mathsf{w}}(\mathcal{H}_i)$ is not given to us). In addition, let $\xi > 0$, γ, b, and M be parameters, such that:*

1. $\forall i \quad w_i, e_i \geqslant 1$,

2. $\forall i \quad e_i/b \leqslant \overline{\mathsf{w}}(\mathcal{H}_i) \leqslant e_i b$, *and*

3. $\Gamma = \sum_i w_i \cdot \overline{\mathsf{w}}(\mathcal{H}_i) \leqslant M$.

Then, one can compute a new sequence of triples $(\mathcal{H}'_1, w'_1, e'_1), \ldots, (\mathcal{H}'_t, w'_t, e'_t)$, that also complies with the above conditions, such that the estimate $Y = \sum_{i=1}^{t} w'_i \overline{\mathsf{w}}(\mathcal{H}'_i)$ is a multiplicative $(1 \pm \xi)$-approximation to Γ, with probability $\geqslant 1 - \gamma$. The running time of the algorithm is $O(r)$, and size of the output sequence is $t = O\left(b^4 \xi^{-2} (\log\log M + \log \gamma^{-1}) \log M\right)$.

Proof. We break the interval $[1, M]$ into $\log M$ intervals in the natural way, where the j^{th} interval is $J_j = \left[2^{j-1}, 2^j\right)$, for $j = 1, \ldots, h = \lceil \log M \rceil$, except if M is a power of 2, in which case the last interval is closed and also includes $2^h = M$. Input triples are sorted into h groups $U_1, \ldots, U_h$, where an input triple $(\mathcal{H}, w, e)$ is in U_j, if $ew \in J_j$. This mapping can be done in $O(r)$ time.

Let $\alpha = O\left(b^4 \xi^{-2} \left[1 + \ln(h/\gamma)\right]\right)$. For $j = 1, \ldots, h$, if $|U_j| \leqslant \alpha$, then set $R_j = U_j$, otherwise compute a sample R_j from U_j of size α. We associate weight $W_j = |U_j| / |R_j|$ with R_j. If a triple $(\mathcal{H}, w, e) \in U_j$, then we have that $w \cdot \overline{\mathsf{w}}(\mathcal{H}) \in \left[2^{j-1}/b, 2^j b\right]$.

For all $j \in [h]$, let $\Gamma_j = \sum_{(\mathcal{H},w,e)\in U_j} w \cdot \overline{\mathsf{w}}(\mathcal{H})$ be the total weight of structures in the j^{th} group. By Lemma 6.1, we have, with probability $\geqslant 1-\gamma/h$, that

$$Y_j = \Big(W_j \sum_{(\mathcal{H},w,e)\in R_j} \overline{\mathsf{w}}(\mathcal{H}) \cdot w\Big) \in \big[(1-\xi)\Gamma_j, (1+\xi)\Gamma_j\big].$$

Summing these inequalities over all $j \in [h]$, implies that Y is the desired approximation with probability $\geqslant 1-\gamma$.

Specifically, the output sequence is constructed as follows. For all $j \in [h]$, and for every triple $(\mathcal{H}, w, e) \in R_j$, we add $(\mathcal{H}, w \cdot W_j, e)$ to the output sequence. Clearly, the output sequence has $t = h\alpha = O\big(b^4\xi^{-2}(\log\log M + \log\gamma^{-1})\log M\big)$ elements. □

Remark 5. The algorithm of Lemma 6.2 does not use the entities $\mathcal{H}_i$ directly at all. In particular, the $\mathcal{H}'_i$s are just copies of some original structures. The only thing that the above lemma uses is the estimates $e_1, \ldots, e_r$ and the weights $w_1, \ldots, w_r$.

Remark 6. We are going to use Lemma 6.2, with $\xi = O(\varepsilon/\log n)$, $\gamma = 1/n^{O(1)}$, $b = O(\log n)$, and $M = n^2$. As such, the size of the output list is

$$L_{\text{len}} = O\left(\log^4 n \cdot \varepsilon^{-2}\log^2 n \cdot (\log\log n + \log n)\log n\right) = O(\varepsilon^{-2}\log^8 n).$$

6.2.2 Estimating subset size via membership oracle queries

We present here a standard tool for estimating the size of a subset via membership oracle queries. This is well known, but we provide the details for the sake of completeness, since we were unable to find a good reference that describe it in this form.

Lemma 6.3. *Consider two (finite) sets $B \subseteq U$, where $n = |U|$. Let $\varepsilon, \gamma \in (0,1)$ be parameters. Let $g > 0$ be a user-provided guess for the size of $|B|$. Consider a random sample R, taken with replacement from U, of size $r = \lceil c_5\varepsilon^{-2}(n/g)\log\gamma^{-1}\rceil$, where $c_5 = c_5(\gamma)$ is sufficiently large. Next, consider the estimate $Y = (n/r)\,|R \cap B|$ to $|B|$. Then, we have the following:*

1. *If $Y < g/2$, then $|B| < g$,*

2. *If* $Y \geqslant g/2$, *then* $(1-\varepsilon)Y \leqslant |B| \leqslant (1+\varepsilon)Y$.

Both statements above hold with probability $\geqslant 1-\gamma$.

Proof. (A) The bad scenario here is that $|B| \geqslant g$, but $Y < g/2$. Let $X_i = 1 \iff$ the i^{th} sample element is in B. We have that $Y = (n/r)X$, where $X = \sum_{i=1}^{r} X_i$. By assumption, we have

$$\mu = \mathbb{E}[X] = \frac{r\,|B|}{n} \geqslant \frac{rg}{n} \geqslant c_5 \frac{\log \gamma^{-1}}{\varepsilon^2}. \tag{6.2}$$

As such, by Chernoff's inequality (Lemma 2.2 2), we have that $\mathbb{P}[Y < g/2] = \mathbb{P}[X < rg/(2n)] = \mathbb{P}\big[X < (1-1/2)\mathbb{E}[X]\big] \leqslant \exp(-\mu/8) \leqslant \gamma^{c_5/(8\varepsilon^2 \ln 2)}$ and this is $\leqslant \gamma$ for c_5 a sufficiently large constant.

(B) We have two cases to consider. First suppose that $|B| < g/4$. In this case, if $X = \sum_{i=1}^{r} X_i$ is the random variable as described part (A), then each X_i is an indicator variable with probability $p = |B|/n < g/(4n)$ and $\mathbb{P}[Y \geqslant g/2] = \mathbb{P}[X \geqslant rg/(2n)] \leqslant \mathbb{P}[X' \geqslant rg/(2n)]$ where X' is the sum of r independent Bernoulli trials with success probability $g/(4n)$. Now $\mathbb{E}[X'] = \frac{rg}{4n} \geqslant c_5 \frac{\log \gamma^{-1}}{4\varepsilon^2}$ so $\mathbb{P}[Y \geqslant g/2] \leqslant \mathbb{P}[X' \geqslant rg/(2n)] = \mathbb{P}[X' \geqslant (1+1)\mathbb{E}[X']] \leqslant \exp(-\mathbb{E}[X']/3) \leqslant \gamma^{c_5/(12\varepsilon^2 \ln 2)}$ by Chernoff's inequality (Lemma 2.2 3) and again this is $\leqslant \gamma$ for c_5 a sufficiently large constant.

For the second case, suppose that $|B| \geqslant g/4$. Then, $\mathbb{E}[X] \geqslant \mathbb{E}[X'] \geqslant c_5 \frac{\log \gamma^{-1}}{4\varepsilon^2}$ and, since Y is a fixed multiple of X, by Chernoff's inequality (Lemma 2.2 2), we have

$$\mathbb{P}\big[Y < (1-\varepsilon)\mathbb{E}[Y]\big] = \mathbb{P}\big[X < (1-\varepsilon)\mathbb{E}[X]\big] \leqslant \exp\big(-\mathbb{E}[X]\varepsilon^2/2\big) \leqslant \gamma^{c_5/(8\ln 2)}$$

which is $\leqslant \gamma/2$ for $c_5 = c_5(\gamma)$ sufficiently large. Similarly, by Chernoff's inequality (Lemma 2.2 3),

$$\mathbb{P}\big[Y > (1+\varepsilon)\mathbb{E}[Y]\big] = \mathbb{P}\big[X > (1+\varepsilon)\mathbb{E}[X]\big] \leqslant \exp\big(-\mathbb{E}[X]\varepsilon^2/3\big) \leqslant \gamma^{c_5/(12\ln 2)}$$

which is $\leqslant \gamma/2$ for $c_5 = c_5(\gamma)$ is sufficiently large. (This last condition on c_5 is the most stringent.) Adding these two failure probabilities together gives a bound of at most γ as required. □

Lemma 6.4. *Consider two sets $B \subseteq U$, where $n = |U|$. Let $\xi, \gamma \in (0,1)$ be parameters, such that $\gamma < 1/\log n$. Assume that one is given an access to a membership oracle that, given an element $x \in U$, returns whether or not $x \in B$. Then, one can compute an estimate s, such that $(1-\xi)|B| \leqslant s \leqslant (1+\xi)|B|$, and computing this estimates requires $O((n/|B|)\xi^{-2}\log\gamma^{-1})$ oracle queries. The returned estimate is correct with probability $\geqslant 1-\gamma$.*

Proof. Let $g_i = n/2^{i+2}$. For $i = 1, \ldots, \log n$, use the algorithm of Lemma 6.3 with $\varepsilon = 0.5$, with the probability of failure being $\gamma/(8\log n)$, and let Y_i be the returned estimate. The algorithm stops this loop as soon as $Y_i \geqslant 4g_i$. Let I be the value of i when the loop stopped. The algorithm now calls Lemma 6.3 again with g_I and $\varepsilon = \xi$, and returns the value of Y, as the desired estimate.

Overall, for $T = 1 + \lceil \log n \rceil$, the above makes T calls to the subroutine of Lemma 6.3, and the probability that any of them to fail is $T\gamma/(8\log n) < \gamma$. Assume that all invocations of Lemma 6.3 were successful. In particular, Lemma 6.3 guarantees that if $Y > 4g_I \geqslant g_I/2$, then the estimate returned is $(1 \pm \varepsilon)$-approximation to the desired quantity.

Computing Y_i requires $r_i = O((n/g_i)\log(\log n/\gamma)) = O(2^i \log\gamma)$ oracle membership queries. As such, the number of membership queries performed by the algorithm overall is $\sum_i r_i + O((n/g_I)\varepsilon^{-2}\log(\log n/\gamma)) = O((n/|B|)\varepsilon^{-2}\log\gamma)$, as claimed. □

Estimating subset size via emptiness oracle queries

Consider the variant where we are given a set $X \subseteq U$. Given a query set $Q \subseteq U$, we have an *emptiness oracle* that tells us whether $Q \cap X$ is empty. Using an emptiness oracle, one can $(1 \pm \varepsilon)$-approximate the size of X using relatively few queries. The following result is implied by the work of Aronov and Har-Peled [22, Theorem 5.6] and Falahatgar *et. al.* [69] – the later result has better bounds if the failure probability is not required to be polynomially small.

Lemma 6.5 ([22, 69])**.** *Consider a set $X \subseteq U$, where $n = |U|$. Let $\varepsilon \in (0,1)$ be a parameter. Assume that one is given an access to an emptiness oracle that, given a query set $Q \subseteq U$, returns whether or not $X \cap Q \neq \varnothing$. Then, one can compute an estimate s such that*

$(1-\varepsilon)\,|X| \leqslant s \leqslant (1+\varepsilon)\,|X|$, *using* $O(\varepsilon^{-2}\log n)$ *emptiness queries. The returned estimate is correct with probability* $\geqslant 1 - 1/n^{O(1)}$.

We sketch the basic idea of the algorithm used in the above lemma. For a guess g of the size of X, consider a random sample Q where every element of U is picked with probability $1/g$. The probability that Q avoids X is $\alpha(g) = (1-1/g)^{|X|}$. The function $\alpha(g)$ is:

1. monotonically increasing,

2. close to zero when $g \ll |X|$,

3. $\approx 1/e$ for $g = |X|$, and

4. close to 1 if $g \gg |X|$.

One can estimate the value $\alpha(g)$ by repeated random sampling and checking if the random sample intersects X using emptiness queries. Given such an estimate one can then perform an approximate binary search for the value of g such that $\alpha(g) = 1/e$, which corresponds to $g = |X|$. See [22, 69] for further details.

6.3 Edge sparsification by random coloring

In this section, we present and prove that coloring vertices, and counting only edges between specific color classes provides a reliable estimate for the number of edges in the graph. This is distinct from standard graph sparsification algorithms which usually sparsify the edges of the graph directly (usually, by sampling edges).

We need the following technical lemma.

Lemma 6.6. *Let C be a set of u elements, colored randomly by k colors – specifically, for every element $x \in C$, one picks randomly (independently and uniformly) a color for it from the set $[k]$. For $\ell \in [k]$, let n_ℓ be the number of elements of C with color ℓ. Let n be a positive integer and $c > 1$ be an arbitrary constant. Then:*

1. *For any index $i \in [k]$, we have $\mathbb{P}\big[|n_i - u/k| > \sqrt{(cu/2)\ln n}\,\big] \leqslant 2/n^c$.*

2. *For any two distinct colors $i, j \in [k]$, we have $\mathbb{P}\big[|n_i - n_j| > \sqrt{2cu\ln n}\big] \leqslant 4/n^c$.*

3. *For any distinct indices $i, j \in [k]$, we have $\mathbb{E}[|n_i - n_j|] \leqslant 12\sqrt{u/2}$.*

Proof. (A) Let X_ℓ be the indicator variable that is 1 with probability $1/k$ and 0 otherwise, for $\ell = 1, \dots, u$. For $X = \sum_{\ell=1}^{k} X_\ell$, we have

$$\mathbb{P}\Big[|X - \mathbb{E}[X]| > \sqrt{(cu/2)\ln n}\Big] \leqslant 2\exp\left(-\tfrac{2}{u}\cdot\tfrac{cu}{2}\ln n\right) \leqslant 2/n^c,$$

using Chernoff's inequality (Lemma 2.2 1).

(B) Observe that $|n_i - n_j| \leqslant |n_i - u/k| + |u/k - n_j|$, and the claim follows from (A).

(C) Arguing as in (A), we have

$$\begin{aligned}\mathbb{E}[|n_i - u/k|] &\leqslant \sum_{i=0}^{\infty}\sqrt{(i+1)u/2}\cdot\mathbb{P}\Big[\sqrt{iu/2} < |X - \mathbb{E}[X]| \leqslant \sqrt{(i+1)u/2}\Big] \\ &\leqslant \sqrt{u/2}\sum_{i=0}^{\infty}\sqrt{i+1}\cdot\mathbb{P}\Big[|X - \mathbb{E}[X]| > \sqrt{iu/2}\Big] \leqslant 2\sqrt{u/2}\sum_{i=0}^{\infty}\sqrt{i+1}\cdot\exp(-i) \leqslant 6\sqrt{u/2}.\end{aligned}$$

The proof then follows as in (B). □

Lemma 6.7. *(A) Let $G = ([n], E)$ be a graph with m edges. For any $1 \leqslant k \leqslant \lfloor n/2 \rfloor$, let $U_1, \dots, U_{2k}$ be a uniformly random partition of $[n]$. Then, there is some constant $\varsigma > 1$, such that*

$$\mathbb{P}\left[\left|\frac{\mathsf{m}}{2k} - \sum_{i=1}^{k}\mathsf{m}\left(U_i, U_{k+i}\right)\right| \geqslant \varsigma\sqrt{\mathsf{m}}\log n\right] \leqslant \frac{1}{n^4}, \qquad \textit{and} \qquad \mathbb{P}\left[\left|\frac{\mathsf{m}}{2k} - \sum_{i=1}^{2k}\mathsf{m}\left(U_i\right)\right| \geqslant \varsigma\sqrt{\mathsf{m}}\log n\right] \leqslant \frac{1}{n^4}.$$

(B) Similarly, for disjoint sets $U, V \subseteq [n]$ and k such that $2 \leqslant k \leqslant \max\{|U|, |V|\}$, let $U_1, \dots, U_k$, $V_1, \dots, V_k$ be uniformly random partitions of U and V, respectively. Then, there is some constant $\varsigma > 1$, such that

$$\mathbb{P}\left[\left|\mathsf{m}\left(U, V\right) - k\sum_{i=1}^{k}\mathsf{m}\left(U_i, V_i\right)\right| \geqslant \varsigma k\sqrt{\mathsf{m}\left(U, V\right)}\log n\right] \leqslant 1/n^4.$$

Proof. (A) Consider the random process that colors vertex t, at time $t \in [n]$, with a uniformly random color $X_t \in [2k]$. The colors correspond to the partition of $[n]$ into classes $U_1, \ldots, U_{2k}$. Define

$$f(X_1, \ldots, X_n) = \sum_{i=1}^{k} \mathsf{m}\left(U_i, U_{k+i}\right).$$

The probability of a specific edge uv to be counted by f is $1/(2k)$. Indeed, fix the color of u, and observe that there is only one choice of the color of v, such that uv would be counted. As such, $\mathbb{E}[f] = \mathsf{m}/(2k)$ and $0 \leqslant f(X_1, \ldots, X_n) \leqslant \mathsf{m}$.

Let $N(t)$ be the set of neighbors of t in the graph and $\deg(t) = |N(t)|$ be the degree of t. Let $N_{<t} = N(t) \cap [t-1]$ and $N_{>t} = N(t) \cap [t+1 : n]$ be the before/after set of neighbors of t, respectively. Let $C^i_{<t}$ (resp. $C^i_{>t}$) be the number of neighbors of t in $N_{<t}$ (resp. $N_{>t}$) colored with color i. For a color $i \in [2k]$, let $\pi(i) = 1 + ((k + i - 1) \bmod 2k)$ be its matching color.

Let $T_i = \mathbb{E}[f \mid X_1, \ldots, X_{t-1}, X_t = i]$, and consider the quantity $\Delta_t = |T_i - T_j|$. Conditioned on the values of $X_1, \ldots, X_{t-1}$, this can be written as

$$\Delta_t = \left|T_i - T_j\right| = \left|C^{\pi(i)}_{<t} + \mathbb{E}\left[C^{\pi(i)}_{>t} \,\middle|\, X_t = i\right] - C^{\pi(j)}_{<t} - \mathbb{E}\left[C^{\pi(j)}_{>t} \,\middle|\, X_t = j\right]\right|.$$

To see why the above is true, observe that any edge involving two vertices in $[t-1]$ has the same contribution to T_i and T_j. Similarly, an edge with a vertex in $[t-1]$, and a vertex in $[t+1 : n]$, has the same contribution to both terms. The same argument holds for an edge involving vertices with indices strictly larger than t. As such, only the edges adjacent to t have a different contribution, which is as stated. edited everything from here to the end of this proof Rearranging, we have by Lemma 6.6 with $C = N(t)$ and $u = \deg(t)$, with probability $\geqslant 1 - \beta$ for $\beta = 4/n^c$ for any constant $c > 1$, that

$$\Delta_t = \left|C^{\pi(i)}_{<t} + \mathbb{E}\left[C^{\pi(i)}_{>t}\right] - C^{\pi(j)}_{<t} - \mathbb{E}\left[C^{\pi(j)}_{>t}\right]\right| \leqslant \left|C^{\pi(i)}_{<t} - C^{\pi(j)}_{<t}\right| + \mathbb{E}\left[\left|C^{\pi(i)}_{>t} - C^{\pi(j)}_{>t}\right|\right] \leqslant c'_t$$

for $c'_t = \sqrt{2c \deg(t) \ln n} + 12\sqrt{\deg(t)}/2$, which is at most $12c\sqrt{\deg(t) \ln n}$. Let $\mathcal{B}$ be the event that any Δ_t (for any choice of i or j) exceeds c_t, and observe that we can choose a constant $c > 1$ such that $\mathbb{P}[\mathcal{B}] \leqslant (2k)^2 n\beta \leqslant 1/n^{10}$, and $\mathsf{m} \cdot \mathbb{P}[\mathcal{B}] \leqslant 1/n^8$. To apply Lemma

Lemma 2.3, we have to bound

$$\max_{i,j}\left\{\left|\mathbb{E}\left[f \;\middle|\; X_1,\ldots,X_{t-1},X_t=i,\bar{\mathcal{B}}\right]-\mathbb{E}\left[f \;\middle|\; X_1,\ldots,X_{t-1},X_t=j,\bar{\mathcal{B}}\right]\right|\right\},$$

for every t. Since,

$$\mathbb{E}[f \mid X_1,\ldots,X_{t-1},X_t=i]-\mathsf{m}\cdot\mathbb{P}[\mathcal{B}] \leqslant \mathbb{E}\left[f \;\middle|\; X_1,\ldots,X_{t-1},X_t=i,\bar{\mathcal{B}}\right] \leqslant \frac{\mathbb{E}[f \mid X_1,\ldots,X_{t-1},X_t=i]}{1-\mathbb{P}[\mathcal{B}]},$$

we can conclude that

$$\max_{i,j}\left\{\left|\mathbb{E}\left[f \;\middle|\; X_1,\ldots,X_{t-1},X_t=i,\bar{\mathcal{B}}\right]-\mathbb{E}\left[f \;\middle|\; X_1,\ldots,X_{t-1},X_t=j,\bar{\mathcal{B}}\right]\right|\right\} \leqslant c_t,$$

for $c_t = 12c\sqrt{\deg(t)\ln n}+\frac{2}{n^8}$. We have $S=\sum_{t=1}^{n} c_t^2$, which is $\sum_{t=1}^{n} 144c^2\deg(t)\ln n+\frac{4}{n^{16}}+48c\sqrt{\deg(t)\ln n}/n = O(\mathsf{m}\ln n)$.

By Lemma 2.3, and setting $s=\varsigma\sqrt{\mathsf{m}}\ln n$, for some constant ς, we have

$$\mathbb{P}\left[\left|f-\frac{\mathsf{m}}{k}\right|>s+\mathsf{m}\cdot\mathbb{P}[\mathcal{B}]\right] \leqslant \exp\left(-2s^2/S\right)+\mathbb{P}[\mathcal{B}] \leqslant 1/n^8+\mathbb{P}[\mathcal{B}] \leqslant 2/n^8,$$

for ς sufficiently large.

The second statement in part (A) follows by a similar argument with $f(X_1,\ldots,X_n)=\sum_{i=1}^{2k}\mathsf{m}(U_i)$. We omit the easy but tedious details.

We now outline the proof of the claim in part (B). Let $\alpha=|U|$ and $\beta=|V|$, and let $U=\{u_1,u_2,\cdots,u_\alpha\}$ and $V=\{v_1,v_2,\cdots,v_\beta\}$. The vertex sequence is $u_1,u_2,\cdots,u_\alpha,v_1,v_2,\cdots,v_\beta$, and $X_i\in[k]$ is the color of the i'th vertex in this sequence, for $i\in[|U|+|V|]$ that is chosen uniformly at random. The rest of the proof follows analogously as in part (A) with $f(X_1,\cdots,X_n)=\sum_{i=1}^{k}\mathsf{m}(U_i,V_i)$. We omit the rest of the details. □

Remark 7. Given an induced bipartite graph $G(U,V)$ with m edges, coloring it with k colors, and taking the bipartite subgraphs of the resulting matching of the coloring, as done in Lemma 6.7, results in k new disjoint bipartite (induced) subgraphs $G_i=G(U_i,V_i)$. for $i=1,\ldots,k$, with total number of edges $\Gamma=\sum_{i=1}^{k}\mathsf{m}(G_i)$. Furthermore, we have that $k\cdot\Gamma$ is an $(1\pm\xi)$-approximation to $\mathsf{m}(G)$, where $\xi=\left(\varsigma k\sqrt{\mathsf{m}}\log n\right)/\mathsf{m}$, with high probability. For our purposes, we need

$$\xi\leqslant\frac{\varepsilon}{8\log n} \iff \frac{\left(\varsigma k\sqrt{\mathsf{m}}\log n\right)}{\mathsf{m}}\leqslant\frac{\varepsilon}{8\log n} \iff \frac{8\left(\varsigma k\log^2 n\right)}{\varepsilon}\leqslant\sqrt{\mathsf{m}} \iff \mathsf{m}=\Omega(k^2\varepsilon^{-2}\log^4 n).$$

Setting $k = 4$, the above implies that one can apply the refinement algorithm of Lemma 6.7 if $\mathsf{m} = \Omega(\varepsilon^{-2} \log^4 n)$. With high probability, the number of edges in the new k subgraphs (i.e., Γ), scaled by k, is a good estimate (i.e., within a $1 \pm \varepsilon/(8 \log n)$ factor) for the number of edges in the original graph, and furthermore, the number of edges in the new subgraphs is small (formally, $\mathbb{E}[\Gamma] \leqslant \mathsf{m}/4$, and with high probability $\Gamma \leqslant \mathsf{m}/2$).

6.4 *Edge estimation using* BIS *queries*

Here we show how to get exact and approximate count for the number of edges in a graph using BIS queries.

6.4.1 Exactly counting edges using BIS queries

Lemma 6.8. *Given two disjoint sets $U, V \subseteq [n]$, one can (deterministically) compute $E(U, V)$, and thus $\mathsf{m}(U, V) = |E(U, V)|$, using $O(1 + \mathsf{m}(U, V) \log n)$ BIS queries. Alternatively, given a query budget $t = \Omega(\log n)$, one can decide if the given graph has $\leqslant t/\log n$ edges (or more) using $O(t)$ BIS queries.*

Proof. We use a recursive divide-and-conquer approach, which intuitively builds a quadtree over the pair (U, V). The quadtree construction can also be interpreted in terms of the incidence matrix of the edges $E(U, V)$.

The algorithm first issues the query $\mathsf{BIS}(U, V)$. If the return value is false, then there are no edges between U and V, and the algorithm sets $\mathsf{m}(U, V)$ to zero, and returns. If $|U| = |V| = 1$, then this also determines if $\mathsf{m}(U, V)$ is 0 or 1 in this case, and the algorithm returns. The remaining case, is that $\mathsf{m}(U, V) \neq 0$, and the algorithm recurses on the four children of (U, V), which will correspond to the pairs $(U_1, V_1), (U_1, V_2), (U_2, V_1)$, and (U_2, V_2), where U_1, U_2 and V_1, V_2 are equipartitions of U and V, respectively. We are using here the identity

$$\mathsf{m}(U, V) = \mathsf{m}(U_1, V_1) + \mathsf{m}(U_1, V_2) + \mathsf{m}(U_2, V_1) + \mathsf{m}(U_2, V_2).$$

If $\mathsf{m}(U, V) = 0$ holds, then the number of queries is exactly equal to 1, and the lemma

is true in this case. For the rest of the proof we assume that $\mathsf{m}(U,V) \geqslant 1$. To bound the number of queries, imagine building the whole quadtree for the adjacency matrix of $U \times V$ with entries for $E(U,V)$. Let X be the set of 1 entries in this matrix, and let $k = |X|$ (i.e., X corresponds to set of leaves that are labeled 1 in the quadtree). The height of the quadtree is $h = O(\max\{\log|U|, \log|V|\})$. Let X_1 be the set of nodes in the quadtree that are either in X or are ancestors of nodes of X. It is not hard to verify that $|X_1| = O\big(k + k\log(|U||V|)\big) = O(k \log n)$. Finally, let X_2 be the set of nodes in the quadtree that are either in X_1, or their parent is in X_1. Clearly, the algorithm visits only the nodes of X_2 in the recursion, thus implying the desired bound.

As for the budgeted version, run the algorithm until it has accumulated $T = O(t/\log n)$ edges in the working set, where $T > t$. If this never happens, then the number of edges of the graph is at most T, as desired, and the above analysis applies. Otherwise, the algorithm stops, and applying the same argument as above, we get that the number of BIS queries is bounded by $O(T \log n) = O(t)$. □

Remark 8. The number of BIS queries made by the algorithm of Lemma 6.8 is at least $\max\{\mathsf{m}(U,V), 1\}$, since every edge with one endpoint in U and the other in V is identified (on its own, explicitly) by such a query.

Though we do not need it in sequel for our algorithms to estimate the number of edges in a graph, we can use the above algorithm to exactly identify the edges of an *arbitrary* graph using BIS queries with a cost of $O(\log n)$ overhead per edge.

Lemma 6.9. *Given a vertex $v \in [n]$, one can compute all the edges adjacent to v in G using $O\big(1 + \deg(v) + \deg(v)\log(n/\deg(v))\big) = O(1 + \deg(v)\ \log n)$ queries.*

Proof. We build a binary tree $\mathcal{T}$ of BIS queries of the form $\mathsf{BIS}(\{v\}, U)$ to determine the entries of the row ρ of the adjacency matrix for G indexed by v. In particular, view the vertices of $B = [n] - \{v\}$ as an ordered set. The algorithm first queries $\mathsf{BIS}(\{v\}, B)$ which returns true iff there is a 1 in ρ. This query is the root of $\mathcal{T}$. For every node ν of $\mathcal{T}$ corresponding to

a query $\mathsf{BIS}(\{v\}, U)$ whose output is true and for which $|U| > 1$, there are two children ν' and ν'' labeled by queries $\mathsf{BIS}(\{v\}, U')$ and $\mathsf{BIS}(\{v\}, U")$ where U' and $U"$ are the first and second halves of U.

If the answer to the root query is false, then only one BIS query is used. Otherwise, consider the subtree of $\mathcal{T}$ labeled by BIS queries that evaluate to true. This subtree has $\deg(v)$ leaves at depth $\lceil \log n \rceil$ and $\deg(v) - 1$ nodes of degree two. The total number of BIS queries (nodes of $\mathcal{T}$) is then $2\deg(v) - 1$ plus twice the number of nodes of this subtree of degree one. If any node of degree one is at a level higher than some node of degree two then we can increase the count of degree one nodes by swapping the levels of those nodes. This implies that all nodes of the subtree at levels up to $\lfloor \log \deg(v) \rfloor$ have degree two in $\mathcal{T}$. Therefore, the number of BIS queries is at most $1 + 2\deg(v) + 2\deg(v) \cdot (\lceil \log n \rceil - \lfloor \log \deg(v) \rfloor)$ which is $O(1 + \deg(v) \cdot \log n)$. □

Lemma 6.10. *Given a vertex $v \in [n]$, and a graph $G = ([n], E)$, let $\mathcal{C}$ be the connected component of v in G. The set of edges in $\mathcal{C}$ (i.e., $E(\mathcal{C})$) can be computed using $O(1 + \mathsf{m}(\mathcal{C}) \log n)$ BIS queries, where $\mathsf{m}(\mathcal{C})$ is the number of edges in $\mathcal{C}$.*

Proof. Do a breadth-first search in G starting from v. Whenever reaching a vertex for the first time, compute its adjacent edges using Lemma 6.9. Clearly, the breadth-first search visits all the vertices in $\mathcal{C}$, and therefore computes all the edges in this connected component. The bound on the number of queries readily follows by observing that $\sum_{v \in V(\mathcal{C})} d(v) \log n$ is $O(\mathsf{m}(\mathcal{C}) \log n)$. □

Lemma 6.11. *For a graph $G = ([n], E)$, one can deterministically compute $\mathsf{m}(E)$ exactly, using at most $O(\log n + |E| \log n)$ BIS queries. Alternatively, given a query budget $t = \Omega(\log n)$, one can decide whether the given graph has at most $t/\log n$ edges, or more than this number, using $O(t)$ BIS queries.*

Proof. For $i = 1, \ldots, T = \lceil \log_2 n \rceil$, define A_i to be the elements of $[n]$ whose i^{th} bit in their binary representation is 1. Let $B_i = [n] \setminus A_i$. Let $W_1 = [n]$.

Iterate for $i = 1, \ldots, T$. In the i^{th} iteration, compute all the edges in $E_i = E(A_i \cap W_i, B_i \cap W_i)$ using the algorithm of Lemma 6.8. This requires $O(1 + |E_i| \log n)$ queries. Let $V_i = V(E_i)$. As long as there is an unvisited vertex $v \in V_i$, apply the algorithm from Lemma 6.10 to produce all edges in the connected component of G containing v. Repeat this process until all the vertices in V_i are visited. This computes a collection $\mathcal{C}_i = \bigcup_{v \in V_i} \mathcal{C}(v)$ of connected components in G, and the number of BIS queries used to compute it is $O(1 + \mathsf{m}(\mathcal{C}_i) \log n)$. The algorithm now sets $W_{i+1} = W_i \setminus V(\mathcal{C}_i)$, and continues to the next iteration. Finally, the algorithm outputs the set $\bigcup_i E(\mathcal{C}_i)$ (or just its size).

For correctness and runtime, observe that $E_i \subseteq E(\mathcal{C}_i)$, and these edges are no longer considered in later iterations of the algorithm, as we remove the vertices of $V(\mathcal{C}_i)$ from W_i to get W_{i+1}. As such, $E(\mathcal{C}_1), \ldots, E(\mathcal{C}_T)$ are disjoint sets and hence the total cost of the iterations is $O(T + \sum_{i=1}^{T} \mathsf{m}(\mathcal{C}_i) \log n) = O(\log n + |E| \log n)$ as required. We claim that $\bigcup_i E(\mathcal{C}_i) = E(G)$. To this end consider any edge $uv \in E(G)$. Assume that the first bit on which vertices u and v differ is their i^{th} bit. We have two cases: (1) If either u or v is not in W_i then there is some first iteration $j < i$ in which one of them was removed. But since they are in the same connected component of G both will be removed and the edge $uv \in E(\mathcal{C}_j)$. (2) If both u and v are in W_i, since they differ on their i^{th} bits, then $uv \in E_i \subseteq E(\mathcal{C}_i)$ as required.

As for the budgeted version – run the algorithm until $\tau = \Omega(t)$ BIS queries were performed. If this does not happen, then the graph has at most τ edges, and they were reported by the algorithm. Otherwise, we know that the graph must have at least $\tau / \log n$ edges, as desired. □

6.4.2 The Coarse Estimator algorithm

Let $G = G([n], E)$ be a graph and let $U, V \subseteq [n]$ be disjoint subsets of the vertices. The task at hand is to estimate $\mathsf{m}(U, V)$, using polylog BIS queries.

For a subset $S \subseteq [n]$, define $N(S)$ to be the union of the neighbors of all the vertices in S.

Algorithm 6.4.1: $\mathsf{CheckEstimate}(U, V, \widetilde{e})$

Input: $((U, V), \widetilde{e})$ where $U, V \subseteq [n]$ are disjoint and $\widetilde{e}$ is a (rough) guess for the value of $\mathsf{m}(U, V)$

for $i = 0, 1, \ldots, \log n$ **do**

Sample $U' \subseteq U$ by choosing each vertex in U with probability $\min(2^i/\widetilde{e}, 1)$.

Sample $V' \subseteq V$ by choosing each vertex of V with probability $1/2^i$.

if $\mathsf{m}(U', V') \neq 0$ **then**

Output **accept**;

Output **reject**.

For a vertex v, let $\deg_S(v)$ denote the number of neighbors of v that lie in S. For $i \in [\log n]$, define the set of vertices in U with degree between 2^i and 2^{i+1} as

$$U_i = \left\{ u \;\middle|\; u \in U,\ 2^i < \deg_V(u) \leqslant 2^{i+1} \right\},$$

and let U_0 denote the vertices in U with $\deg_V(v) \leqslant 2$.

Claim 6.12. *There exists an* $\alpha \in \{0, 1, \ldots, \log n\}$ *such that*

$$\mathsf{m}(U_\alpha, V) \geqslant \frac{\mathsf{m}(U, V)}{\log n + 1} \qquad \textit{and} \qquad |U_\alpha| \geqslant \frac{\mathsf{m}(U, V)}{2^{\alpha+1}(\log n + 1)}.$$

Proof. Since $\sum_{i=0}^{\log n} \mathsf{m}(U_i, V) = \mathsf{m}(U, V)$, the first inequality is stating that there is a term as large as the average. As for the second inequality, observe that for every i, we have $|U_i|2^i \leqslant \mathsf{m}(U_i, V) \leqslant |U_i|2^{i+1}$. Hence, using the first inequality $|U_\alpha| \geqslant \dfrac{\mathsf{m}(U_\alpha, V)}{2^{\alpha+1}} \geqslant \dfrac{\mathsf{m}(U, V)}{2^\alpha} \cdot \dfrac{1}{2(\log n + 1)}$. □

Suppose that we have an estimate $\widetilde{e}$ for the number of edges between U and V in the graph. Consider the test $\mathsf{CheckEstimate}$, depicted in Algorithm 6.4.1, for checking if the estimate $\widetilde{e}$ is correct up to polylogarithmic factors using a logarithmic number of BIS queries.

Claim 6.13. *Let $n \geqslant 16$. If $\mathsf{m}(U,V) > 0$, then*

1. *if $\widetilde{e} \geqslant 4\mathsf{m}(U,V)(\log n + 1)$, then* CheckEstimate$(U,V,\widetilde{e})$ *accepts with probability at most $1/4$.*

2. *if $\widetilde{e} \leqslant \frac{\mathsf{m}(U,V)}{4\log n}$, then* CheckEstimate$(U,V,\widetilde{e})$ *accepts with probability at least $1/2$.*

Proof. (A) For any value of the loop variable i, the probability that a fixed edge is present in the induced subgraph on U' and V' is $\min(2^i/\widetilde{e}, 1) \cdot (1/2^i) \leqslant 1/\widetilde{e}$. Thus, $\mathbb{E}[\mathsf{m}(U',V')] \leqslant \mathsf{m}(U,V)/\widetilde{e} \leqslant \frac{1}{4(\log n+1)}$. For a fixed iteration i, by Markov's inequality, we have

$$\mathbb{P}[\mathsf{m}(U',V') \neq 0] = \mathbb{P}[\mathsf{m}(U',V') \geqslant 1] \leqslant \mathbb{E}[\mathsf{m}(U',V')] \leqslant \frac{1}{4(\log n+1)}.$$

By a union bound over the loop variable values, the probability that the test accepts is at most $1/4$.

(B) It is enough to show that the probability is at least $1/2$ when the loop variable attains the value α given by Claim 6.12. In this case, we have that $|U_\alpha| \geqslant \frac{\mathsf{m}(U,V)}{2^{\alpha+1}(\log n+1)}$, and thus

$$\begin{aligned}\mathbb{P}[U' \cap U_\alpha = \varnothing] = \left(1 - \frac{2^\alpha}{\widetilde{e}}\right)^{|U_\alpha|} &\leqslant \exp\left(-\frac{2^\alpha}{\widetilde{e}} \cdot |U_\alpha|\right) \leqslant \exp\left(-\frac{2^\alpha}{\mathsf{m}(U,V)/(4\log n)} \cdot \frac{\mathsf{m}(U,V)}{2^{\alpha+1}(\log n+1)}\right) \\ &\leqslant \exp\left(-\frac{4\log n}{2(\log n+1)}\right) \leqslant \frac{1}{e^{1.6}},\end{aligned}$$

since $n \geqslant 16$. Furthermore, since $\deg_V(u) \geqslant 2^\alpha$ for all $u \in U_\alpha$, it follows that when $U' \cap U_\alpha \neq \varnothing$, then $|N(U' \cap U_\alpha)| \geqslant 2^\alpha$. So, we can bound

$$\mathbb{P}\Big[V' \cap N(U' \cap U_\alpha) = \varnothing \;\Big|\; U' \cap U_\alpha \neq \varnothing\Big] \leqslant \left(1 - \frac{1}{2^\alpha}\right)^{2^\alpha} \leqslant \frac{1}{e}.$$

From the above, we get

$$\mathbb{P}[\mathsf{m}(U',V') \neq 0] = \mathbb{P}\Big[U' \cap U_\alpha \neq \varnothing\Big] \cdot \mathbb{P}\Big[V' \cap N(U' \cap U_\alpha) \neq \varnothing \;\Big|\; U' \cap U_\alpha \neq \varnothing\Big] \geqslant \left(1 - \frac{1}{e^{1.6}}\right)\left(1 - \frac{1}{e}\right) \geqslant \frac{1}{2}$$

□

Algorithm 6.4.2: $\mathsf{CoarseEstimator}(U, V)$

Input: (U, V) where $U, V \subseteq [n]$ are disjoint

Output: An estimate $\widetilde{e}$ for the number of edges $\mathsf{m}(U, V)$ computed using BIS queries

if $\mathsf{m}(U, V) = 0$ **then** Output 0;

for $j = 2 \log n, \ldots, 0$ **do**

Run $t := 128 \log n$ independent trials of $\mathsf{CheckEstimate}(U, V, 2^j)$.

if *at least* $3t/8$ *of them output* ***accept*** **then**

Output 2^j;

Armed with the above test, we can easily estimate the number of edges up to a $O(\log n)$ factor by doing a search, where we start with $\widetilde{e} = n^2$ and halve the number of edges each iteration. The algorithm is depicted in Algorithm 6.4.2.

Claim 6.14. *For $n \geqslant 16$, $\mathsf{CoarseEstimator}(U, V)$ outputs $\widetilde{e} \leqslant n^2$ satisfying $\frac{\mathsf{m}(U,V)}{8 \log n} \leqslant \widetilde{e} \leqslant 8\mathsf{m}(U, V) \log n$, with probability at least $1 - 4n^{-4} \log n$. The number of BIS queries made is $c_{\mathsf{ce}} \log^3 n$ for some constant c_{ce}.*

Proof. For any fixed value of the loop variable j such that $2^j \geqslant 4\mathsf{m}(U, V)(\log n + 1)$, the expected number of accepts is at most $t/4$ using Claim 6.13 1, where $t = 128 \log n$. The probability that we see at least $3t/8 = t/4 + t/8$ accepts is bounded by $\exp(-2(t/8)^2/t) = \exp(-t/32) \leqslant n^{-4}$ by Chernoff's inequality (Lemma 2.2 1). Taking the union over all values of j, the probability that the algorithm returns 2^j, when $2^j \geqslant 4\mathsf{m}(U, V)(\log n + 1)$, is at most $2n^{-4} \log n$.

On the other hand, when $2^j \leqslant \mathsf{m}(U, V)/(4 \log n)$, the expected number of accepts is at least $t/2$, by Claim 6.13 2, and so the probability that we see at least $3t/8 = t/2 - t/8$ accepts is at least $1 - \exp(-2t/8^2) \geqslant 1 - n^{-4}$ by Chernoff's inequality (Lemma 2.2 1). Hence, conditioned on the event that the algorithm has not already returned a bigger value of j, the probability that we accept for the unique j that satisfies $\mathsf{m}(U, V)/8 \leqslant 2^j \log n < \mathsf{m}(U, V)/4$, is at least $1 - n^{-4}$.

Overall, by a union bound, the probability that the estimator outputs an estimate $\widetilde{e}$ that does not satisfy $(8\log n)^{-1} \leqslant \widetilde{e}/\mathsf{m}(U,V) \leqslant 8\log n$ is at most $4n^{-4}\log n$. The number of BIS queries is bounded by $O\left(\log^3 n\right)$, since for each value of j there are $t = 128\log n$ trials of $\mathsf{CheckEstimate}$, each of which makes $\log n + 1$ queries to the BIS oracle. □

Summarizing the above, we get the following result

Lemma 6.15. *For $n \geqslant 16$, and arbitrary $U, V \subseteq [n]$ that are disjoint, the randomized algorithm $\mathsf{CoarseEstimator}(U,V)$ makes at most $c_{\mathsf{ce}}\log^3 n$ BIS queries (for a constant c_{ce}) and outputs $\widetilde{e} \leqslant n^2$ such that, with probability at least $1 - 4n^{-4}\log n$, we have $(8\log n)^{-1} \leqslant \widetilde{e}/\mathsf{m}(U,V) \leqslant 8\log n$.*

6.4.3 The overall BIS approximation algorithm

Given a graph $G = ([n], E)$, we describe here an algorithm that makes $\mathrm{polylog}(n)/\varepsilon^4$ BIS queries to estimate the number of edges in the graph within a factor of $(1 \pm \varepsilon)$.

The algorithm for estimating the number of edges in the graph is going to maintain a data-structure $\mathcal{D}$ containing:

1. An accumulator φ - this is a counter that maintains an estimate of the number of edges already handled.

2. A list of triples $(U_1, V_1, \mathsf{w}_1), \ldots, (U_u, V_u, \mathsf{w}_u)$ where $U_i, V_i \subseteq [n]$ and $\mathsf{w}_1 > 1$ is a non-negative weight.

The *estimate* based on $\mathcal{D}$ of the number of edges in the original graph $G = ([n], E)$ is

$$\mathsf{m}(\mathcal{D}) = \varphi + \sum_i \mathsf{w}_i \cdot \mathsf{m}(U_i, V_i).$$

The number of *active* edges in $\mathcal{D}$ is $\mathsf{m}_{\mathrm{active}}(\mathcal{D}) = \sum_i \mathsf{m}(U_i, V_i)$.

Cleanup, refine, and reduce

The algorithm is going to use three subroutines: cleanup, refine, and reduce, described next.

1. Cleanup: The cleanup stage removes from $\mathcal{D}$ all induced subgraphs that have few edges, by explicitly counting the number of their edges. Let

$$L_{\text{small}} = \Theta(\varepsilon^{-2} \log^4 n) \tag{6.3}$$

as specified by Remark 7. Given the data-structure $\mathcal{D}$, the algorithm scans the list of triples $(U, V, \mathsf{w}) \in \mathcal{D}$. For each triple (U, V, w), using the algorithm of Lemma 6.8, it decides if $\mathsf{m}(U, V) \leqslant 2L_{\text{small}}$. If so, the value of $\mathsf{m}(U, V)$ was just computed, and it adds $\mathsf{w} \cdot \mathsf{m}(U, V)$ to φ. Finally, it removes this triple from $\mathcal{D}$.

If $\mathcal{D}$ has no triples in it, then the algorithm returns φ as the desired approximation.

2. Refine: We are given the data-structure $\mathcal{D}$, where the graph associated with every triple has at least L_{small} edges. The algorithm replaces every triple $(U, V, \mathsf{w}) \in \mathcal{D}$ by the four induced subgraphs resulting from 4-coloring the graph $G(U, V)$, as described by Lemma 6.7(B) (see also Remark 7). Specifically, the coloring results in the pairs (U_i, V_i), for $i = 1, 2, 3, 4$. The triple (U, V, w) is replaced in $\mathcal{D}$ by the triples $\{(U_1, V_1, 4\mathsf{w}), \ldots, (U_4, V_4, 4\mathsf{w})\}$. This increases the number of triples in $\mathcal{D}$ by a factor of four.

3. Reduce: If $\mathcal{D}$ has more than $2L_{\text{len}}$ triples, where $L_{\text{len}} = O(\varepsilon^{-2} \log^8 n)$ as specified by Remark 6, then the algorithm reduces the number of triples.

To this end, the algorithm first computes for each triple $(U, V, \mathsf{w}) \in \mathcal{D}$, a coarse estimate $\tilde{e}$ of the number of edges in $\mathsf{m}(U, V)$, such that $\mathsf{m}(U, V)/(8 \log n) \leqslant \tilde{e} \leqslant \mathsf{m}(U, V)\, 8 \log n$, by using the algorithm of Claim 6.14. This requires $O(\log^3 n)$ BIS queries per triple.

Next, the algorithm uses the summation reduction algorithm of Lemma 6.2 applied to the list of triples in $\mathcal{D}$, with $\xi = \varepsilon/(8 \log n)$. This reduces the number of triples in $\mathcal{D}$ to be at most L_{len}, while introducing a multiplicative error of $(1 \pm \xi)$.

The algorithm in detail

The algorithm input is the graph $G = ([n], E)$, and a parameter $\varepsilon > 0$. Let $\mathcal{R} = O\left(\varepsilon^{-4} \log^{14} n\right)$ be some parameter. The algorithm works as follows.

1. Check if G has at most $O(\mathcal{R}/\log^2 n)$ edges, using the algorithm of Lemma 6.11, which requires $O(\mathcal{R})$ BIS queries. If so, the algorithm returns the exact number of edges in G, and stops.

2. Compute a random 2-coloring of the vertices of the graph, creating two sets $U \cup V = [n]$, see Lemma 6.7 (A). We now create a data-structure as described above, with $\mathcal{D} = [\varphi, (U, V, 1)]$, where φ is initialized to value 0.

3. As long as $\mathcal{D}$ contains some triple the algorithm does the following:

   (a) Performs Cleanup on $\mathcal{D}$, as described in Section 6.4.3 1.

   (b) Performs Refine on $\mathcal{D}$, as described in Section 6.4.3 2.

   (c) Performs Reduce on $\mathcal{D}$, as described in Section 6.4.3 3.

4. The algorithm now returns the value φ as the desired approximation.

Analysis

Number of iterations. Initially, the number of active edges is at most m. Every time Refine is executed, this number reduces by a factor of 2 with high probability using Lemma 6.7(B) (in expectation, the reduction is by a factor of 4). As such, after $\lceil \log \mathsf{m} \rceil \leqslant \lceil \log \binom{n}{2} \rceil \leqslant 2 \log n$ iterations there are no active edges, and then the algorithm terminates.

Number of BIS queries. Clearly, because Reduce is used on $\mathcal{D}$ in each iteration, the algorithm maintains the invariant that the number of triples in $\mathcal{D}$ is at most $O(L_{\text{len}})$, where $L_{\text{len}} = O(\varepsilon^{-2}\log^8 n)$ as specified by Remark 6.

The procedure Cleanup, applies the algorithm of Lemma 6.8, to decide whether a triple in the list has at $2L_{\text{small}}$ edges associated with it, or fewer than than number, where $L_{\text{small}} = \Theta(\varepsilon^{-2}\log^4 n)$ (see Eq. (6.3) and Remark 7). This takes $O(L_{\text{small}}\log n)$ BIS queries. Overall the Cleanup step performs is $O(L_{\text{small}}L_{\text{len}}\log n)$ queries in each iteration. The procedure Refine does not perform any BIS queries. The procedure Reduce, performs $O(L_{\text{len}}\log^3 n)$ BIS queries in the estimation stage.

As such, overall, the algorithm performs $O(L_{\text{small}}L_{\text{len}}\log n) = O\left(\varepsilon^{-2}\log^4 n \cdot \varepsilon^{-2}\log^8 n \cdot \log n\right) = O(\varepsilon^{-4}\log^{13} n)$ BIS queries per iteration. There are $O(\log n)$ iterations, and as such, the overall number of BIS queries is $\mathcal{R} = O(\varepsilon^{-4}\log^{14} n)$, which also bounds the number of BIS queries in the first step of the algorithm.

Approximation error. The initial 2-coloring of the graph, in 2, introduces a $(1 \pm \varepsilon_0)$-multiplicative error, by Lemma 6.7, where

$$\varepsilon_0 = O(\sqrt{1/\mathsf{m}}\log n) \ll \xi = \frac{\varepsilon}{8\log n}.$$

Inside each iteration, Cleanup introduces no error. By the choice of parameters, Refine introduces a multiplicative error that is at most $1 \pm \xi$; see Remark 7. Similarly, Reduce introduces a multiplicative error bounded by $1 \pm \xi$; see Remark 6. As such, the multiplicative approximation of the algorithms lies in the interval

$$[(1-\varepsilon_0)(1-\xi)^{2\log n}, (1+\varepsilon_0)(1-\xi)^{2\log n}] \subseteq [1-\varepsilon, 1+\varepsilon],$$

since $(1-\varepsilon/(8\log n))^{1+2\log n} \geqslant 1-\varepsilon$ and $(1+\varepsilon/(8\log n))^{1+2\log n} \leqslant 1+\varepsilon$ as easy calculations show.

Probability of success. Throughout this analysis, c will be a constant that can be chosen to be arbitrarily large. The algorithm may fail due to the following reasons: (i) the random

two-coloring in Step (B) gives an estimate that is far from its expectation − this probability is at most $1/n^c$ using Lemma 6.7(A); (ii) the Refine step fails − the probability for the failure of each iteration is at most $1/n^c$ using Lemma 6.7(B) ; (iii) the coarse estimate in Reduce step fails − the probability for the failure of each iteration is at most $1/n^c$ using Claim 6.14; and lastly (iv) the summation reduction in the Reduce step fails − the probability for the failure of each iteration is at most $1/n^c$ using Lemma 6.2. Overall, every step performed by the algorithm had probability at most $1/n^c$ to fail. The algorithm performs $O(\text{polylog}(n))$ steps with high probability, which implies that the algorithm succeeds with probability at least $1 - 1/n^{O(1)}$.

The overall BIS result

Theorem 6.16. *Let $G = ([n], E)$ be an undirected graph. For a parameter $\varepsilon \in (0,1)$, one can compute an estimate $\widetilde{\mathsf{m}}$ for the number of edges in G, such that $(1-\varepsilon)\mathsf{m}(G) \leqslant \widetilde{\mathsf{m}} \leqslant (1+\varepsilon)\mathsf{m}(G)$, where $\mathsf{m}(G)$ is the number of edges of G. The algorithm performs $O(\varepsilon^{-4}\log^{14} n)$ BIS queries and succeeds with probability $\geqslant 1 - 1/n^{O(1)}$.*

6.4.4 Degree estimation using BIS queries

We provide an auxiliary degree estimation result, connecting BIS queries to local queries (e.g., [70, 75]).

Lemma 6.17. *Given a graph $G = ([n], E)$, a parameter $\varepsilon \in (0,1)$, and a vertex $v \in [n]$, one can $(1 \pm \varepsilon)$-approximate $\deg(v)$ in G using $O(\varepsilon^{-2}\log n)$ BIS queries. The approximation is correct with high probability.*

Proof. Let $N(v) = \{i \mid vi \in E\}$ be the set of neighbors of v, and let $E_v = \{vi \mid vi \in E\}$ be the corresponding set of edges. We have $\deg(v) = |N(v)| = |E_v|$. Given a set of edges $E_Q \subseteq E_v = \{vi \mid i \in [n]\}$, the corresponding set of vertices is $Q = \{i \mid vi \in E_Q\}$. In particular, $Q \cap N(v) \neq \varnothing \iff E_Q \cap E_v \neq \varnothing$. Deciding if $E_Q \cap E_v \neq \varnothing$ is equivalent to deciding if any of the edges adjacent to v is in E_Q, and this is answered by the BIS query for $(\{v\}, Q)$.

Namely, the BIS oracle can function as an emptiness oracle for $N(v) \subseteq [n]$. Now, using the algorithm of Lemma 6.5 we can $(1 \pm \varepsilon)$-approximation $|N(v)|$ using $O(\varepsilon^{-2} \log n)$ queries, as claimed. □

6.5 Edge estimation using IS queries

This section describes and analyzes our IS query algorithm (Theorem 6.25). At the end, we also discuss limitations of IS queries, suggesting that IS queries may indeed be weaker than BIS queries.

6.5.1 Exactly counting edges using IS queries

We start with an exact edge counting algorithm for IS queries. At a high-level, we use Lemma 6.8 after efficiently computing a suitable decomposition of our graph.

Lemma 6.18. *Given disjoint sets of vertices $U, V \subseteq [n]$, such that both U and V are independent sets, one can compute the number of edges $\mathsf{m}(U \cup V)$ using $O(\mathsf{m}(U \cup V) \log n)$ IS queries, assuming $\mathsf{m}(U, V) > 0$.*

Proof. Since U and V are disjoint and independent, we have that $\mathsf{m}(U \cup V) = \mathsf{m}(U, V)$. Furthermore, for any $U' \subseteq U$ and $V' \subseteq V$, the query $\mathsf{BIS}(U', V')$ is equivalent to the query $\mathsf{IS}(U' \cup V')$. As such, we can use the algorithm of Lemma 6.8, using the IS queries as a replacement for the BIS queries, yielding the result. □

The next step is to break the set of interest U into independent sets.

Lemma 6.19. *Given a set $U \subseteq [n]$, one can decompose it into disjoint independent sets $V_1, V_2, \ldots, V_t$, such that*

1. *$U = \bigcup_{i=1}^{t} V_i$, and*
2. *for any $i, j \in [t]$, with $i < j$, we have $\mathsf{m}(V_i, V_j) > 0$.*

Furthermore, computing this decomposition uses only $O(1 + \mathsf{m}(U) \log n)$ IS queries.

Proof. Order the elements of $U = \{u_1, \ldots, u_k\}$ in an arbitrary order. Compute the largest $i \in [k]$, such that $u_1, \ldots, u_i$ is an independent set. Using binary search, this can be done using $O(\log n)$ IS queries, and let U_1 be this first set. Continue decomposing $U \setminus U_1$ in the same fashion. This results in a decomposition of U into disjoint independent sets $U_1, U_2, \ldots, U_\sigma$, and requires $O(\sigma \log n)$ queries. Observe however, that for every i, the set U_i in this decomposition has at least one edge between one of its vertices and the first vertex that was not added yet to any set. As such, we can charge the $O(\log n)$ IS queries used in computing U_i to this guaranteed edge, implying that this stage used $O(\mathsf{m}(U) \log n)$ queries.

In the second stage of the algorithm, we group these independent sets together. In the τ^{th} iteration, for $\tau = 1, \ldots, \sigma$, the algorithm does the following:

1. Assume that the algorithm already computed the independent sets $V_1, \ldots, V_{f(\tau)}$ (initially, $f(1) = 0$).

2. For $i = 1, \ldots, f(\tau)$, check using IS query, if the set $V_i \cup U_\tau$ is an independent set. If it is, then set $f(\tau + 1) = f(\tau)$, and $V_i \leftarrow V_i \cup U_\tau$. The algorithm then continues to the next (outer) iteration.

3. Otherwise, set $f(\tau + 1) = f(\tau) + 1$, and set $V_{f(\tau+1)} = U_\tau$.

Clearly, the resulting decomposition $V_1, \ldots, V_{f(\sigma)}$ of U has the desired properties. As for the number of IS queries, observe that every time that U_τ get rejected, in the i^{th} iteration of the inner loop, this is because of an edge present in the set $E(V_i, U_\tau)$. We charge the IS query to such an edge. An edge get charged at most once by this process. We conclude that the algorithm performs at most $O(\mathsf{m}(U))$ queries (for this part). □

Lemma 6.20. *Given $U \subseteq [n]$, one can deterministically compute $E(U)$, using $O(1 + \mathsf{m}(U) \log n)$ IS queries. Alternatively, given a budget $t > 0$ and set $U \subseteq [n]$, one can decide if $\mathsf{m}(U) > t$ using $O(t \log n)$ IS queries.*

Proof. Using the algorithm of Lemma 6.19, compute the decomposition of U into independent sets $V_1, \ldots, V_t$. By construction, for any $i < j$, we have that $\mathsf{m}(V_i, V_j) \geqslant 1$, as some vertex of V_i is connected to some vertex in V_i. As such, going over all $1 \leqslant i < j \leqslant t$, compute the set of edges $E(V_i, V_j)$ using the algorithm of Lemma 6.18.

This requires $O(\mathsf{m}(V_i, V_j) \log n)$ IS queries. As such, the total number of IS queries used by this algorithm is $O\big(\mathsf{m}(U) \log n + \sum_{i<j} \mathsf{m}(V_i, V_j) \log n\big) = O(\mathsf{m}(U) \log n)$. □

6.5.2 Algorithms for edge estimation using IS queries

Our IS algorithm has two main subroutines. We first describe and analyze these, then we combine them for the overall algorithm, which is presented in Theorem 6.25.

Growing Search

The following is an immediate consequence of Lemma 6.20.

Lemma 6.21. *Let*

$$L_{\text{base}} = \lceil c_1 \varepsilon^{-4} \log^4 n \rceil,$$

where c_1 is some sufficiently large constant. Given a set U, one can decide if $\mathsf{m}(U) \leqslant L_{\text{base}}$, and if so get the exact value of $\mathsf{m}(U)$, using $O(\varepsilon^{-4} \log^5 n)$ IS queries.

Lemma 6.22. *Given parameters t, $\varepsilon \in (0, 1]$, and a set $U \subseteq [n]$, such that $\mathsf{m}(U) \geqslant \max(L_{\text{base}}, t^2)$, an algorithm can decide if $\mathsf{m}(U) > 2t^2$, or alternatively return a $(1 \pm \varepsilon)$-approximation to $\mathsf{m}(U)$ if $t^2 \leqslant \mathsf{m}(U) \leqslant 2t^2$. The algorithm uses $O(\varepsilon^{-1} t \log^2 n)$ IS queries and succeed with probability $1 - 1/n^{O(1)}$.*

Proof. We color the vertices in U randomly using $k = \lceil t\varepsilon/(\varsigma \log n) \rceil$ colors for a constant ς to be specified shortly, and let $U_1, \ldots, U_k$ be the resulting partition. By Lemma 6.7, we have for the estimate $\Gamma = \sum_{i=1}^{k} \mathsf{m}(U_i)$ that

$$|\mathsf{m}(U) - k \cdot \Gamma| \leqslant \varsigma k \sqrt{\mathsf{m}(U)} \log n,$$

and this holds with probability $\geqslant 1 - n^{-c_3}$, where c_3 is an arbitrarily large constant, and ς is a constant that depends only on c_3. For this to be an $(1 \pm \varepsilon)$-approximation, we need that

$$\frac{\varsigma k \sqrt{\mathsf{m}(U)} \log n}{\mathsf{m}(U)} \leqslant \varepsilon \iff \mathsf{m}(U) \geqslant \left(\frac{\varsigma k \log n}{\varepsilon}\right)^2 = t^2,$$

which holds because of the assumption $\mathsf{m}(U) \geqslant \max\{L_{\text{base}}, t^2\}$ in the statement.

To proceed, the algorithm starts computing the terms in the summation defining Γ, using the algorithm of Lemma 6.20. If at any point in time, the summation exceeds $M = 8(t^2/k) = O(\varepsilon^{-1} t \log n)$, then the algorithm stops and reports that $\mathsf{m}(U) > 2t^2$. Otherwise, the algorithm returns the computed count $k \cdot \Gamma$ as the desired approximation. In both cases we are correct with high probability by Lemma 6.7.

We now bound the number of IS queries. If the algorithm computed Γ by determining exact edge counts for $m(U_i)$ for all $i \in [k]$, then the number of queries would be $\sum_{i=1}^{k} O\left(1 + \mathsf{m}(U_i) \log n\right)$. However, the choice of stopping early if the number of queries exceeds $M = O(\varepsilon^{-1} t \log n)$ implies that total the number of queries is bounded by $O(k + M \log n) = O\left(\varepsilon^{-1} t \log^2 n\right)$. □

Lemma 6.23. *Given $\varepsilon \in (0, 1]$, and a set $U \subseteq [n]$, one can compute a $(1 \pm \varepsilon)$-approximation for $\mathsf{m}(U)$. The algorithm uses at most $O(\varepsilon^{-4} \log^5 n + \varepsilon^{-1} \sqrt{\mathsf{m}(U)} \log^2 n)$ IS queries and succeeds with probability $1 - 1/n^{O(1)}$.*

Proof. The algorithm starts by checking if the number of edges in $\mathsf{m}(U)$ is $L_{\text{base}} = O(\varepsilon^{-4} \log^4 n)$ using the algorithm of Lemma 6.21. Otherwise, in the i^{th} iteration, the algorithm sets $t_i = \sqrt{2} t_{i-1}$, where $t_0 = \sqrt{L_{\text{base}}}$, and invokes the algorithm of Lemma 6.22 for t_i as the threshold parameter. If the algorithm succeeds in approximating the right size we are done. Otherwise, we continue to the next iteration. Taking a union bound over the iterations, we have that the algorithm stops with high probability before $t_\alpha > 4\sqrt{\mathsf{m}(U)}$. Let α be the minimum value for which this is holds. The number of IS queries performed by the algorithm is $O(\sum_{i=1}^{\alpha} t_i \varepsilon^{-1} \log^2 n) = O(\varepsilon^{-1} \sqrt{\mathsf{m}(U)} \log^2 n)$, since this is a geometric sum. □

Shrinking Search

We are given a graph $G = ([n], E)$, and a set $U \subseteq [n]$. The task at hand is to approximate $\mathsf{m}(U)$. Let $\mathcal{N} = |U|$.

Given an oracle that can answers IS queries, we can decide if a specific edge uv exists in the set $E(U)$, by performing an IS query on $\{u, v\}$. We can treat such IS queries as membership oracle queries in the set E of edges in the graph, where the ground set is the set of all possible edges $Z = \binom{U}{2} = \{ij \mid i < j \text{ and } i, j \in U\}$, where $|Z| = \mathcal{N}(\mathcal{N}-1)/2$. Invoking the algorithm of Lemma 6.4 in this case, with $\gamma = 1/n^{O(1)}$, implies that one can $(1 \pm \varepsilon)$-approximate $\mathsf{m}(U)$ using $O((\mathcal{N}^2/\mathsf{m}(U))\varepsilon^{-2} \log n)$ IS queries. For our purposes, however, we need a budgeted version of this.

Lemma 6.24. *Given parameters $t > 0$, $\xi \in (0, 1]$, and a set $U \subseteq [n]$, with $\mathcal{N} = |U|$, then an algorithm can return either: (a) $\mathsf{m}(U) \leqslant \mathcal{N}^2/(2t)$, or (b) return $(1 \pm \xi)$-approximation to $\mathsf{m}(U)$. The algorithm uses $O(t \log n)$ IS queries in case (a), and $O(t\xi^{-2} \log n)$ in case (b). The returned result is correct with high probability.*

Proof. The idea is to use the sampling as done in Lemma 6.3, with $g = \mathcal{N}^2/(16t)$ and $\varepsilon = 1/2$ on the sets of edges $E(U) \subseteq \binom{U}{2}$. The sample R used is of size $O((\mathcal{N}^2/g) \log n) = O(t \log n)$, and we check for each one of the sampled edges if it is in the graph by using an IS query. If the returned estimate is at most $g/2$, then the algorithm returns that it is in case (a).

Otherwise, we invoke the algorithm of Lemma 6.3 again, with $\varepsilon = \xi$, to get the desired approximation, which is case (b). □

The overall IS Search algorithm

Theorem 6.25. *Given a graph $G = ([n], E)$, with access to the edges of the graph via an IS oracle. Then, one can $(1 \pm \varepsilon)$-approximate the number of edges in G, denoted by $\mathsf{m} = |E|$. The algorithm uses*

$$O(\varepsilon^{-4} \log^5 n + \min(\sqrt{\mathsf{m}}, n^2/\mathsf{m})\varepsilon^{-2} \log^2 n)$$

IS queries, and it succeeds with high probability $1 - 1/n^{O(1)}$.

Proof. Let $t_0 = \lceil c_1 \varepsilon^{-4} \log^4 n \rceil$, for some constant c_1. Using the algorithm of Lemma 6.21, we can decide if $\mathsf{m} \leqslant t_0$, and if so, return (the just computed) m.

The algorithm now loops for $i = 1, 2, 3, \ldots,$ n. In the ith iteration, it does the following:

1. If $i = 1$ then let $t_1 = \sqrt{t_0}$, otherwise set $t_i = 2t_{i-1}$.

2. Using the algorithm of Lemma 6.22 decide if $\mathsf{m} \leqslant 2t_i^2$, and if so it returns the desired $(1 \pm \varepsilon)$-approximation to m. This uses $O(t_i \varepsilon^{-1} \log^2 n)$ IS queries.

3. Using the algorithm of Lemma 6.24, decide if $\mathsf{m} \leqslant n^2/(2t_i)$, and if so continue to the next iteration. This uses $O(t_i \log n)$ IS queries.

   Otherwise, the algorithm of Lemma 6.24 returned the desired $(1 \pm \varepsilon)$-approximation, using $O(t_i \varepsilon^{-2} \log n)$ IS queries.

Combining the two bounds on the IS queries, we get that the i^{th} iteration used $O(t_i \varepsilon^{-2} \log^2 n)$ IS queries.

The algorithm stopped in the i^{th} iteration, if $t_i \geqslant \sqrt{\mathsf{m}/2}$, or $t_i \geqslant n^2/\mathsf{m}$. In particular, for the stopping iteration I, we have $t_I = O(\min(\sqrt{\mathsf{m}}, n^2/\mathsf{m}))$. As such, the total number of IS queries in all iterations except the last one is bounded by $O(\sum_{i=1}^{I} t_i \varepsilon^{-2} \log^2 n) = O(t_I \varepsilon^{-2} \log^2 n)$. The stopping iteration uses $O(t_I \varepsilon^{-2} \log^2 n)$ IS queries. Each bound holds with high probability, and a union bound implies the same for the final result. □

Corollary 6.26. *For a graph $G = ([n], E)$, with an access to G via IS queries, and a parameter $\varepsilon > 0$, one can $(1 \pm \varepsilon)$-approximate m using $O(\varepsilon^{-4} \log^5 n + n^{2/3} \varepsilon^{-2} \log^2 n)$ IS queries.*

Proof. Follows readily as $\min(\sqrt{\mathsf{m}}, n^2/\mathsf{m}) \leqslant n^{2/3}$, for any value of m between 0 and n^2. □

6.5.3 Limitations of IS queries

In this section, we discuss several ways in which IS queries seem more restricted than BIS queries.

Simulating degree queries with IS queries. A degree query can be simulated by $O(\log n)$ BIS queries, see Lemma 6.17. In contrast, here we provide a graph instance where $\Omega\left(n/\deg\left(v\right)\right)$ IS queries are needed to simulate a degree query. In particular, we show that IS queries may be no better than edge existence queries for the task of degree estimation. Since it is easy to see that $\Omega\left(n/\deg\left(v\right)\right)$ edge existence queries are needed to estimate $\deg\left(v\right)$, this lower bound also applies to IS queries.

For the lower bound instance, consider a graph which is a clique along with a separate vertex v whose neighbors are a subset of the clique. We claim that IS queries involving v are essentially equivalent to edge existence queries. Any edge existence query can be simulated by an IS query. On the other hand, any IS query on the union of v and at least two clique vertices will always detect a clique edge. Thus, the only informative IS queries involve exactly two vertices.

Coarse estimator with IS queries. It is natural to wonder if it is possible to replace the coarse estimator (Lemma 6.15) with an analogous algorithm that makes $\mathrm{polylog}(n)$ IS queries. This would immediately imply an algorithm making $\mathrm{polylog}(n)/\varepsilon^4$ IS queries that estimates the number of edges. We do not know if this is possible, but one barrier is a graph consisting of a clique U on $O(\sqrt{\mathsf{m}})$ vertices along with a set V of $n - O(\sqrt{\mathsf{m}})$ isolated vertices. We claim that for this graph, the algorithm $\mathsf{CoarseEstimator}(U, V)$ from Section 6.4.2, using IS queries instead of BIS queries, will output an estimate $\widetilde{\mathsf{m}}$ that differs from m by a factor of $\Theta(n^{1/3})$. Consider the execution of $\mathsf{CheckEstimate}(U, V, \widetilde{e})$ from Algorithm 6.4.1. A natural way to simulate this with IS queries would be to use an IS query on $U' \cup V'$ instead of a BIS query on (U', V'). Assume for the sake of argument that $\mathsf{m} = n^{4/3}$ and $|U| = \sqrt{\mathsf{m}} = n^{2/3}$. Consider when the estimate $\widetilde{e}$ satisfies $\widetilde{e} = cn^{5/3}$ for a small constant c. In the CheckEstimate execution, there will be a value $i = \Theta(\log n)$ such that, with constant probability, $U' \subseteq U$ will contain at least two vertices and $V' \subseteq V$ will contain at least one vertex. In this case, $\mathsf{m}\left(U' \cup V'\right) \neq 0$ even though $\mathsf{m}\left(U', V'\right) = 0$. Thus, using IS queries will lead to incorrectly accepting on such a sample, and this would lead to the CoarseEstimator outputting the estimate $\widetilde{e} = \Theta(n^{5/3})$

even though the true number of edges is $\mathsf{m} = n^{4/3}$.

6.6 Conclusions

In this chapter, we explored the task of using either BIS or IS queries to estimate the number of edges in a graph. We presented randomized algorithms giving a $(1+\varepsilon)$-approximation using $\text{polylog}(n)/\varepsilon^4$ BIS queries and $\min\left\{n^2/(\varepsilon^2\mathsf{m}), \sqrt{\mathsf{m}}/\varepsilon\right\} \cdot \text{polylog}(n)$ IS queries. Our algorithms estimated the number of edges by first sparsifying the original graph and then exactly counting edges spanning certain bipartite subgraphs. Below we describe a few open directions for future research.

6.6.1 Open directions

An obvious unresolved question is whether there is an algorithm to estimate the number of edges with $o(\sqrt{\mathsf{m}})$ IS queries when $\mathsf{m} = o(n^{4/3})$. In this context proving a lower bound of $\Omega(\sqrt{\mathsf{m}})$ IS queries would also be interesting. The arguments in Section 6.5.3 lend some support to the possibility that a non-trivial lower bound might hold for the case of IS queries.

Other open questions include using a polylogarithmic number of BIS queries to estimate the number of cliques in a graph (see [64] for an algorithm using degree, neighbor and edge existence queries) or to sample a uniformly random edge (see [65] for an algorithm using degree, neighbor and edge existence queries). In general, any graph estimation problems may benefit from BIS or IS queries, possibly in combination with standard queries (such as neighbor queries). Finally, it would be interesting to know what other oracles, besides subset queries, enable estimating graph parameters with a polylogarithmic number of queries.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Summary of Thesis

We presented several results about the algorithms and complexity of pairwise problems in data science. In Chapter 3, we provided new algorithms and proved new lower bounds for computing a similarity join in a distributed model. Our main insight involved connecting similarity search to questions in extremal graph theory. In Chapter 4, we then dived further into the question of edge-isoperimetric inequalities for the Hamming distance similarity graph. We demonstrated tighter upper bounds on the maximum number of pairs of close vectors in subsets of the binary hypercube. Chapter 5 concerned a new clustering algorithm for edit distance, with motivation coming from the emerging area of DNA data storage. We exhibited new algorithms, with strong guarantees both in theory and practice, and we evaluated these algorithms on real data from a DNA data storage system. Finally, in Chapter 6, we considered an oracle model, based on independent set queries, and gave new randomized algorithms for estimating the number of edges in an unknown graph. Our results are the first to achieve a sub-polynomial query complexity for this graph parameter estimation problem.

In many ways, a thesis is only the beginning of the story. The results contained here are part of a larger body of work, and they give rise to many interesting open questions. Many of the open questions have already been mentioned in the conclusions of the individual chapters. We add a few more to the list.

7.2 Better Embeddings for Edit Distance

We highlight an open question from Chapter 5 about developing a better embedding from edit distance to a simpler metric space, that works for random strings and constant noise

rate for clusters. Recall that $\mathcal{U}_m$ is the uniform distribution on Σ^m. For random strings $x, y \in \mathcal{U}_m$, the signatures σ_q provide an embedding from edit distance to Hamming distance that preserves distances up to a multiplicative factor of q, which we can set to be $O(\log m)$, by Lemma 5.13. An important open question is improving this to work for an $O(1)$ factor, which would allow the embedding to be used with constant noise rate.

In particular, we ask the following question. Is there a function $f : \Sigma^m \to \mathcal{Y}$ for a metric space $(\mathcal{Y}, d_{\mathcal{Y}})$ with the following properties.

1. $f(x)$ is computable in time $\text{poly}(m)$, and can be computed for each $x \in \Sigma^m$ independently for a dataset $S \subseteq \Sigma^m$.

2. $f(x)$ can be represented using $\widetilde{O}(m \log |\Sigma|)$ bits.

3. $d_{\mathcal{Y}}(x, y)$ can be computed in time $\widetilde{O}(m \log |\Sigma|)$.

4. For all $x, y \in \Sigma^m$, with $d_{\mathrm{E}}(x, y) \leqslant pm$, for a small constant $p > 0$, it holds that there exists a universal constant $C > 0$ such that $d_{\mathcal{Y}}(f(x), f(y)) \leqslant Cpm$.

5. For random $x, y \sim \mathcal{U}_m$, it holds that there exists a universal constant $C > 0$ such that

$$\frac{m}{C} \leqslant d_{\mathcal{Y}}(f(x), f(y)),$$

with high probability, when m is large enough.

For example, it might be possible to let $\mathcal{Y}$ be Hamming distance on $O(m)$ bits, but it seems like a fundamentally different embedding is needed, perhaps looking at q-grams for many different values of q at the same time. Any embedding f should use the randomness of the strings, and likely will fail for worst-case strings (just as the signatures do). Lower bounds for embedding worst-case edit distance into ℓ_1 show that the distances must distort by an $\Omega(\log m)$ factor [103], and a lower bound of $\Omega(\sqrt{m})$ is known for ℓ_2 embeddings (even for embedding Hamming distance into ℓ_2) [117]. A reasonable first step could be to modify or better analyze known embeddings for this case of random strings [46, 47, 124].

7.3 Larger Context: Data Science Algorithms

In contemporary data-driven applications, the ideal algorithm offers a tunable trade-off between guarantees and efficiency. These guarantees can be on estimation accuracy or algorithmic approximation, and the efficiency can be for storage, computation, communication, or number of queries. This unfolding, complex space of trade-offs will lead to a wealth of new models, tools, and insights. A central challenges is that large datasets rule out all but nearly-linear time algorithms. Many fields have adopted approximation algorithms and domain-specific techniques, and classical worst-case analysis has become outdated. Unfortunately, there is still no unified, contemporary replacement for the current framework. Modern data science demands a new theory of the possibilities and limitations of algorithms.

Average-Case Analysis.

Average-case analysis of algorithms will lead to a better understanding of the best methods for various bioinformatics and data science applications.

A key lesson from Chapter 5 about our clustering algorithm for DNA data storage was that natural input properties, such as randomness, can enable efficient approximate solutions that sacrifice negligible accuracy. Analogous observations appear in the machine learning and optimization communities. It is an interesting problem to find other domains where we can develop a rigorous understanding of why certain techniques work well in practice.

Modeling Trade-offs.

This thesis contains results that involved optimizing for many trade-offs (computation, accuracy, communication). A limitation is that these results have different notions of correctness and complexity, and they were analyzed for disparate models. To achieve broader impact, it would be desirable to unify the models and trade-offs of distributed algorithms for various tasks. Examples include optimization, statistical estimation, and graph analytics questions. We give details for extending similarity search and clustering to other domains.

Similarity Search and Clustering

Modern data science domains have complex notions of similarity. Besides Hamming distance, it would be exciting to design algorithms and prove lower bounds for similarity search for some of these diverse metrics. In the database and network science communities, graph edit distance has surfaced as an important measure, and our results may be useful for distributed search or clustering. Another avenue of research comes from robotics and virtual reality, where finding all-pairs correspondences between visual scenes is a core preprocessing step, and new algorithms could have significant impact. A third application comes from the area of genomic analysis, where components of our clustering algorithm could improve the state-of-the-art methods for aligning DNA sequences or for finding structurally similar genes.

As a larger goal, we could identify general properties of input sets that would greatly improve the throughput and accuracy of various algorithms. Overall, the research in thesis is a small contribution to the emerging area of distributed algorithm design, which solves problems that are neither embarrassingly parallel nor impenetrably global.

BIBLIOGRAPHY

[1] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating Branching Programs with Edit Distance and Friends: Or: A polylog shaved is a lower bound made. In *STOC*, 2016.

[2] Margareta Ackerman, Shai Ben-David, David Loker, and Sivan Sabato. Clustering Oligarchies. In *AISTATS*, 2013.

[3] Margareta Ackerman and Sanjoy Dasgupta. Incremental Clustering: The Case for Extra Clusters. In *Advances in Neural Information Processing Systems*, pages 307–315, 2014.

[4] Foto N Afrati, Anish Das Sarma, David Menestrina, Aditya Parameswaran, and Jeffrey D Ullman. Fuzzy Joins using MapReduce. In *ICDE*. IEEE, 2012.

[5] Foto N. Afrati, Anish Das Sarma, Anand Rajaraman, Pokey Rule, Semih Salihoglu, and Jeffrey D. Ullman. Anchor-Points Algorithms for Hamming and Edit Distances Using MapReduce. In *ICDT*, 2014.

[6] Thomas Dybdahl Ahle, Rasmus Pagh, Ilya Razenshteyn, and Francesco Silvestri. On the Complexity of Inner Product Similarity Join. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 151–164. ACM, 2016.

[7] Rudolf Ahlswede and Ning Cai. A Counterexample to Kleitman's Conjecture Concerning an Edge-Isoperimetric Problem. *Combinatorics, Probability and Computing*, 8(04):301–305, 1999.

[8] Rudolf Ahlswede and Ning Cai. Appendix: On Edge-Isoperimetric Theorems for Uniform Hypergraphs. In *General Theory of Information Transfer and Combinatorics*, pages 979–1005. Springer, 2006.

[9] Rudolf Ahlswede and Gyula Katona. Contributions to the Geometry of Hamming Spaces. *Discrete Mathematics*, 17(1):1–22, 1977.

[10] Rudolf Ahlswede and Gyula OH Katona. Graphs with maximal number of adjacent pairs of edges. *Acta Mathematica Hungarica*, 32(1-2):97–120, 1978.

[11] Dror Aiger, Haim Kaplan, and Micha Sharir. Reporting Neighbors in High-Dimensional Euclidean Space. *SIAM J. Comput.*, 43(4):1363–1395, 2014.

[12] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[13] Josh Alman, Timothy M Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. *arXiv*, abs/1608.04355, 2016.

[14] Josh Alman and Ryan Williams. Probabilistic Polynomials and Hamming Nearest Neighbors. In *Proceedings, 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–150, 2015.

[15] Noga Alon and Imre Z Ruzsa. Non-averaging subsets and non-vanishing transversals. *Journal of Combinatorial Theory, Series A*, 86(1):1–13, 1999.

[16] Ian Anderson. *Combinatorics of Finite Sets*. Courier Corporation, 1987.

[17] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *NIPS*, pages 1225–1233, 2015.

[18] Alexandr Andoni and Robert Krauthgamer. The Computational Hardness of Estimating Edit Distance. *SIAM Journal on Computing*, 39(6):2398–2429, 2010.

[19] Alexandr Andoni and Robert Krauthgamer. The Smoothed Complexity of Edit Distance. *ACM Transactions on Algorithms (TALG)*, 8(4):44, 2012.

[20] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal Hashing-based Time-Space Trade-offs for Approximate Near Neighbors. *arXiv*, abs/1608.03580, 2016.

[21] Alexandr Andoni and Ilya Razensteyn. Tight Lower Bounds for Data-Dependent Locality-Sensitive Hashing. In Sándor Fekete and Anna Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[22] B. Aronov and S. Har-Peled. On Approximating the Depth and Related Problems. *SIAM J. Comput.*, 38(3):899–921, 2008.

[23] Pranjal Awasthi, Afonso S Bandeira, Moses Charikar, Ravishankar Krishnaswamy, Soledad Villar, and Rachel Ward. Relax, no need to round: Integrality of clustering formulations. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 191–200. ACM, 2015.

[24] Arturs Backurs and Piotr Indyk. Edit Distance Cannot be Computed in Strongly Subquadratic time (unless SETH is false). In *STOC*, 2015.

[25] Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient distributed locality sensitive hashing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2174–2178, New York, NY, USA, 2012. ACM.

[26] Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Clustering Under Approximation Stability. *Journal of the ACM (JACM)*, 60(2):8, 2013.

[27] Maria-Florina Balcan, Yingyu Liang, and Pramod Gupta. Robust Hierarchical Clustering. *Journal of Machine Learning Research*, 15(1):3831–3871, 2014.

[28] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[29] Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A Sublinear Algorithm for Weakly Approximating Edit Distance. In *STOC*, 2003.

[30] P. Beame, S. Har-Peled, S. Natarajan Ramamoorthy, C. Rashtchian, and M. Sinha. Edge estimation with independent set oracles. In Anna R. Karlin, editor, *Innov. Theo. Comp. Sci.* (ITCS), volume 94 of *LIPIcs*, pages 38:1–38:21, 2018.

[31] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. In *PODS*. ACM, 2013.

[32] Paul Beame and Cyrus Rashtchian. Massively-Parallel Similarity Join, Edge-Isoperimetry, and Distance Correlations on the Hypercube. In *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 289–306, 2017.

[33] I. Ben-Eliezer, T. Kaufman, M. Krivelevich, and D. Ron. Comparing the strength of query types in property testing: The case of k-colorability. *Computational Complexity*, 22(1):89–135, 2013.

[34] Arthur J Bernstein. Maximally connected arrays on the n-cube. *SIAM J. Applied Mathematics*, 15(6):1485–1489, 1967.

[35] Brenda Betancourt, Giacomo Zanella, Jeffrey W Miller, Hanna Wallach, Abbas Zaidi, and Beka Steorts. Flexible Models for Microclustering with Application to Entity Resolution. In *NIPS*, 2016.

[36] Sergei Bezrukov. Edge Isoperimetric Problems on Graphs. *Graph Theory and Combinatorial Biology*, 7:157–197, 1999.

[37] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin, Heidelberg, 2006.

[38] Béla Bollobás. *Combinatorics: Set systems, Hypergraphs, Families of Vectors, and Combinatorial Probability*. Cambridge University Press, 1986.

[39] Béla Bollobás and Imre Leader. Sums in the grid. *Discrete Mathematics*, 162(1-3):31–48, 1996.

[40] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A DNA-based Archival Storage System. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649. ACM, 2016.

[41] Andrei Z Broder. On the Resemblance and Containment of Documents. In *Compression and Complexity of Sequences*, pages 21–29. IEEE, 1997.

[42] Andrei Z Broder. Identifying and filtering near-duplicate documents. In *Combinatorial pattern matching*, pages 1–10. Springer, 2000.

[43] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic Clustering of the Web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[44] Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

[45] S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015.

[46] Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming Algorithms for Embedding and Computing Edit Distance in the Low Distance Regime. In *STOC*, 2016.

[47] Moses Charikar and Robert Krauthgamer. Embedding the Ulam Metric into L_1. *Theory of Computing*, 2(11):207–224, 2006.

[48] Moses S Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing (STOC)*, pages 380–388. ACM, 2002.

[49] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near Optimal LP Rounding Algorithm for Correlation Clustering on Complete and Complete k-partite Graphs. In *STOC*, 2015.

[50] C. L. Chen and W. H. Swallow. Using group testing to estimate a proportion, and to test the b inomial model. *Biometrics*, 46(4):10351046, December 1990.

[51] Jiecao Chen, He Sun, David Woodruff, and Qin Zhang. Communication-Optimal Distributed Clustering. In *Advances in Neural Information Processing Systems*, pages 3720–3728, 2016.

[52] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[53] Fan Chung and Linyuan Lu. Concentration Inequalities and Martingale Inequalities: a Survey. *Internet Mathematics*, 3(1):79–127, 2006.

[54] George F Clements and Bernt Lindström. A Generalization of a Combinatorial Theorem of Macaulay. *Journal of Combinatorial Theory*, 7(3):230–238, 1969.

[55] David Conlon, Jacob Fox, and Benny Sudakov. An Approximate Version of Sidorenko's Conjecture. *Geometric and Functional Analysis*, 20(6):1354–1366, 2010.

[56] David Conlon and Joonkyung Lee. Finite Reflection Groups and Graph Norms. *Advances in Mathematics*, 315:130–165, 2017.

[57] H. Dell and J. Lapinskas. Fine-grained reductions from approximate counting to decision. *CoRR*, abs/1707.04609, 2017.

[58] Dong Deng, Guoliang Li, Shuang Hao, Jiannan Wang, and Jianhua Feng. Massjoin: A Mapreduce-based Method for Scalable String Similarity Joins. In *ICDE*, pages 340–351. IEEE, 2014.

[59] Martin Dietzfelbinger. Universal Hashing and k-Wise Independent Random Variables via Integer Arithmetic Without Primes. In *STACS*, 1996.

[60] R. Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, 14(4):436–440, 12 1943.

[61] D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[62] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[63] T. Eden, A. Levi, D. Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. *SIAM J. Comput.*, 46(5):1603–1646, 2017.

[64] T. Eden, D. Ron, and C. Seshadhri. On Approximating the Number of k-cliques in Sublinear Time. *CoRR*, abs/1707.04858, July 2017.

[65] T. Eden and W. Rosenbaum. On sampling edges almost uniformly. In *1st Symp. Simplicity Alg. (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 7:1–7:9. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[66] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16, 2007.

[67] Paul Erdős and Miklos Simonovits. Compactness Results in Extremal Graph Theory. *Combinatorica*, 2(3):275–288, 1982.

[68] Yaniv Erlich and Dina Zielinski. DNA Fountain Enables a Robust and Efficient Storage Architecture. *Science*, 355(6328):950–954, 2017.

[69] M. Falahatgar, A. Jafarpour, A. Orlitsky, V. Pichapati, and A. T. Suresh. Estimating the number of defectives with group testing. In *IEEE Int. Symp. Inf. Theo. ISIT*, pages 1376–1380. IEEE, 2016.

[70] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35(4):964–984, 2006.

[71] A. V Fishkin. Disk graphs: A short survey. In *Int. Workshop Approx. and Online Alg. (WAOA)*, pages 260–264. Springer, 2003.

[72] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics New York, 2001.

[73] Shirshendu Ganguly, Elchanan Mossel, and Miklós Z. Rácz. Sequence Assembly from Corrupted Shotgun Reads. In *ISIT*, pages 265–269, 2016. http://arxiv.org/abs/1601.07086.

[74] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *Nature*, 494(7435):77–80, 2013.

[75] O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Struct. Algo.*, 32(4):473–493, 2008.

[76] Sreenivas Gollapudi and Rina Panigrahy. A Dictionary for Approximate String Search and Longest Prefix Search. In *CIKM*, 2006.

[77] M. Gonen, D. Ron, and Y. Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM J. Discrete Math*, 25(3):1365–1411, 2011.

[78] Teofilo F Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

[79] Luis Gravano, Panagiotis G Ipeirotis, Hosagrahar Visvesvaraya Jagadish, Nick Koudas, Shanmugauelayut Muthukrishnan, Divesh Srivastava, et al. Approximate String Joins in a Database (almost) for Free. In *VLDB*, volume 1, pages 491–500, 2001.

[80] Sudipto Guha, Yi Li, and Qin Zhang. Distributed Partial Clustering. *arXiv preprint arXiv:1703.01539*, 2017.

[81] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

[82] Hiroyuki Hanada, Mineichi Kudo, and Atsuyoshi Nakamura. On Practical Accuracy of Edit Distance Approximation Algorithms. *arXiv preprint arXiv:1701.06134*, 2017.

[83] Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173. American mathematical society Providence, 2011.

[84] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[85] L. H. Harper. Optimal Assignments of Numbers to Vertices. *Journal of the Society for Industrial and Applied Mathematics*, 12(1):131–135, 1964.

[86] L. H. Harper. On a Problem of Kleitman and West. *Discrete Mathematics*, 93(2):169–182, 1991.

[87] L. H. Harper. *Global Methods for Combinatorial Isoperimetric Problems*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2004.

[88] Sergiu Hart. A Note on the Edges of the n-cube. *Discrete Mathematics*, 14(2):157–163, 1976.

[89] Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J Miller. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *PVLDB*, 2(1):1282–1293, 2009.

[90] Christian Hennig, Marina Meila, Fionn Murtagh, and Roberto Rocci. *Handbook of Cluster Analysis*. CRC Press, 2015.

[91] R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *J. Comp. Sys. Sci.*, 67:367–375, 2001.

[92] Yu Jiang, Dong Deng, Jiannan Wang, Guoliang Li, and Jianhua Feng. Efficient Parallel Partition-based Algorithms for Similarity Search and Join with Edit Distance Constraints. In *Joint EDBT/ICDT Workshops*, 2013.

[93] Yu Jiang, Guoliang Li, Jianhua Feng, and Wen-Syan Li. String Similarity Joins: An Experimental Evaluation. *Proceedings of the VLDB Endowment*, 7(8):625–636, 2014.

[94] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale Similarity Search with GPUs. *arXiv preprint arXiv:1702.08734*, 2017.

[95] J. Kahn, G. Kalai, and N. Linial. The Influence of Variables on Boolean Functions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 68–80, Washington, DC, USA, 1988. IEEE Computer Society.

[96] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On Clusterings: Good, Bad and Spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

[97] Gyula Katona. Intersection Theorems for Systems of Finite Sets. *Acta Mathematica Hungarica*, 15(3-4):329–337, 1964.

[98] Nets Hawk Katz and Terence Tao. A New Bound on Partial Sum-sets and Difference-sets, and Applications to the Kakeya Conjecture. *arXiv preprint 9906.097*, 1999.

[99] Leonard Kaufman and Peter J Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*, volume 344. John Wiley & Sons, 2009.

[100] R. J. Klein, C. Zeiss, E. Chew, J.-Y. Tsai, R.S. Sackler, C. Haynes, A.K. Henning, J.P. SanGiovanni, S.M. Mane, S.T. Mayne, R.B. Bracken, F.L. Ferris, J. Ott, C. Barnstable, and J. Noh. Complement factor H polymorphism in age-related macular degeneration. *Science*, 308(5720):385–389, 2005.

[101] Daniel J. Kleitman. On a Combinatorial Conjecture of Erdős. *Journal of Combinatorial Theory*, 1(2):209 – 214, 1966.

[102] Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. A Hierarchical Algorithm for Extreme Clustering. In *KDD*, 2017.

[103] Robert Krauthgamer and Yuval Rabani. Improved Lower Bounds for Embeddings Into L_1. *SIAM J. on Computing*, 38(6):2487–2498, 2009.

[104] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[105] Gad M Landau, Eugene W Myers, and Jeanette P Schmidt. Incremental String Comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

[106] Hans-Peter Lenhof and Michiel Smid. Sequential and Parallel Algorithms for the k-closest Pairs Problem. *International J. Comput. Geometry & Applications*, 5(03):273–288, 1995.

[107] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[108] Heng Li and Richard Durbin. Fast and Accurate Short Read Alignment with Burrows–Wheeler Transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[109] JL Li and Balazs Szegedy. On the Logarithimic Calculus and Sidorenko's Conjecture. *arXiv preprint arXiv:1107.1153*, 2011.

[110] Ping Li and Christian König. b-Bit Minwise Hashing. In *WWW*, pages 671–680. ACM, 2010.

[111] Ping Li, Art Owen, and Cun-Hui Zhang. One Permutation Hashing. In *NIPS*, 2012.

[112] John H Lindsey. Assignment of Numbers to Vertices. *The American Mathematical Monthly*, 71(5):508–516, 1964.

[113] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, pages 281–297, 1967.

[114] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu–Linial Stable Instances of Max Cut and Minimum Multiway Cut. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 890–906. SIAM, 2014.

[115] Gustavo Malkomes, Matt J Kusner, Wenlin Chen, Kilian Q Weinberger, and Benjamin Moseley. Fast Distributed k-center Clustering with Outliers on Massive Data. In *NIPS*, 2015.

[116] William J Masek and Michael S Paterson. A Faster Algorithm Computing String Edit Distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

[117] Jiří Matoušek. *Lectures on discrete geometry*, volume 212. Springer New York, 2002.

[118] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *SIGKDD*, pages 169–178. ACM, 2000.

[119] Marina Meilă and David Heckerman. An Experimental Comparison of Model-based Clustering Methods. *Machine learning*, 42(1-2):9–29, 2001.

[120] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discrete Mathematics*, 21(4):930–935, 2007.

[121] Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory*, 6(1):5, 2014.

[122] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proc. 23rd ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1123–1131, 2012.

[123] L. Organick, S. Dumas Ang, Y-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. Takahashi, S. Newman, H-Y. Parker, C. Rashtchian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss. Random Access in Large-Scale DNA Data Storage. *Nature biotechnology*, 36(3):242, 2018.

[124] Rafail Ostrovsky and Yuval Rabani. Low Distortion Embeddings for Edit Distance. *Journal of the ACM (JACM)*, 54(5):23, 2007.

[125] Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear Probing with Constant Independence. *SIAM J. Comput.*, 39(3):1107–1120, September 2009.

[126] Rasmus Pagh. Locality-sensitive hashing without false negatives. In *SODA*, pages 1–9. SIAM, 2016.

[127] Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-Efficient Similarity Join. In *Algorithms-ESA 2015*, pages 941–952. Springer, 2015.

[128] Xinghao Pan, Dimitris Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Parallel Correlation Clustering on Big Graphs. In *Advances in Neural Information Processing Systems*, pages 82–90, 2015.

[129] Rina Panigrahy, Kunal Talwar, and Udi Wieder. A Geometric Approach to Lower Bounds for Approximate Near-Neighbor Search and Partial Match. In *FOCS*, 2008.

[130] Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower Bounds on Near Neighbor Search via Metric Expansion. In *FOCS*, 2010.

[131] Ninh Pham and Rasmus Pagh. Scalability and total recall with fast coveringlsh. *arXiv preprint 1602.02620*, 2016.

[132] Zeehasham Rasheed, Huzefa Rangwala, and Daniel Barbara. Efficient Clustering of Metagenomic Sequences using Locality Sensitive Hashing. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 1023–1034. SIAM, 2012.

[133] Cyrus Rashtchian, Konstantin Makarychev, Miklos Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering Billions of Reads for DNA Data Storage. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 3360–3371. Curran Associates, Inc., 2017.

[134] Ilya Razenshteyn. *High-Dimensional Similarity Search and Sketching: Algorithms and Hardness*. PhD thesis, Massachusetts Institute of Technology, 2017.

[135] D. Ron and G. Tsur. The power of an example: Hidden set size approximation using group queries and conditional sampling. *ACM Trans. Comp. Theo.*, 8(4):15:1–15:19, 2016.

[136] Anish Das Sarma, Foto N. Afrati, Semih Salihoglu, and Jeffrey D. Ullman. Upper and Lower Bounds on the Cost of a Map-reduce Computation. *Proc. VLDB Endow.*, 6(4):277–288, February 2013.

[137] C. Seshadhri. A simpler sublinear algorithm for approximating the triangle count. *CoRR*, abs/1505.01927, May 2015.

[138] Alexander Sidorenko. A Correlation Inequality for Bipartite Graphs. *Graphs and Combinatorics*, 9(2-4):201–204, 1993.

[139] H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1:801–804, 1956.

[140] L. Stockmeyer. The complexity of approximate counting (preliminary version). In *Proc. 15th Annu. ACM Sympos. Theory Comput.* (STOC), pages 118–126, Boston, Massachusetts, 1983.

[141] L. Stockmeyer. On approximation algorithms for #P. *SIAM J. Comput.*, 14(4):849–861, 1985.

[142] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming Similarity Search Over One Billion Tweets Using Parallel Locality-Sensitive Hashing. *Proceedings of the VLDB Endowment*, 6(14):1930–1941, 2013.

[143] W. H. Swallow. Group testing for estimating infection rates and probabilities of disease transmission. *Phytopathology*, 75(8):882, 1985.

[144] Mikkel Thorup. High Speed Hashing for Integers and Strings. *CoRR*, abs/1504.06804, 2015.

[145] Esko Ukkonen. Algorithms for Approximate String Matching. *Information and control*, 64(1-3):100–118, 1985.

[146] Esko Ukkonen. Approximate String-matching with q-grams and Maximal Matches. *Theoretical computer science*, 92(1):191–211, 1992.

[147] Hongya Wang, Jiao Cao, LihChyun Shu, and Davood Rafiei. Locality Sensitive Hashing Revisited: Filling the Gap Between Theory and Algorithm Analysis. In *CIKM*, pages 1969–1978, New York, NY, USA, 2013. ACM.

[148] J. Wang, E. Lo, and M. L. Yiu. Identifying the most connected vertices in hidden bipartite graphs using group testing. *IEEE Tran. Knowl. Data Eng.*, 25(10):2245–2256, 2013.

[149] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient Similarity Joins for Near-duplicate Detection. *ACM Transactions on Database Systems*, 36(3):15, 2011.

[150] Cairong Yan, Xue Zhao, Qinglong Zhang, and Yongfeng Huang. Efficient string similarity join in multi-core and distributed systems. *PloS one*, 12(3):e0172526, 2017.

[151] Andrew C. Yao. Lower Bounds by Probabilistic Arguments. In *FOCS*, 1983.

[152] SM Hossein Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and Error-Free DNA-Based Data Storage. *bioRxiv*, page 079442, 2016.

[153] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. String similarity search and join: a survey. *Frontiers of Computer Science*, 10(3):399–417, 2016.

[154] Peisen Yuan, Chaofeng Sha, and Yi Sun. Hashˆ{ed}-Join: Approximate String Similarity Join with Hashing. In *International Conference on Database Systems for Advanced Applications*, pages 217–229. Springer, 2014.

[155] Reza Bosagh Zadeh and Ashish Goel. Dimension Independent Similarity Computation. *The Journal of Machine Learning Research*, 14(1):1605–1626, 2013.

[156] Haoyu Zhang and Qin Zhang. EmbedJoin: Efficient Edit Similarity Joins via Embeddings. In *KDD*, 2017.

[157] Eduard Valera Zorita, Pol Cuscó, and Guillaume Filion. Starcode: Sequence Clustering Based on All-pairs Search. *Bioinformatics*, 2015.