Sublinear Algorithms for Massive Data Problems

by

Sepideh Mahabadi

M.Sc., Massachusetts Institute of Technology (2013)
B.Sc., Sharif University of Technology (2011)

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

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

© Massachusetts Institute of Technology 2017. All rights reserved.

Author

Department of Electrical Engineering and Computer Science

August 24, 2017

Certified by

Professor Piotr Indyk

Professor of Electrical Engineering and Computer Science

Thesis Supervisor

Accepted by

Professor Leslie A. Kolodziejski

Chair, Committee on Graduate Students
Department of Electrical Engineering and Computer Science
Sublinear Algorithms for Massive Data Problems

by

Sepideh Mahabadi

Submitted to the Department of Electrical Engineering and Computer Science
on August 24, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In this thesis, we present algorithms and prove lower bounds for fundamental computational problems in the models that address massive data sets. The models include streaming algorithms, sublinear time algorithms, property testing algorithms, sub-linear query time algorithms with preprocessing, or computing small summaries for large data. More precisely, we study the following problems.

The (Approximate) Nearest Neighbor problem models the task of searching among a large data set of objects. Given a data set of \( n \) points in a high dimensional space, its goal is to search for the closest point in the data set to a given query point, in sublinear time, and by suitably preprocessing the data. This problem has numerous applications in image and video databases, information retrieval, clustering, and many others. In these applications, the points model the objects in a large data set, and their closeness measure similarity between the objects. However, for the purpose of many applications, the basic formulation of Nearest Neighbor as described, encounters several challenges which we address in this thesis: we show how to deal with the case where the data is corrupted or incomplete, how to handle multiple related queries, and how to handle a data set of more complex objects rather than simple points.

Next, we show a general approach for solving massive data problems. We introduce the notion of Composable Coresets, defined as small summaries of multiple data sets that can be aggregated together to summarize the whole data. We show how to compute such summaries for several clustering problems, and at the same time, demonstrate that no such summaries are possible for other natural problems such as maximum coverage.

Finally, we study the Set Cover problem in alternate sublinear models: streaming algorithms (where one makes a small number of passes over the data using small storage), and sublinear time algorithms (where one computes the answer without reading the whole input). We present tight approximation algorithms for the Set Cover problem in both of these models.

In this thesis, we introduce theoretical problems and concepts that model computational issues arising in databases, computer vision and other areas. Most of the presented algorithms are simple and practical to implement.
Thesis Supervisor: Professor Piotr Indyk
Title: Professor of Electrical Engineering and Computer Science
“In the name of God, the Most Gracious, the Most Merciful”

To My Beloved Family
Acknowledgments

Foremost, I would like to express my deepest gratitude to my advisor Piotr Indyk for his perfect guidance, caring, engagement, patience, and support throughout my Ph.D. studies. I have always been very impressed by his immense knowledge, deep insight, and excellent advising approach, as well as his admirable personality. He guided me in shaping the research direction that I am very much excited about, which has also led to completion of this thesis. His constant presence and constructive feedback have contributed significantly to my research, and enhancing my writing and presentation skills. He was always ready to help his students with every challenge we had during our Ph.D., and I do not remember having difficulty in reaching out to him to get advice from him or talk about research, even when he was on sabbatical. I feel very fortunate having had him as my advisor, and in fact, I cannot express in words my appreciation for all I learned from him about research, communication, teaching and life. He has set a perfect example of an ideal advisor for me. Thanks a lot Piotr!

I would like to deeply thank my committee members who strongly supported me during my Ph.D. studies. In particular, I would like to thank Sariel Har-Peled who has been the second advisor to me. The summer I spent at UIUC was very productive and I learned a lot from Sariel about research (and in particular Geometry) and also writing. I always enjoy discussing both research and non-research topics with him, and moreover his sense of humor makes it also fun to work with him. Ronitt Rubinfeld is a source of strength and motivation. She has been very supportive to me, and every advice that I received from her helped me in making a better decision. She also introduced me to the field of sublinear time algorithms which lead to a chapter in this thesis. Thanks a lot Ronitt! I am also very grateful to Stefanie Jegelka and I was very fortunate that Stefanie joined MIT at the perfect time and could benefit from her expertise in my work and thesis. The door of her office was always open to me and I rarely left her office without learning a new topic or gaining an invaluable intuition on the applications concerned with theory.

I am also very thankful of all my mentors during my internships: Mohammad
Mahdian, Howard Karloff, Edo Liberty, Julia Chuzhoy, and Yury Makarychev. I learned a lot from each of them and my delightful experiences at Google, Yahoo! Research, and TTIC deepened my insight and knowledge in different areas of TCS. I would like to thank my B.Sc. advisor, Mohammad Ghodsi for his invaluable support since I was participating in IOI. I would like to thank the rest of the people from whom I received advice every now and then, and in particular, Alexander Madry, Vahab Mirrokni, Mohsen Ghaffari, Michael Kapralov, and Shayan Oveis Gharan.

I would like to thank all my collaborators Erik Demaine, Sariel Har-Peled, Piotr Indyk, Robert Kleinberg, Mohammad Mahdian, Vahab Mirrokni, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, Anak Yodpinyane, and Yang Yuan for their contributions in the publications that appear in this thesis.

I would like to say thank you to my friends at the theory group of MIT. Foremost, to Ali Vakilian, who was a perfect friend, teammate and collaborator since undergrad. To Ilya Razenshteyn and Rati Gelashvili for our enjoyable friendship since we were teammates at ACM-ICPC. To Zeyuan Allen-Zhu, who has helped me since I was preparing for IOI, and years later he became my student buddy when I first arrived at MIT. To all people in Piotr’s group over these years and in particular, Arturs Backurs, Ilya, Ludwig Schmidt, and Tal Wagner for all I learned from them. To my friends, Maryam Aliakbarpour, Ioana Ivan, Madalina Persu, Mohammad Bavarian, Mira Radeva, Anak Yodpinyane, Adam Bouland, Henry Yuen, Themistoklis Gouleakis, Christos Tzamos, Morteza Zadimoghaddam, Saeed Mehraban, Nicholas Schiefer, Shibani Santurkar, Adam Sealfon, and the rest of the theory group at MIT, as well as MIT itself, for making my Ph.D. journey an enjoyable one.

I would like to thank my social friends who helped me in ups and downs, in particular Shiva, Parisa, Ameer, Samaneh, Mehrnoosh, Amirali, Ameneh, Mohammad, Fatemeh, Elaheh, Fereshteh, Asma, Ardavan, Amin, Ehsan, and Kamran.

Last but not least, I would like to sincerely thank my mother Parivash, my father Javad, and my brother Shahryar, for their continuous encouragement, support, and love. I cannot express in words how much I owe them. So this thesis is dedicated to them.
# Contents

1 Introduction .......................................................... 19
   1.1 Approximate Nearest Neighbor .............................. 21
      1.1.1 Robust Nearest Neighbor Search ...................... 23
      1.1.2 Nearest Line Search ................................ 24
      1.1.3 Sparse Linear Regression: Nearest Induced Flat .... 24
      1.1.4 Simultaneous Nearest Neighbor Search ............... 25
   1.2 Composable Coresets ......................................... 26
   1.3 The Set-Cover Problem ...................................... 27
      1.3.1 Streaming and the Distributed Algorithms .......... 28
      1.3.2 Sublinear Time Algorithms ............................ 29

I Approximate Nearest Neighbor .................................. 30

2 Robust Nearest Neighbor Search ................................ 31
   2.1 Introduction and Background .............................. 31
      2.1.1 Results ................................................ 32
      2.1.2 Techniques ............................................ 33
   2.2 Preliminaries ................................................ 35
      2.2.1 The Problem .......................................... 35
   2.3 The $k$-Robust ANN under the $L_{\rho}$-Norm ............ 37
      2.3.1 The Preprocessing and Query Algorithms ............ 37
      2.3.2 Analysis .............................................. 38
### 3 Nearest Line Search

- **3.1 Introduction and Background** ........................................... 43
  - 3.1.1 Overview of the Algorithms ........................................ 44
- **3.2 Definitions** ..................................................................... 49
- **3.3 Basic Modules** .............................................................. 50
  - 3.3.1 Unbounded Module ......................................................... 50
  - 3.3.2 Net Module .................................................................. 58
  - 3.3.3 Parallel Module ............................................................ 60
- **3.4 General Case of NLS** ..................................................... 62
  - 3.4.1 Data Structure ............................................................... 63
  - 3.4.2 Query Procedure ........................................................... 66
- **3.5 Analysis of NLS Algorithm** ............................................ 69
- **3.6 Almost Parallel Nearest Line Search (APNLS)** .................. 73
  - 3.6.1 Data Structure ............................................................... 74
  - 3.6.2 Query Procedure ........................................................... 78
  - 3.6.3 Analysis of APNLS Algorithm ......................................... 81

### 4 Sparse Linear Regression: Nearest Induced Flat

- **4.1 Introduction and Background** ........................................... 85
  - 4.1.1 Results .................................................................... 87
  - 4.1.2 Related work ............................................................... 89
  - 4.1.3 Techniques and Sketch of the Algorithms ......................... 89
- **4.2 Preliminaries** ................................................................. 91
  - 4.2.1 Notations .................................................................. 91
  - 4.2.2 Problems .................................................................. 93
- **4.3 Approximating the Nearest Induced and Nearest Linear Induced Flats** 93
  - 4.3.1 Approximating the Nearest Neighbor in a Uniform Star ........ 93
  - 4.3.2 Approximating the Nearest Flat in a Bouquet .................. 95
  - 4.3.3 The Result ................................................................. 97
4.4 Approximating the Nearest Induced Simplex .......................... 98
  4.4.1 Simplices and Distances ........................................ 98
  4.4.2 Approximating the Nearest Page in a Book ..................... 102
  4.4.3 Result: Nearest Induced Simplex ................................ 107
4.5 Lower bound .......................................................... 108
  4.5.1 Connection to Hopcroft’s Problem .............................. 109

5 Simultaneous Nearest Neighbor Search 111
  5.1 Introduction and Background ........................................ 111
    5.1.1 Results ...................................................... 113
    5.1.2 Techniques ................................................ 114
  5.2 Definitions and Preliminaries ...................................... 116
    5.2.1 0-Extension Problem ....................................... 117
  5.3 Independent Nearest Neighbors Algorithm ......................... 120
    5.3.1 Bounding the Pruning Gap Using 0-extension ............... 121
    5.3.2 Sparse Graphs .............................................. 124
  5.4 Lower Bound ...................................................... 126
  5.5 Experiments ....................................................... 128
    5.5.1 De-noising with Patches .................................... 131

II Composable Coresets 133

6 Composable Coresets 135
  6.1 Introduction and Background ...................................... 135
    6.1.1 Results ...................................................... 137
    6.1.2 Techniques ................................................ 140
    6.1.3 Related Work .............................................. 140
  6.2 Preliminaries ..................................................... 142
    6.2.1 GMM Algorithm ............................................. 142
    6.2.2 Local Search Algorithm .................................. 143
    6.2.3 Prefix Algorithm .......................................... 143
III The Set Cover Problem

7 The Streaming Set Cover Problem

7.1 Introduction and Background
7.1.1 Results
7.1.2 Techniques

7.2 Streaming Algorithm for Set Cover
7.2.1 Algorithm
7.2.2 Analysis

7.3 Lower Bound for Single Pass Algorithms

7.4 Geometric Set Cover
7.4.1 Preliminaries
7.4.2 Algorithm
7.4.3 Analysis

7.5 Lower Bound for Multipass Algorithms

7.6 Lower Bound for Sparse Set Cover in Multiple Passes

8 Set Cover in the Sublinear Time Model

8.1 Introduction and Background
8.1.1 Results
8.1.2 Related work

8.2 SubLinear Algorithms for the Set Cover Problem
8.2.1 Preliminaries
8.2.2 The \texttt{smallSetCover} Algorithm ........................................ 198
8.2.3 The \texttt{largeSetCover} Algorithm ....................................... 204
8.3 Lower Bound for the Cover Verification Problem ....................... 207
  8.3.1 Underlying Set Structure ...................................................... 208
  8.3.2 Proof of Theorem 8.3.1 ....................................................... 211
8.4 Simplified Lower Bounds for the Set Cover Problem ................... 215
  8.4.1 Construction of the Median Instance $I^*$ ............................... 216
  8.4.2 Distribution $\mathcal{D}(I^*)$ of Modified Instances $I'$ Derived from $I^*$ . 219
  8.4.3 Proof of Theorem 8.4.2 ....................................................... 222
  8.4.4 Reduction via Replication for Increasing the Minimum Set Cover
      Size .................................................................................. 223
  8.4.5 Reduction via Independent Copies for the Approximate Set Cover
      Problem ........................................................................... 224
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Net Module</td>
<td>45</td>
</tr>
<tr>
<td>3-2</td>
<td>Parallel Module</td>
<td>46</td>
</tr>
<tr>
<td>3-3</td>
<td>Unbounded Module</td>
<td>46</td>
</tr>
<tr>
<td>3-4</td>
<td>Figure of Lemma 3.3.1</td>
<td>53</td>
</tr>
<tr>
<td>3-5</td>
<td>An illustration of Lemma 3.3.12</td>
<td>60</td>
</tr>
<tr>
<td>3-6</td>
<td>Figure of Claim 3.5.4</td>
<td>71</td>
</tr>
<tr>
<td>3-7</td>
<td>Example of APNLS</td>
<td>84</td>
</tr>
<tr>
<td>4-1</td>
<td>Illustration for Lemma 4.3.1</td>
<td>95</td>
</tr>
<tr>
<td>4-2</td>
<td>Example base angles when $k = 3$</td>
<td>103</td>
</tr>
<tr>
<td>5-1</td>
<td>Results of INN for denoising</td>
<td>130</td>
</tr>
<tr>
<td>5-2</td>
<td>Improved results for denoising</td>
<td>132</td>
</tr>
<tr>
<td>7-1</td>
<td>Example for geometric set cover</td>
<td>165</td>
</tr>
<tr>
<td>7-2</td>
<td>Set Chasing and Intersection Set Chasing examples</td>
<td>184</td>
</tr>
<tr>
<td>7-3</td>
<td>Gadgets of the reduction of Intersection Set Chasing to Set Cover</td>
<td>185</td>
</tr>
<tr>
<td>7-4</td>
<td>Merging the two Set Chasing instances</td>
<td>186</td>
</tr>
<tr>
<td>7-5</td>
<td>An Illustration of the reduction of Intersection Set Chasing to Set Cover</td>
<td>187</td>
</tr>
<tr>
<td>8-1</td>
<td>A basic slab and an example of a swapping operation</td>
<td>210</td>
</tr>
<tr>
<td>8-2</td>
<td>Cover Verification example</td>
<td>210</td>
</tr>
<tr>
<td>8-3</td>
<td>Effect of a swap on ELTOf and SETOF tables</td>
<td>212</td>
</tr>
</tbody>
</table>
## List of Tables

4.1 Summary of results for SLR ........................................... 87

5.1 Empirical evaluation of INN algorithm ............................. 129

6.1 Notions of diversity .................................................. 139

7.1 Results on Streaming Set Cover .................................. 160

8.1 Summary of results for Sublinear Set Cover .................... 196
Chapter 1

Introduction

Recently, there has been an immense increase in the size of available data sets. These data sets come from different sources, such as media (images, video, live streams), social media (Facebook, Twitter), archives of scanned documents, public web, and many others. The big problem is how to handle these large amounts of data. We need to read, store, and search through this data, and ideally extract the most relevant information from it in order to solve problems over these huge data sets.

This development has had a significant impact on the design of algorithms and has led to the emergence of new computational models that capture various aspects of massive data computation. Examples include streaming algorithms, sublinear time algorithms, property testing algorithms, distributed algorithms, and sublinear query time algorithms with preprocessing. In this thesis, we will present algorithms and prove lower bounds for fundamental computational problems in some of these models.

One of the most basic tasks in data processing and analysis is searching through this data, which becomes challenging as the size of data grows. This task is often formalized as the (Approximate) Nearest Neighbor problem, where the goal is to search for the most similar object to a given query in a large data set, in sublinear query time, and by suitably preprocessing the data. This fundamental problem has numerous applications in information retrieval, image and video databases, clustering, and in many others.

Despite an extensive amount of research on this topic, the basic formulation of
nearest neighbor encounters several challenges in many applications which we address in this thesis. This includes dealing with the case where the data is corrupted or incomplete, handling multiple related queries, and handling a data set of lines as opposed to points.

One general approach for solving a problem over a massive data set is to extract the most relevant information from the data. In this thesis, we introduce the notion of Composable Coresets, defined as small summaries of multiple data sets that can be aggregated together to summarize the whole data. This notion has many applications. For example in distributed algorithms where the data set is partitioned over multiple repositories and we are interested to solve a problem over the whole data without putting together all of it in one place. In fact, every repository can compute the small summary for its own data and only send this composable coreset to the computation center. In this thesis, we show how to compute such summaries for several clustering problems, and at the same time, demonstrate that no such summaries are possible for some of the natural problems such as maximum coverage.

Subsequently, we investigate alternate approaches to coverage problems, which includes streaming algorithms (where one makes a small number of passes over the data using small storage), and sublinear time algorithms (where one computes the answer without reading the whole input). In particular, here we show tight approximation algorithms for the Set Cover problem in both the streaming and the sublinear time models.

More broadly, this thesis contains problems that are interesting from the theoretical point of view and at the same time have real world applications. We introduce new problems and concepts that model computational issues arising in databases, computer vision and other areas, and most of the presented algorithms are simple and practical to implement.
1.1 Approximate Nearest Neighbor

Nearest Neighbor (NN) search is an important and widely studied problem which is of major importance in several areas. It is defined as follows. Given a collection $P$ of $n$ points in a $d$-dimensional space, the goal of NN is to build a data structure, that, given a query point $q$, quickly reports a point from $P$ which is closest to $q$. This problem has numerous applications in data mining, data compression, information retrieval, image and video databases, machine learning, pattern recognition, statistics and data analysis, and in many others. A particularly interesting and well-studied instance is where the data points live in a $d$-dimensional space under some (e.g., Euclidean) distance function. Typically in the aforementioned applications, each object of interest (document, image, etc.) is represented as a point in $\mathbb{R}^d$. In this high dimensional space, each dimension corresponds to a feature of these objects and the distance metric between the points is used to measure the similarity between the corresponding objects. The basic problem then is to perform indexing or similarity searching for query objects. The number of features (i.e., the dimensionality) ranges anywhere from tens to millions. For example, one can represent a $1000 \times 1000$ image as a vector in a 1,000,000-dimensional space, one dimension per pixel.

There are several efficient algorithms known for the case when the dimension $d$ is low (e.g., up to 10 or 20). The first such data structure, called $kd$-trees was introduced in 1975 by Jon Bentley [Ben75], and remains one of the most popular data structures used for searching in multidimensional spaces. Many other multidimensional data structures are known, see [Sam05] for an overview. However, despite decades of intensive effort, the current solutions suffer from either space or query time that is exponential in $d$. In fact, for large enough $d$, in theory or in practice, they often provide little improvement over a linear time algorithm that compares the query to each point from the database. This phenomenon is often called “the curse of dimensionality”.

Fortunately, researchers have been able to overcome the running time bottleneck by using approximation (e.g., [AMN+94, Kle97, IM98, KOR00, Har01, KL04, DIIM04, CR10, Pan06, AC06], see also [SDI08, Ind04]). In this formulation, the goal is to
report a point whose distance to \( q \) is within a \((1 + \varepsilon)\)-factor of the optimal distance, and \((1 + \varepsilon)\) is called the \textit{approximation factor} [IM98, HIM12]. The insight is that in many cases, an approximate nearest neighbor is almost as good as the exact one. In particular, if the distance measure accurately captures the notion of similarity between the objects, then small differences in the distance should not matter for the purpose of the user quality.

\textbf{Near Neighbor.} The ANN problem is often reduced to several instances of the \textit{Approximate Near Neighbor} problem, which is the decision variant of the approximate nearest neighbor problem. In the \textit{Near Neighbor} problem, a threshold parameter \( r \) is also given in advance and the goal is to report any point within distance \( r \) of the query point \( q \), if there exists such a point. In the approximate near neighbor problem, the goal is to output any point within distance \((1 + \varepsilon)r\) of the query point \( q \), if there exists any point within distance \( r \) of \( q \). Efficient solutions exist for this problem in high dimensions, and in particular, several data structures with query time of \((d + \log n + 1/\varepsilon)O(1)\) using \(n^{(1/\varepsilon)O(1)}\) space are known. [IM98, KOR00, Har01].

\textbf{Locality Sensitive Hashing.} Another popular approach for the nearest neighbor problems proposed by Indyk and Motwani, is based on a hashing technique called \textit{Locality Sensitive Hashing} (LSH) [IM98, HIM12]. In order to retrieve the close points to the query, LSH uses several hash functions randomly chosen from a family of hash functions \( \mathcal{H} \) and then hashes all the points using the chosen hash functions. This family \( \mathcal{H} \) has the key property that if two points are closer to each other, the probability that they are hashed to the same bucket (under a randomly chosen hash function from \( \mathcal{H} \)) is higher than if the two points are farther apart. Thus, for a query point, it can be shown that for an appropriate setting of the parameters, the set of points that are hashed to the same bucket as the query point by at least one of the hash functions, contains an approximate near neighbor to the query. This family \( \mathcal{H} \) is known to exist for several important metrics. For example, for the \( L_1 \) (i.e., Manhattan) distance, this algorithm finds the \((1 + \varepsilon)\)-ANN with query time \( \tilde{O}(dn^\rho) \),
and preprocessing/space $O(dn + n^{1+\rho})$ where $\rho = O((1+\varepsilon)^{-1})$ [IM98], and $\widetilde{O}$ hides a factor that is polynomial in $\log n$. For the $L_2$ (i.e., Euclidean) norm, this improves to $\rho = O((1+\varepsilon)^{-2})$ [AI08]. Moreover, there has been recent improvements along this approach, by considering data dependent hashing, where the randomization depends on the data points themselves [AINR14, AR15].

**Results.** Despite the extensive research on this topic, the basic formulation of nearest neighbor suffers from significant limitations while encountering some of the very natural applications and extensions of the basic problem. In what follows, we will describe those limitations, and our approaches to overcome them.

### 1.1.1 Robust Nearest Neighbor Search

Despite the ubiquity of ANN methods, the vast majority of current algorithms suffer from significant limitations when applied to data sets with corrupted, noisy, irrelevant or incomplete data. This is unfortunate since in the real world, rarely can one acquire data without some noise embedded in it. This could be because the data is based on real world measurements, which are inherently noisy, or the data describes complicated entities and properties where only part of it is relevant for the task at hand.

In Chapter 2, we will address this issue by formulating and solving a variant of the NN problem that allows for some data coordinates to be arbitrarily corrupted. Given a parameter $k$, the $k$-Robust Nearest Neighbor for a query point $q$, is a point $p \in P$ whose distance to the query point is minimized subject to ignoring a set of $k$ coordinates. We present a sampling technique which achieves a bi-criteria approximation for this problem: if the distance to the nearest neighbor after ignoring $k$ coordinates is $r$, the data-structure returns a point that is within a distance of $O(r)$ after ignoring $O(k)$ coordinates.
1.1.2 Nearest Line Search

The ANN problem generalizes naturally to the case where the database objects are more complex than simple points. Perhaps the simplest generalization is where the data items are represented not by points, but by lines or higher-dimensional flats (affine subspaces). Lines and low-dimensional flats are used to model data under linear variations [BHJM07], i.e., data sets where the objects are linearly dependent on a small set of unknown parameters. For example, images under varying light intensity can be modeled by varying the light gain parameter; thus the space of resulting images spans a one-dimensional line.

In Chapter 3, we will study the Approximate Nearest Line Search (NLS) problem. Given a set \( L \) of \( n \) lines in the high dimensional Euclidean space \( \mathbb{R}^d \), the goal is to build a data structure that finds a \((1 + \varepsilon)\) approximate closest line to any given query point \( q \). We present the first high-dimensional data structure for this problem with poly-logarithmic query time and polynomial space, and the achieved bounds match, up to polynomials, the bounds for the approximate nearest neighbor problem for point sets.

1.1.3 Sparse Linear Regression: Nearest Induced Flat

The Nearest Line Search problem can be generalized to higher dimensional objects. One such extension is the Nearest Linear Induced Flat (NLIF) problem, where given a set \( P \) of \( n \) points in \( d \) dimensions and a \( d \)-dimensional vector \( q \), the task is to find the closest \( k \)-dimensional sub-space spanning a subset of \( k \) points in \( P \), to the query point \( q \). This problem is equivalent to the well-known Sparse Linear Regression (SLR) problem, where the goal is to find a sparse linear model explaining a given set of observations. Formally, we are given a matrix \( M \in \mathbb{R}^{d \times n} \), and a vector \( q \in \mathbb{R}^d \), and we would like to find a vector \( v \) that is \( k \)-sparse (has at most \( k \) non-zero entries), minimizing \( \|q - Mv\| \). In the approximate variant, the algorithm is allowed to output a \( k \)-sparse vector \( v' \) such that \( \|Mv' - q\| \) is within a factor of \((1 + \varepsilon)\) of the optimum.

In Chapter 4, we present approximation algorithms for several sparse variants
of the problem with query time $\tilde{O}(n^{k-1})$. For small $k$, e.g., 2 or 3, this provides a significant improvement over the naive algorithm that exhaustively searches all $\binom{n}{k}$ subsets. For $k = d - 1$, this matches up to polylogarithmic factors, the lower bound that relies on the *affinely degenerate conjecture* (i.e., deciding if $n$ points in $\mathbb{R}^d$ contains a subset of $d + 1$ points that are contained in a hyperplane takes $\Omega(n^d)$ time). We also show that, under a natural complexity-theoretic conjecture, relying on Hopcroft’s problem and the $k$-sum problem, an exponential dependence on $k$ in the running time of the algorithm is unavoidable.

### 1.1.4 Simultaneous Nearest Neighbor Search

Many search applications involve dealing with multiple related queries. For example, suppose that we want to match features extracted from one image to features extracted from a large dataset of images, and moreover we want the matched features to come from the same unknown image. We address these types of applications by formulating *Simultaneous Nearest Neighbor* problem, where we have $k$ simultaneous query points $q_1, \ldots, q_k$ and the goal is to find $k$ points $p_1, \ldots, p_k$ in $P$, such that, first, $p_i$ is close to $q_i$, and second, $p_1, \ldots, p_k$ are compatible. In the image example above, a simultaneous NN query would jointly match all features from an image $I$ (these features are the queries), to features in the same unknown image $J$ (this is the compatibility condition) in the dataset. We can also encourage spatial consistency: for example, features that are near each other in $I$ should be matched to features that are near each other in $J$. The compatibility can be specified by an underlying graph $G$ (e.g., a complete graph shows that all pairs of $p_i$’s in the returned solution should be compatible). We can formalize the problem more precisely as:

$$\arg\min_{p_1, \ldots, p_k \in P} \sum_{i=1}^{k} \text{dist}(q_i, p_i) + \sum_{i,j \in E_G} \text{dist}(p_i, p_j) \quad (1.1)$$

In Chapter 5, we propose and analyze a simple preprocessing algorithm which prunes the data set from size $n$ down to $k$, where for each query point $q_i$, we find
its (approximate) nearest neighbor point $\hat{p}_i$ in $P$; this can be done efficiently using existing ANN data-structures. We show that replacing $P$ with $\hat{P} = \{\hat{p}_1, \ldots, \hat{p}_k\}$ satisfies an $O(\log k / \log \log k)$-approximation guarantee. We prove that although there are instances for which this algorithm would result in approximation $\Omega(\sqrt{\log k})$, the approximation factor is only a constant for compatibility graphs frequently occurring in practice, e.g., 2D grids, 3D grids or planar graphs. Finally, we validate our theoretical results by experiments showing that the “empirical approximation factor” provided by the above approach is very close to 1.

1.2 Composable Coresets

One of the most popular approaches for processing massive data is to first extract a compact representation of the data and then perform further processing only on the extracted representation itself. This approach significantly reduces the cost of processing, communicating, and storing the data as the representation can be much smaller than the original data set. Typically, the representation provides a smooth tradeoff between its size and the representation accuracy. Several notions have been previously introduced to capture this concept, such as coresets [AHV04], $\epsilon$-kernels [AHV04], and mergeable summaries [ACH+12]. In this thesis, we introduce the notion of Composable Coresets which is one such representation. As we show in Chapter 6, it enables efficient solutions to a wide variety of massive data processing applications, including nearest neighbor search, streaming algorithms and distributed computations such as MapReduce. A formal definition of composable coresets, is as follows.

**Definition.** Let $S_1, \ldots, S_k$ be (possibly overlapping) data sets, and our goal is to solve an optimization problem on the union of these data sets $S = \bigcup_i S_i$. Subsets $T_1 \subset S_1, \ldots, T_k \subset S_k$ are called composable coresets if the optimal value of the problem over the union of these coresets $T = \bigcup_i T_i$ is a good approximation of the optimal value over the union of the data sets $S$. Furthermore, we require that the size of each coreset $T_i$ to be “small”.
Diversity maximization. Diversity-aware summarization and search has attracted significant attention over the last few years [GS09, ZMKL05, AK11, JSH04, YLAY09, WCO11, DP10, AAYIM13, LBX09]. The goal of this line of research is to design efficient methods for searching and summarizing large data sets in a way that preserves the diversity of the data. In most formulations, the summary is a subset of the original data of some predefined size (say $k$) that maximizes a certain diversity objective. For example one could require that the minimum distance between any pair of data points in the summary is as large as possible, i.e., the summary does not contain two “highly similar” items. Many other, more refined, diversity objectives have been studied.

Results. In this thesis, we show how to find efficient composable coresets for some of the basic diversity maximization problems, and rule out the existence of composable coresets for the well-studied coverage maximization problem which is also a notion used for diversity maximization.

This negative result led us to investigate the coverage problem, as well as the closely related Set Cover problem, in sublinear computational models. These are basic optimization problems of interest in machine learning and data mining [KJ13].

1.3 The Set-Cover Problem

The Set Cover problem is one of the classic tasks in combinatorial optimization. Given a universe set of $n$ elements $U = \{e_1, \ldots, e_n\}$ and a collection of $m$ sets $\mathcal{F} = \{S_1, \ldots, S_m\}$, the goal is to pick a minimum number of sets from the collection so as their union covers the whole universe of elements. That is, for each element $e \in U$, there should exist a set containing $e$ among the sets we have picked. This is a well-studied problem with applications in operations research [GW97, KK82, BCS09], information retrieval and data mining [SG09], learning theory [KV94], web host analysis [CKT10], and many others.

Although the problem of finding an optimal solution is NP-hard, a natural greedy
algorithm which iteratively picks the “best” remaining set (the set that covers the most number of uncovered elements) is widely used. The algorithm finds a solution of size at most $k \ln n$ where $k$ is the optimum cover size, and can be implemented to run in time and space linear in the input size.

However, the input size itself could be as large as $\Theta(mn)$, so for large data sets even reading the input might be infeasible. Moreover, due to its sequential nature, the greedy algorithm does not scale very well to massive data sets (e.g., see Cormode et al. [CKW10] for an experimental evaluation). This difficulty has motivated a considerable research effort whose goal was to design algorithms that are capable of handling large data efficiently on modern architectures.

Of particular interest are data stream algorithms, which compute the solution using only a small number of sequential passes over the data using a limited memory, and the sublinear time algorithms which compute the solution by making few queries to the input to get partial information about it. In this thesis, we study this problem in these two sublinear models of computation.

### 1.3.1 Streaming and the Distributed Algorithms

In the streaming set cover problem defined by Saha and Getoor [SG09], the set of elements is stored in the memory in advance; the sets in the collection $\mathcal{F}$ are stored consecutively in a read-only repository and an algorithm can access the sets only by performing sequential scans of the repository. However, the amount of read-write memory available to the algorithm is limited, and in particular smaller than the input size (which could be as large as $\Omega(mn)$, as each of the $m$ sets might contain $\Omega(n)$ elements). The objective is to design an efficient approximation algorithm that performs few passes over the data, and uses as little memory as possible.

In Chapter 7, we present the first streaming algorithm that makes a constant number of passes and uses sublinear space for logarithmic approximation. We provide an $O(1/\delta)$-pass algorithm with a strongly sublinear $\tilde{O}(mn^\delta)$ space and $O((\ln n)/\delta)$ approximation. We partially complement this result by showing that for the exact algorithm, the tradeoff between the number of passes and space exhibited by our
algorithm is tight.

Moreover, in a series of work [DIMV14, HIMV16, AKL16, Ass17], this algorithm is shown to be tight. In particular, [AKL16] showed that in a single pass regime, our algorithm generalizes to a broader class of covering problems and that it achieves an optimal tradeoff. Also, [Ass17] showed that this trade-off is tight for multiple-pass algorithms even if the approximation is logarithmic.

### 1.3.2 Sublinear Time Algorithms

In the sublinear time model, the algorithm is allowed to query the input by asking for either the $i$th set containing a given element, or the $j$th element in a given set. This problem was initially studied in [NO08, YYI12], where the algorithm only provided a mixed multiplicative/additive guarantee. The dependence of the running time on the maximum set size was also exponential, and it only output the approximate cover size and not the cover itself.

In Chapter 8, we present two algorithms with sub-linear number of queries. First, we show that the streaming algorithm presented in Chapter 7 can be adapted so that it returns an $\alpha$-approximate cover using $\tilde{O}(m(n/k)^{1/(\alpha-1)} + nk)$ queries to the input, which could be quadratically smaller than $mn$. Second, we present a simple algorithm, with query complexity $\tilde{O}(mn/k)$, which is tailored to the case when the value of $k$ is large.

We complement the first result by proving that for lower values of $k$, the required number of queries is $\tilde{\Omega}(m(n/k)^{1/(2\alpha)})$, even for estimating the optimal cover size. Moreover, we prove that even checking whether a given collection of sets covers all the elements, would require $\Omega(nk)$ queries. These two lower bounds provide strong evidence that the first upper bound is almost tight for certain values of the parameter $k$. Lastly, we partially complement the second algorithm, by showing a lower bound of $\tilde{\Omega}(mn/k^2)$ if the approximation ratio is a small constant.
Part I

Approximate Nearest Neighbor
Chapter 2

Robust Nearest Neighbor Search

2.1 Introduction and Background

In this chapter, we formulate and solve a variant of the nearest neighbor problem that allows for some data coordinates to be arbitrarily corrupted. Given a parameter $k$, the $k$-Robust Nearest Neighbor for a query point $q$, is a point $x \in P$ whose distance to the query point is minimized ignoring the optimal set of $k$-coordinates (the term ‘robust’ ANN is used as an analogy to Robust PCA [CLMW11]). That is, the $k$ coordinates are chosen so that deleting these coordinates, from both $x$ and $q$ minimizes the distance between them. In other words, the problem is to solve the ANN problem in a different space (which is definitely not a metric), where the distance between any two points is computed ignoring the worst $k$ coordinates. To the best of our knowledge, this is the formulation of the Robust ANN problem.

This problem has natural applications in various fields such as computer vision, information retrieval, etc. In these applications, the value of some of the coordinates (either in the dataset points or the query point) might be either corrupted, unknown, or simply irrelevant. In computer vision, examples include image de-noising where some percent of the pixels are corrupted, or image retrieval under partial occlusion (e.g. see [HE07]), where some part of the query or the dataset image has been occluded. In these applications there exists a perfect match for the query after we ignore some dimensions. Also, in medical data and recommender systems, due to
incomplete data [SWC+09, CFV+13, WCNK13], not all the features (coordinates) are known for all the people/recommendations (points), and moreover, the set of known values differ for each point. Hence, the goal is to find the perfect match for the query ignoring some of those features.

For the binary hypercube, under the Hamming distance, the $k$-robust nearest neighbor problem is equivalent to the near neighbor problem. Indeed, there exists a point $x$ within distance $r$ of the query point $q$ if and only if $r$ coordinates can be ignored such that the distance between $x$ and $q$ is zero.

**Budgeted Version.** In the weighted generalization of the problem, the amount of uncertainty varies for each feature. In this model, each coordinate is assigned a weight $0 \leq w_i \leq 1$ in advance, which tries to capture the certainty level about the value of the coordinate ($w_i = 1$ indicates that the value of the coordinate is correct and $w_i = 0$ indicates that it can not be trusted). The goal is to ignore a set of coordinates $B$ of total weight at most 1, and find a point $x \in P$, such that the distance of the query to the point $x$ is minimized ignoring the coordinates in $B$. Surprisingly, even computing the distance between two points under this measure is NP-COMPLETE (it is almost an instance of Min Knapsack).

### 2.1.1 Results

We present a general reduction from the robust ANN problem to the “standard” ANN problem. This results in a bi-criterion constant factor approximation, with sublinear query time, for the $k$-robust nearest neighbor problem. For $L_1$-norm the result can be stated as follows. If there exists a point $q'$ whose distance to the query point $q$ is at most $r$ by ignoring $k$ coordinates, our algorithm would report a point $x$ whose distance to the query point is at most $O(r/\delta)$, by ignoring $O(k/\delta)$ coordinates. The query algorithm performs $O(n^3)$ of ANN queries in 2-ANN data structures, where $\delta \in (0, 1)$ is a prespecified parameter.

In this chapter, we present the above result in the somewhat more general settings of the $L_\rho$ norm. Further generalizations of this result to the budgeted variant and
improving the approximation to \((1 + \varepsilon)\) are presented in [HM17].

The new algorithms are clean and practical to implement – specifically, all bounds are polynomial in the dimension of the space, and the robustness parameter \(k\). Moreover, our results for the \(k\)-robust ANN hold for a wide range of the parameter \(k\), from \(O(1)\) to \(O(d)\).

2.1.2 Techniques

By definition of the problem, we cannot directly apply Johnson-Lindenstrauss lemma to reduce the dimensions (in the \(L_2\) norm case). Intuitively, dimension reduction has the reverse effect of what we want – it spreads the mass of a coordinate “uniformly” in the projection’s coordinates – thus contaminating all projected coordinates with noise from the “bad” coordinates.

The basic idea of our approach is to generate a set of random projections, such that all of these random projections map far points to far points (from the query point), and at least one of them projects a close point to a close point. Thus, doing ANN queries in each of these projected point sets, generates a set of candidate points, one of them is the desired robust ANN.

Our basic approach is based on a simple sampling scheme, similar to the Clarkson-Shor technique [CS89] and LSH [HIM12]. The projection matrices we use are \textit{probing} matrices. Every row contains a single non-zero entry, thus every row copies one original coordinate, and potentially scales it up by some constant.

A sketch of the technique. Consider the case where we allow to drop \(k\) coordinates. For a given query point \(q\), it has a robust nearest neighbor \(q^* \in P\), such that there is a set \(B\) of \(k\) “bad” coordinates, such that the distance between \(q\) and \(q^*\) is minimum if we ignore the \(k\) coordinates of \(B\) (and this is minimum among all such choices).

We generate a projection matrix by picking the \(i\)th coordinate to be present with probability \(1/(\alpha k)\), where \(\alpha\) is some constant, for \(i = 1, \ldots, d\). Clearly, the probability that such a projection matrix avoids picking the \(k\) bad coordinates is \((1 - \frac{1}{\alpha k})^k \approx e^{-1/\alpha}\).
In particular, if we repeat this process $\beta \ln n$ times, where $\beta$ is some constant, then the resulting projection avoids picking any bad coordinate with probability $\approx e^{-\beta \ln n/\alpha} = n^{-\beta/\alpha}$. On the other hand, imagine a “bad” point $x \in P$, such that one has to remove, say, $(\alpha/\beta)k$ coordinates before the distance of the point to the query $q$ is closer than the robust NN $q^*$ (when ignoring only $k$ coordinates). Furthermore, imagine the case where picking any of these coordinates is fatal – the value in each one of these bad coordinates is so large, that choosing any of these bad coordinates results in this bad point being mapped to a far away point. Then, the probability that the projection fails to select any of these bad coordinates is going to be roughly $(1 - \frac{1}{\alpha k})^{nk \ln n} \approx 1/n$. Namely, somewhat informally, with decent probability all bad points get mapped to faraway points, and the near point gets mapped to a nearby point. Thus, with probability roughly $\approx n^{-\beta/\alpha}$, doing a regular ANN query on the projected points, returns the desired ANN. As such, repeating this embedding $O(n^{\beta/\alpha} \log n)$ times, and returning the best ANN encountered returns the desired robust ANN with high probability.

**The good, the bad, and the truncated.** Ultimately, our technique works by probing the coordinates, trying to detect the “hidden” mass of the distance of a bad point from the query. The mass of such a distance might be concentrated in few coordinates (say, a point has $k + 1$ coordinates with huge value in them, but all other coordinates are equal to the query point) – such a point is arguably still relatively good, since ignoring slightly more than the threshold $k$ coordinates results in a point that is pretty close by.

On the other hand, a point where one has to ignore a large number of coordinates (say $2k$) before it becomes reasonably close to the query point is clearly bad in the sense of robustness. As such, our data-structure would classify points, where one has to ignore slightly more than $k$ coordinates to get a small distance, as being close.

To capture this intuition, we want to bound the influence a single coordinate has on the overall distance between the query and a point. To this end, if the robust nearest neighbor distance, to $q$ when ignoring $k$ coordinates, is $r$, then we consider capping
the contribution of every coordinate, in the distance computation, by a certain value, roughly, \( r/k \). Under this truncation, our data-structure returns a point that is \( O(r) \) away from the query point, where \( r \) is the distance to the \( k \)-robust NN point.

Thus, our algorithm can be viewed as a bicriterion approximation algorithm - it returns a point where one might have to ignore slightly more coordinates than \( k \), but the resulting point is constant approximation to the nearest-neighbor when ignoring \( k \) points.

In particular, a point that is still bad after such an aggressive truncation, is amenable to the above random probing. By carefully analyzing the variance of the resulting projections for such points, we can prove that such points would be rejected by the ANN data-structure on the projected points.

### 2.2 Preliminaries

#### 2.2.1 The Problem

**Definition 2.2.1.** For a point \( x \in \mathbb{R}^d \), let \( \pi = \text{sort}(x) \) be a permutation of \([d] = \{1, \ldots, d\}\), such that \( |x_{\pi(1)}| \geq |x_{\pi(2)}| \geq \cdots \geq |x_{\pi(d)}| \). For a parameter \( 1 \leq i \leq d \), the **\( i \)-tail of \( x \)** is the point \( x_{\setminus i} = (0, \ldots, 0, |x_{\pi(i+1)}|, |x_{\pi(i+2)}|, \ldots, |x_{\pi(d)}|) \). Note, that given \( x \in \mathbb{R}^d \) and \( i \), computing \( x_{\setminus i} \) can be done in \( O(d) \) time, by median selection.

Thus, given two points \( x, v \in \mathbb{R}^d \), their distance (in the \( L_{\rho} \)-norm), ignoring the \( k \) worst coordinates (which we believe to be noise), is \( \| (u - v)_{\setminus k} \|_\rho \). Here, we are interested in computing the nearest neighbor when one is allowed to ignore \( k \) coordinates. Formally, we have the following:

**Definition 2.2.2.** For parameters \( \rho, k \), a set of points \( P \subseteq \mathbb{R}^d \), and a query point \( q \in \mathbb{R}^d \), the **\( k \)-robust nearest-neighbor** to \( q \) in \( P \) is \( \text{nn}_{\setminus k}(P, q) = \arg\min_{x \in P} \| (x - q)_{\setminus k} \|_\rho \), which can be computed, naively, in \( O(d |P|) \) time.

**Definition 2.2.3.** For a point \( x \in \mathbb{R}^d \) and a set of coordinates \( I \subseteq [d] \), we define \( x_{\setminus I} \) to be a point in \( \mathbb{R}^{d-|I|} \) which is obtained from \( x \) by deleting the coordinates that are in \( I \).
Projecting a point set.

**Definition 2.2.4.** Consider a sequence $m$ of $\ell$, not necessarily distinct, integers $i_1, i_2, \ldots, i_\ell \in [d]$, where $[d] = \{1, \ldots, d\}$. For a point $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, its **projection** by $m$, denoted by $mx$, is the point $(x_{i_1}, \ldots, x_{i_\ell}) \in \mathbb{R}^\ell$. Similarly, the projection of a point set $P \subseteq \mathbb{R}^d$ by $m$ is the point set $mP = \{mx \mid x \in P\}$. Given a weight vector $w = (w_1, \ldots, w_d)$ the **weighted projection** by a sequence $m$ of a point $x$ is $m_w(x) = (w_{i_1}x_{i_1}, \ldots, w_{i_\ell}x_{i_\ell}) \in \mathbb{R}^\ell$.

Note, that can interpret $m$ as matrix with dimensions $\ell \times d$, where the $j$th row is zero everywhere except for having $w_{ij}$ at the $i_j$th coordinate (or 1 in the unweighted case), for $j = 1, \ldots, \ell$. This is a restricted type of a projection matrix.

**Definition 2.2.5.** Given a probability $\Pr > 0$, a natural way to create a projection, as defined above, is to include the $i$th coordinate, for $i = 1, \ldots, d$, with probability $\Pr$. Let $\mathcal{D}_{\Pr}$ denote the distribution of such projections.

Given two sequences $m = i_1, \ldots, i_\ell$ and $n = j_1, \ldots, j_\epsilon$, let $m|n$ denote the concatenated sequence $m|n = i_1, \ldots, i_\ell, j_1, \ldots, j_\epsilon$. Let $\mathcal{D}_{\Pr}^\ell$ denote the distribution resulting from concatenating $t$ such independent sequences sampled from $\mathcal{D}_{\Pr}$. (I.e., we get a random projection matrix, which is the result of concatenating $t$ independent projections.)

Observe that for a point $x \in \mathbb{R}^d$ and a projection $M \in \mathcal{D}_{\Pr}^\ell$, the projected point $Mx$ might be higher dimensional than the original point $x$ as it might contain repeated coordinates of the original point.

**Remark 2.2.6 (Compressing the projections).** Consider a projection $M \in \mathcal{D}_{\Pr}^\ell$ that was generated by the above process (for either the weighted or unweighted case). Note, that since we do not care about the order of the projected coordinates, one can encode $M$ by counting for each coordinate $i \in [d]$, how many time it is being projected. As such, even if the range dimension of $M$ is larger than $d$, one can compute the projection of a point in $O(d)$ time. One can also compute the distance between two such projected points in $O(d)$ times.
2.3 The $k$-Robust ANN under the $L_\rho$-Norm

In this section, we present an algorithm for approximating the $k$-robust nearest neighbor under the $L_\rho$-norm, where $\rho$ is some prespecified fixed constant (say 1 or 2). As usual in such applications, we approximate the $\rho$th power of the $L_\sigma$-norm, which is a sum of $\rho$th powers of the coordinates.

2.3.1 The Preprocessing and Query Algorithms

**Input.** The input is a set $P$ of $n$ points in $\mathbb{R}^d$, and a parameter $k$. Furthermore, we assume we have access to a data-structure that answers (regular) $c^{1/\rho}$-ANN queries efficiently, where $c$ is a quality parameter associated with these data-structures.

**Preprocessing.** Let $\alpha, \beta, \delta$ be three constants to be specified shortly, such that $\delta \in (0,1)$. We set $Pr = 1/(\alpha k)$, $t = \beta \ln n$, and $L = O(n^\delta \log n)$. We randomly and independently pick $L$ sequences $M_1, \ldots, M_L \in \mathcal{D}_{Pr}$. Next, the algorithm computes the point sets $P_i = M_i P$, for $i = 1, \ldots, L$, and preprocesses each one of them for $c^{1/\rho}$-approximate nearest-neighbor queries for the $L_\sigma$-norm ($c \geq 1$), using a standard data-structure for ANN that supports this. Let $D_i$ denote the resulting ANN data-structure for $P_i$, for $i = 1, \ldots, L$ (for example we can use the data structure of Indyk and Motwani [IM98, HIM12] for the $L_1/L_2$ cases).

**Query.** Given a query point $q \in \mathbb{R}^d$, for $i = 1, \ldots, L$, the algorithm computes the point $q_i = M_i q$, and its ANN $v_i$ in $P_i$ using the data-structure $D_i$. Each computed point $v_i \in P_i$ corresponds to an original point $x_i \in P$. The algorithm returns the $k$-robust nearest neighbor to $q$ (under the $L_\rho$-norm) among $x_1, \ldots, x_L$ via direct calculation.
2.3.2 Analysis

Points: Truncated, light and heavy

**Definition 2.3.1.** For a point \( x = (x_1, \ldots, x_d) \in \mathbb{R}^d \), and a threshold \( \psi > 0 \), let \( x_{\leq \psi} = (x'_1, \ldots, x'_d) \) be the \( \psi \)-**truncated** point, where \( x'_i = \min(|x_i|, \psi) \), for \( i = 1, \ldots, d \). In words, we max out every coordinate of \( x \) by the threshold \( \psi \). As such, the \( \psi \)-truncated \( L^\rho \)-norm of \( x = (x_1, \ldots, x_d) \) is \( \|x_{\leq \psi}\|_\rho = \left( \sum_{i=1}^d (\min(|x_i|, \psi))^\rho \right)^{1/\rho} \).

**Definition 2.3.2.** For parameters \( \psi \) and \( r \), a point \( x \) is **(\( \psi, r \))-light** if \( \|x_{\leq \psi}\|_\rho \leq r \). Similarly, for a parameter \( R > 0 \), a point is **(\( \psi, R \))-heavy** if \( \|x_{\leq \psi}\|_\rho \geq R \).

Intuitively a light point can have only a few large coordinates. The following lemma shows that being light implies a small tail.

**Lemma 2.3.3.** For a number \( r \), if a point \( x \in \mathbb{R}^d \) is \((r/k^{1/\rho}, r)-light\), then \( \|x_{\leq k}\|_\rho \leq r \).

**Proof.** Let \( \psi = r/k^{1/\rho} \), and let \( y \) be the number of truncated coordinates in \( x_{\leq \psi} \). If \( y > k \) then \( \|x_{\leq \psi}\|_\rho \geq \sqrt{y\psi^\rho} > \sqrt{k\psi^\rho} = k^{1/\rho}\psi = r \), which is a contradiction. As such, all the non-zero coordinates of \( x_{\leq k} \) are present in \( x_{\leq \psi} \), and we have that \( \|x_{\leq k}\|_\rho \leq \|x_{\leq \psi}\|_\rho \leq r \). \( \square \)

On the probability of a heavy point to be sampled as light, and vice versa

**Lemma 2.3.4.** Let \( x = (x_1, \ldots, x_d) \) be a point in \( \mathbb{R}^d \), and consider a random \( m \in \mathcal{D}_{\Pr} \), see 2.2.5. We have that \( \mathbb{E}[\|mx\|_\rho^\rho] = \Pr \|x\|_\rho^\rho \), and \( \mathbb{V}[\|mx\|_\rho^\rho] = \Pr(1 - \Pr) \|x\|_\rho^{2\rho} \).

**Proof.** Let \( X_i \) be a random variable that is \( |x_i|^\rho \) with probability \( \Pr \) and 0 otherwise. For \( Z = \|mx\|_\rho^\rho \), we have that \( \mathbb{E}[Z] = \sum_{i=1}^d \mathbb{E}[X_i] = \sum_{i=1}^d \Pr |x_i|^\rho = \Pr \|x\|_\rho^\rho \).

As for the variance, we have \( \mathbb{V}[X_i] = \mathbb{E}[X_i^2] - (\mathbb{E}[X_i])^2 = \Pr |x_i|^{2\rho} - \Pr^2 |x_i|^{2\rho} = \Pr(1 - \Pr) |x_i|^{2\rho} \). As such, we have \( \mathbb{V}[Z] = \sum_{i=1}^d \mathbb{V}[X_i] = \Pr(1 - \Pr) \sum_{i=1}^d |x_i|^{2\rho} = t\Pr(1 - \Pr) \|x\|_\rho^{2\rho} \). \( \square \)

**Lemma 2.3.5.** Let \( x \) be a point in \( \mathbb{R}^d \), and let \( \psi > 0 \) be a number. We have that \( \|x_{\leq \psi}\|_\rho^{2\rho} \leq \psi^\rho \|x\|_\rho^\rho \).
Proof. Consider the \( \psi \)-truncated point \( \mathbf{v} = \mathbf{x}_{\leq \psi} \), see 2.3.1. Each coordinate of \( \mathbf{v} \) is smaller than \( \psi \), and thus \( \|\mathbf{v}\|_{2\rho} = \sum_{i=1}^{d} |v_i|^{2\rho} \leq \sum_{i=1}^{d} \psi^\rho |v_i|^{\rho} \leq \psi^\rho \|\mathbf{v}\|_\rho^\rho \). \( \square \)

Lemma 2.3.6. Consider a sequence \( m \in \mathcal{D}_\mathbf{Pr} \). If \( \mathbf{x} \) is a \((r/k^{1/\rho}, R)\)-heavy point and \( R \geq (8\alpha)^{1/\rho} r \), then \( \Pr \left[ \|m\mathbf{x}\|_\rho^\rho \geq \frac{1}{2} \mathbb{E} \left[ \|m\mathbf{x}_{\leq \psi}\|_\rho^\rho \right] \right] \geq 1/2 \), where \( \psi = r/k^{1/\rho} \).

Proof. Consider the \( \psi \)-truncated point \( \mathbf{v} = \mathbf{x}_{\leq \psi} \). Since \( \mathbf{x} \) is \((r/k^{1/\rho}, R)\)-heavy, we have that \( \|\mathbf{x}\|_\rho^\rho \geq \|\mathbf{v}\|_\rho^\rho \geq R \). Now, setting \( Z = \|m\mathbf{v}\|_\rho^\rho \), and using 2.3.4, we have \( \mu = \mathbb{E}[Z] = \Pr \|\mathbf{v}\|_\rho^\rho \) and \( \sigma^2 = \mathbb{V}[Z] = \Pr(1 - \Pr) \|\mathbf{v}\|_{2\rho}^2 \leq \Pr(1 - \Pr)\psi^\rho \|\mathbf{v}\|_\rho^\rho \), by 2.3.5. Now, we have that \( \Pr[\|m\mathbf{x}\|_\rho^\rho \leq \mu/2] \leq \Pr[\|m\mathbf{v}\|_\rho^\rho \leq \mu/2] \). As such, by Chebyshev’s inequality, and since \( \Pr = 1/(\alpha k) \), if \( R \geq (8\alpha)^{1/\rho} r \), we have

\[
\Pr \left[ Z \leq \frac{\mu}{2} \right] \leq \Pr \left[ |Z - \mu| \geq \frac{\mu/2}{\sigma^2} \right] \leq \left( \frac{\sigma}{\mu/2} \right)^2 \leq \frac{\Pr(1 - \Pr)\psi^\rho \|\mathbf{v}\|_\rho^\rho}{\Pr^2 \|\mathbf{v}\|_{2\rho}^2 / 4} \leq \frac{4\psi^\rho}{\Pr \|\mathbf{v}\|_\rho^\rho} \leq \frac{4\alpha k r^\rho}{k R^\rho} \leq \frac{1}{2}.
\]

\( \square \)

Lemma 2.3.7. Let \( \mathbf{x} \) be a prespecified point. The probability that a sequence \( \mathbf{M} \) sampled from \( \mathcal{D}^k_{\mathbf{Pr}} \) does not sample any of the \( k \) heaviest coordinates of \( \mathbf{x} \) is \( \geq n^{-\beta/\alpha - 2\beta/(\alpha^2 k)} \approx 1/n^{\beta/\alpha} \) (for the simplicity of exposition, in the following, we use this rougher estimate).

Proof. Let \( S \) be the set of \( k \) indices of the coordinates of \( \mathbf{x} \) that are largest (in absolute value). The probability that \( m_i \) does not contain any of these coordinates is \( (1 - \Pr)^k \), and overall this probability is \( \nu = (1 - \frac{1}{\alpha k})^k t = (1 - \frac{1}{\alpha k})^{ak(\beta/\alpha) \ln n} \). Now, we have \( \nu \geq \exp(-\frac{\beta \ln n}{\alpha}) \left( 1 - \frac{1}{\alpha k} \right)^{ak(\beta/\alpha) \ln n} \geq n^{-\beta/\alpha} \exp(-\frac{2\beta \ln n}{\alpha^2 k}) = n^{-\beta/\alpha - 2\beta/(\alpha^2 k)} \), since, for any integer \( m \), we have \( (1 - 1/m)^{m-1} \geq 1/e \geq (1 - 1/m)^m \), and \( e^{-2x} \leq 1 - x \leq e^x \), for \( x \in (0, 1/2) \). \( \square \)

Lemma 2.3.8. Consider a point \( \mathbf{x} \) such that \( \|\mathbf{x}_{\leq k}\|_\rho \leq r \) (see 2.2.1). Conditioned on the event of 2.3.7, we have that \( \Pr \left[ \|M\mathbf{x}_{\leq k}\|_\rho^\rho \geq 2t \Pr r^\rho \right] \leq 1/2 \), where \( M \in \mathcal{D}^k_{\mathbf{Pr}} \).

Proof. By 2.3.4 for \( Z = \|M\mathbf{x}_{\leq k}\|_\rho^\rho \), we have \( \mu = \mathbb{E}[Z] = t \Pr \|\mathbf{x}_{\leq k}\|_\rho^\rho \leq t \Pr r^\rho \). The desired probability is \( \Pr[Z \geq 2\mu] \leq 1/2 \), which holds by Markov’s inequality. \( \square \)
Lemma 2.3.9. Let $R \geq (8\alpha)^{1/\rho}$. If $x$ is a $(r/k^{1/\rho}, R)$-heavy point, then
\[
\Pr\left[\|Mx\|^\rho \geq tPrR^\rho/8\right] \geq 1 - 2/n^{\beta/8}.
\]

Proof. Let $\psi = r/k^{1/\rho}$ and $v = x_{\leq \psi}$, and for all $i$, let $Y_i = \|m_i v\|^\rho$. By 2.3.6, with probability at least half, we have that $Y_i \geq E[Y_i]/2 \geq \Pr v/2 \geq \Pr R^\rho/2$. In particular, let $Z_i = \min(Y_i, \Pr R^\rho/2)$, and observe that $E[Z_i] \geq (\Pr R^\rho/2) \Pr Y_i \geq \Pr R^\rho/2 \geq \Pr R^\rho/4$. Thus, we have that $\mu = E[Z] = E[\sum_{i=1}^t Z_i] \geq tPrR^\rho/4$. Now set $U = tPrR^\rho/8$ and note that $\mu \geq 2U$. Now, by Hoeffding’s inequality, we have that
\[
\Pr[\|Mx\|^\rho \leq U] \leq \Pr[Z \leq U] \leq \Pr[|Z - \mu| \geq \mu - U] \leq 2\exp\left(-\frac{2(\mu - U)^2}{t(\Pr R^\rho/2)^2}\right)
\leq 2\exp\left(-\frac{8(\mu/2)^2}{tPr^2 R^{2\rho}}\right) \leq 2\exp\left(-\frac{8(tPrR^\rho/8)^2}{tPr^2 R^{2\rho}}\right) = 2\exp\left(-\frac{\beta \ln n}{8}\right) \leq \frac{2}{n^{\beta/8}}.
\]

Putting everything together.

Lemma 2.3.10. Let $\delta \in (0, 1)$ be a parameter. One can build the data-structure described in 2.3.1 with the following guarantees. For a query point $q \in \mathbb{R}^d$, let $q^* \in P$ be its $k$-robust nearest neighbor in $P$ under the $L_\rho$ norm, and let $r = \|q^* - q\|_\rho$. Then, with high probability, the query algorithm returns a point $v \in P$, such that $q - v$ is a $(r/k^{1/\rho}, O(r(c + 1/\delta)^{1/\rho}))$-light. The data-structure performs $O(n^\delta \log n)$ of $c^{1/\rho}$-ANN queries under $L_\rho$-norm.

Proof. We start with the painful tedium of binding the parameters. For the bad probability, bounded by 2.3.9, to be smaller than $1/n$, we set $\beta = 16$. For the good probability $1/n^{\beta/\alpha}$ of 2.3.7 to be larger than $1/n^\delta$, implies $n^\delta \geq n^{\beta/\alpha}$, thus requiring $\alpha \geq \beta/\delta$. Namely, we set $\alpha = \beta/\delta$. Finally, 2.3.9 requires $R \geq (8\alpha)^{1/\rho}r = (128/\delta)^{1/\rho}r$. Let $\lambda = \max(128/\delta, 16c)$ and let $R = \lambda^{1/\rho}r$.

For a query point $q$, let $q^*$ be its $k$-robust NN, and let $S$ be the set of $k$ largest coordinates in $z = q - q^*$. Let $E$ denote the event of sampling a projection $M_i \in \mathcal{D}_\rho$, that does not contain any of the coordinates of $S$. By 2.3.7, with probability $p \approx 1/n^{\beta/\alpha} = 1/n^\delta$, the event $E$ happens for the data-structure $D_i$, for any $i$. 

40
As such, since the number of such data-structures built is \( L = \Theta(n^\delta \log n) = O((\log n)/p) \), we have that, by Chernoff inequality, with high probability, that there are at least \( m = \Omega(\log n) \) such data structures, say \( D_1, \ldots, D_m \).

Consider such a data-structure \( D_i \). The idea is now to ignore the coordinates of \( S \) all together, and in particular, for a point \( x \in \mathbb{R}^d \), let \( x_{\setminus S} \in \mathbb{R}^{d-k} \) be the point where the \( k \) coordinates of \( S \) are removed (as defined in 2.2.3). Since by assumption \( \|z_{\setminus S}\|_\rho = \|z_{\setminus k}\|_\rho \leq r \), by 2.3.8, with probability at least half, the distance of \( M_i, q^* \) from \( M_i, q \) is at most \( \ell = (2tPr)^{1/\rho} \). Since there are \( \Omega(\log n) \) such data-structures, we know that, with high probability, in one of them, say \( D_1 \), this holds. By 2.3.9, any point \( x_{\setminus S} \) (of \( P \)), that is \( (r/k^{1/\rho}, R) \)-heavy, would be in distance at least \( \ell' = (tPr/8)^{1/\rho}R \geq (2ctPr)^{1/\rho}R = c^{1/\rho} \cdot \ell \) in the projection \( M_1 \) from the projected \( q \). Since \( D_1 \) is a \( c^{1/\rho} \)-ANN data-structure under the \( L_\rho \) norm, we conclude that no such point can be returned, because the distance from \( q \) to \( q^* \) in this data-structure is smaller than \( c^{1/\rho} \ell \leq \ell' \). Note that since for the reported point \( v \), the point \( v_{\setminus S} \) cannot be \( (r/k^{1/\rho}, \lambda^{1/\rho} R) \)-heavy, and that the coordinates in \( S \) can contribute at most \( k (r/k^{1/\rho})^\rho = r^\rho \). We conclude that the point \( v \) cannot be \( (r/k^{1/\rho}, (\lambda + 1)^{1/\rho} R) \)-heavy. Thus, the data-structure returns the desired point with high probability.

As for the query performance, the data-structure performs \( L \) queries in \( c^{1/\rho} \)-ANN data-structures.

This lemma translates to the following theorem using 2.3.3.

### 2.3.3 The Result

**Theorem 2.3.11.** Let \( P \subseteq \mathbb{R}^d \) be a set of \( n \) points with the underlying distance being the \( L_\rho \) metric, and \( k > 0, \delta \in (0, 1), \) and \( c \geq 1 \) be parameters. One can build a data-structure for answering the \( k \)-robust ANN queries on \( P \), with the following guarantees:

(A) Preprocessing time/space is equal to the space/time needed to store \( M = O(n^\delta \log n) \) data-structures for performing \( c^{1/\rho} \)-ANN queries under the \( L_\rho \) metric, for a set of \( n \) points in \( O((d/k) \log n) \) dimensions.
(B) The query time is dominated by the time it takes to perform $c^{1/\rho}$-ANN queries in the $M$ ANN data-structures.

(C) For a query point $q$, the data-structure returns, with high probability, a point $v \in P$, such that if one ignores $O(k(1/\delta + c))$ coordinates, then the $L_{\rho}$ distance between $q$ and $v$ is at most $O(r(1/\delta + c)^{1/\rho})$, where $r$ is the distance of the nearest neighbor to $q$ when ignoring $k$ coordinates. (Formally, $q - v$ is $(r/k^{1/\rho}, O(r((c + 1/\delta)^{1/\rho}))$-light.)

**Corollary 2.3.12.** Setting $c = 2$, the algorithm reports a point $v$ using $2^{1/\rho}$-ANN data-structures, such that if one ignores $O(k/\delta)$ coordinates, the $L_{\rho}$ distance between $q$ and $v$ is at most $O(r/\delta^{1/\rho})$. Formally, $q - v$ is $(r/k^{1/\rho}, O(r/\delta^{1/\rho}))$-light.
Chapter 3

Nearest Line Search

3.1 Introduction and Background

In this chapter, we consider the generalization of ANN to higher-dimensional flats (affine subspaces) and in particular lines. Despite the importance of this problem, there are only two results for the high dimensional variant of the problem, i.e., finding an approximate nearest \(k\)-flat [BHZA07, Mag02]. The first algorithm does not provide multiplicative approximation guarantees, although it does provide some alternative guarantees and has been validated by several experiments on computer vision data sets. The second algorithm, due to Magen, provides provable guarantees and fast query time of \((d + \log N + 1/\varepsilon)^{O(1)}\). However, the space requirement of the algorithm is quasi-polynomial, of the form \(2^{(\log N)^{O(1)}}\).

**Results.** Here, we consider the nearest subspace problem for the case of lines, i.e., 1-flats. Specifically, we are given a set \(L\) of \(N\) lines in the \(d\)-dimensional Euclidean space \(\mathbb{R}^d\). The goal is to build a data structure that, given a query point \(q \in \mathbb{R}^d\), if the closest line \(\ell^* \in L\) has distance \(r\) from the query point, then it reports a line \(\ell \in L\) with distance at most \((1 + \varepsilon)r\) from \(q\). We show:

**Theorem 3.1.1.** For any sufficiently small fixed \(\varepsilon > 0\), there exists a data structure using \(O(N + d)^{O(1/\varepsilon^2)}\) space, and an algorithm that given a query point \(q\), reports a
(1 + c\varepsilon)\text{-approximate nearest line with probability at least } 1 - \frac{6}{\log N} \text{ in time } (d + \log N + 1 / \varepsilon)^{O(1)}.

The performance of our data structure matches, up to polynomials, the performance of the best known data structure for the approximate point nearest neighbor (ANN) problem [KOR00, IM98]. In particular, it shows that the exponential dependence on the dimension can be avoided for this problem, which mimics the situation for the point data sets. Furthermore, the result is obtained via reductions to ANN, thereby showing that, in a sense, the problem over lines is not harder than the problem over points. \footnote{We note that the reductions to ANN used in this chapter are randomized. However, the best algorithm for ANN for point sets is randomized as well.} To the best of our knowledge, our result provides the first algorithm with poly-logarithmic query time and polynomial space for the approximate nearest neighbor problem for objects that are more complex than simple points.

**Related work.** In addition to the results in [BHZM07, Mag02], we also note that the dual variant of the problem has been considered in the literature. In the dual problem, the query is a \(k\)-dimensional flat for some small value of \(k\), and the data set consists of points. For the case when the query is a line, i.e., 1-flat, Andoni et. al. [AIKN09] provided a data structure with query time of \(O(d^2N^{0.5+t})\) and space of \(d^2N^{O(1/t^2+1/t^2)}\) for any desired \(t > 0\). This result was further generalized to higher dimensional flats [MNSS15]. We also note that in low dimensions several approximate algorithms for general polytope membership are known [AFM11]. However, those algorithms have query time and space exponential in the dimension and therefore, our results are specially interesting in high dimensions.

### 3.1.1 Overview of the Algorithms

In this section we give an overview of the difficulties arising when designing an algorithm for high dimensional lines, and the techniques that address them. On a high level our algorithms are obtained by reductions to approximate nearest neighbor over
a set of points. They employ three basic reduction types, also referred to as *modules*. They are depicted in Figures 3-1, 3-2 and 3-3.

**Net Module.** The simplest reduction proceeds by discretizing each line into a set of points by sampling regularly spaced points along the line. Clearly, given a sufficient sampling density, the nearest sample point identifies the approximately closest line. However, the required sampling density might be large, depending on the line configuration. First, in an unbounded region, the total number of samples we would need is infinite. Second, even in a bounded region, if the two lines are parallel and their distance is small, then the total number of samples we would need could be very large, since the query point could lie anywhere in the gap between the lines. Fortunately, in this case we can use a reduction of the second type.

![Figure 3-1: Net Module](image)

**Parallel Module.** We say that two lines $\ell_1, \ell_2$ are *almost parallel* if $\sin(\alpha) = O(\varepsilon)$ where $\alpha$ is the angle between $\ell_1$ and $\ell_2$. Consider the case where a set of lines are pairwise almost parallel. In this case we create a set of hyperplanes that are perpendicular to one of the lines. For each hyperplane, an ANN data structure is created for the point set which is the intersection of the lines with that hyperplane. Then, during the query time, the query point is also projected on the nearest hyperplane and uses the corresponding ANN data structure. It can be seen that the distance between the projected query point and the nearest point on the hyperplane is approximately equal to the distance between the actual query point and the corresponding line, so again the problem is reduced to ANN. However, the density of the hyperplanes needed to assure the accuracy depends on the angles between the lines. Also, for unbounded regions, unless the lines are exactly parallel, the total number of hyperplanes needed
is unbounded.

**Unbounded Module.** Both of the aforementioned reductions can only be used for bounded regions. To complement them, the third reduction is used to bound the search region. Let $B(o, r)$ be a ball intersecting all lines, and let $R = \frac{r}{\varepsilon \delta}$ where $\delta < \varepsilon$ so that $R > r$ ($\delta$ identifies the accuracy of the module). We define $P$ to be the set of intersections of the lines with the surface of the (larger) ball $B(o, R)$. Then, we build an ANN data structure for $P$.

![Figure 3-3: Unbounded Module](image)

Given a query point $q$, we proceed as follows. If the query point lies within the ball $B(o, R)$, then the problem can be solved using net module, i.e., by discretizing the parts of the lines contained in that ball. If the query point lies outside of $B(o, R)$, we project it on the surface and find the approximate nearest neighbor using the data structure. We show that the projection approximately preserves the order of the distances between the query $q$ and any pair of lines $\ell_1$ and $\ell_2$ s.t. $\sin \angle(\ell_1, \ell_2) \geq \delta$, i.e., whose angle has sin value greater than $\delta$. This allows us to either find a true approximate nearest line, or else find a line $\ell$ whose angle is close to the true nearest line $\ell^*$ (i.e. $\sin \angle(\ell^*, \ell) < \delta$). Then we can further restrict our search to lines whose angles are close to $\ell$ and use the Parallel module to find the approximate closest line among them.
Outline of Algorithms. Our main algorithm consists of two procedures: Almost Parallel Nearest Line Search (APNLS) and Nearest Line Search (NLS). APNLS solves the problem for the case where all lines are almost parallel to each other. On the other hand, NLS solves the problem for the general configuration of lines and uses APNLS as a subroutine. Both of these algorithms are recursive and use random sampling in the following manner. At each level of recursion, to solve the problem over a set $S$ of $n$ lines, we first solve the problem over a randomly chosen subset $T$ of lines of size $n/2$. With high probability, one of the $\log n$ closest lines to the query in the set $S$ is sampled in the set $T$. Since the recursive step returns a $(1 + O(\varepsilon))$ approximate closest line, the returned line (say $\ell \in T$) has the property that at most $\log n$ lines in $S \setminus T$ are much closer (closer by a factor of $(1 - \Theta(\varepsilon))$ to the query point than $\ell$. After the recursive step, the algorithm performs $\log n$ improvement steps. Given a line $\ell \in S$ with distance $x$ to the query point, the improvement step returns another line that is closer to the query point. Therefore, after $\log n$ steps, the algorithm finds the approximate closest line to the query point. The main difficulty lies in the improvement step. In the following, we sketch the improvement step for each of the two algorithms.

Improvement step of NLS. We are given a line $\ell \in S$ and the goal is to find another line $\ell' \in S$ which is closer to $q$ than $\ell$. Suppose that the nearest line $\ell^*$ is not almost parallel to $\ell$, i.e., $\sin \angle(\ell, \ell^*) = \Omega(\varepsilon)$. Let $x$ be the distance of $\ell$ to the query point $q$. Therefore all lines that are candidates for improvement intersect the ball $B(q, x)$. This ball is a bounded region, so if it was specified in advance, we could apply the net module. However, the query point $q$ and therefore the ball $B(q, x)$ is not known to the data structure in advance.

Instead, we construct a set of balls $B$ of polynomial size that depends on the input lines alone. The set $B$ has the following property: for each query point $q$ and a line $\ell$ with distance $x$ to $q$, there exists a ball of radius $x/\varepsilon^O(1)$ in $B$ that contains all lines that are not almost parallel to $\ell$ and their distance to $q$ is at most $x$. Furthermore, this ball can be retrieved in sublinear time. For each ball in $B$, we create a net module.
of polynomial size inside of it with sufficient density, as well as an unbounded module outside of it. Then \( q \) is either inside the ball (in which case we use the net module to find a closer line), or it lies outside of the ball (and we use the unbounded module to find the approximate closest line).

We remark that our construction of the set of balls, \( \mathcal{B} \), relies on the fact that \( \ell^* \) is not almost parallel to \( \ell \). If two lines are not almost parallel, then the lines diverge quickly from their “intersection” (the pair of points on the lines which have the minimum distance from each other). Thus if we know that the candidate lines for improvement are within distance at most \( x \) of \( q \) (and thus within distance at most \( 2x \) of \( \ell \)), then their intersections with \( \ell \) cannot be farther than \( O(\frac{x}{\varepsilon}) \) from \( q' \) (projection of \( q \) on \( \ell \)). This helps us in designing the set of balls \( \mathcal{B} \). More formally, we can have the lines sorted based on the position of their intersection with the line \( \ell \) and only retrieve the ones which are within \([-O(\frac{x}{\varepsilon}), O(\frac{x}{\varepsilon})]\) of \( q' \). However this property does not hold for the case of almost parallel lines.

**Improvement step of APNLS.** We first partition the space \( \mathbb{R}^d \) into \( O(n^2) \) parts using \( O(n^2) \) hyperplanes perpendicular to the line \( \ell \) so that we get the following properties. First, in each part, there is a unique ordering for the lines based on their projective distance \(^2\) from \( \ell \). Then no matter where the query falls, we can retrieve the set of relevant lines (which are the lines within distance \( O(x) \) of \( \ell \)). Second, in each part, as we move along \( \ell \) the projective distance of any pair of lines is monotone. As a consequence, in each part the minimum ball intersecting a set of lines has its center on the boundary of the part.

For each part we come up with a polynomial number of concentric balls of different radii. We build an unbounded module for each ball and a set of parallel modules for each region between any two successive balls. Given the query point, we find the two successive balls \( B_1, B_2 \) that the query point falls in between. We first use the unbounded module of the inner ball \( B_1 \) to differentiate between the candidate lines whose pairwise angle have sin value greater than some value \( \delta \) (where \( \delta \) depends on

---

\(^2\)We will define the notion of distance we use here, later in section 3.6.1.
the radius of $B_1$). Suppose the result of this phase is $\ell$. We then use the parallel modules to differentiate between the lines whose pairwise angle with $\ell$ have sin value smaller than $\delta$. Further we can show that for each part a polynomial number of unbounded modules and parallel modules suffices.

3.2 Definitions

Let $L$ be a set of $N$ lines in the $d$ dimensional Euclidean space $\mathbb{R}^d$ and let $q \in \mathbb{R}^d$ denote the query point.\footnote{We use $N = |L|$ to denote the total number of lines in the data set, while we use $n$ to denote the number of lines in a specific recursion level of our algorithms.} Let dist be the Euclidean distance. This induces the distance between two sets of points $S_1$ and $S_2$ defined as $\text{dist}(S_1, S_2) = \min_{p \in S_1, p' \in S_2} \text{dist}(p, p')$. We also define the distance between a point and a line $\text{dist}(p, \ell) = \min_{p' \in \ell} \text{dist}(p, p')$ and the distance between two lines $\text{dist}(\ell, \ell') = \min_{p \in \ell, p' \in \ell'} \text{dist}(p, p')$.

**Definition 3.2.1** (Approximate Nearest Neighbor). We define $\text{ANN}(P, \varepsilon)$, for any point set $P$ and error parameter $\varepsilon$, to be the Approximate Nearest Neighbor data structure constructed for $P$, with error parameter $\varepsilon$. Also we let $\text{ANN}_P(q)$ denote the approximate nearest point found by $\text{ANN}(P, \varepsilon)$ given the query point $q$. Moreover, we let $S_{\text{ANN}}(n, \varepsilon)$, $T_{\text{ANN}}(n, \varepsilon)$ and $C_{\text{ANN}}(n, \varepsilon)$ (respectively) denote the space bound, query time and data structure construction time (respectively) used by $\text{ANN}(P, \varepsilon)$ for $|P| = n$.

Furthermore, for a set of lines $S$, we define $\ell^*_S$ to be the line in $S$ that is closest to the query point $q$. Also for a point $p$, we use $\ell_p$, to denote the line that $p$ lies on, splitting the ties arbitrarily. It should be clear from the context what is the ground set for choosing $\ell_p$.

Let $B(o, r)$ denote the ball of radius $r$ centered at $o$, i.e., $B(o, r) = \{p \in \mathbb{R}^d | \text{dist}(o, p) \leq r\}$, and let $S(o, r)$ denote the surface of the ball $B(o, r)$, i.e., $S(o, r) = \{p \in \mathbb{R}^d | \text{dist}(o, p) = r\}$. We also refer to it as a sphere. Moreover we define the angle between two lines $\ell_1$ and $\ell_2$ to be the smaller of the two angles between them, and we denote it by $\text{angle}(\ell_1, \ell_2)$. 

49
Definition 3.2.2 ($\delta$-close, $\delta$-far, almost parallel). For any two lines $\ell_1$ and $\ell_2$, and any value of $\delta \leq \varepsilon$, we have the following definitions. $\ell_1$ and $\ell_2$ are $\delta$-close (strictly $\delta$-close, respectively) to each other if we have $\sin \angle(\ell_1, \ell_2) \leq \delta$ ($\sin \angle(\ell_1, \ell_2) < \delta$, respectively). Otherwise if $\sin \angle(\ell_1, \ell_2) \geq \delta$ ($\sin \angle(\ell_1, \ell_2) > \delta$, respectively), we say that the lines are $\delta$-far (strictly $\delta$-far, respectively) from each other. Also we say that the two lines are almost parallel if they are strictly $2\varepsilon$-close to each other.

For a set of lines $S$, a line $\ell$, and $\delta \leq \varepsilon$, we define $S_{\ell, \delta} \subset S$ to be the subset of all lines in $S$ that are strictly $\delta$-close to $\ell$. For any two lines $\ell_1$ and $\ell_2$, let the closest point $\text{Cl}_{\ell_1 \rightarrow \ell_2}$ be the point on the line $\ell_1$ which is closest to $\ell_2$. That is, $\text{Cl}_{\ell_1 \rightarrow \ell_2}$ and $\text{Cl}_{\ell_2 \rightarrow \ell_1}$ is the closest pair of points from the two lines. Note that the line $\text{Cl}_{\ell_1 \rightarrow \ell_2} \text{Cl}_{\ell_2 \rightarrow \ell_1}$ is perpendicular to both $\ell_1$ and $\ell_2$.

We also generalize the intersection of a set of lines $\ell_1, \ell_2, \ldots, \ell_k$, to be the smallest ball which touches all lines. Also, for any collection $S = \{S_1, \ldots, S_t\}$ of subsets of $\mathbb{R}^d$, and another subset $C$ of $\mathbb{R}^d$, we define their intersection as $S \cap C = \{S_1 \cap C, \ldots, S_t \cap C\}$. For example, if $S$ is a set of lines and $C$ is a ball, then the intersection is a set of segments, whereas if $C$ is a sphere then the intersection is a set of points.

3.3 Basic Modules

In this section, we present three modules which will be used several times later in our algorithms.

3.3.1 Unbounded Module

Intuition. For a set of lines $S$, this module addresses the case when the query point is far enough from the “intersection” of the lines. First suppose that all lines in $S$ pass through some point $o$. Then it is easy to find the closest line to the query point, as the closest line is the one with the smallest angular distance to the line $\overline{oq}$. Hence, for an arbitrary value of the radius $R$, we can build a ANN data structure for the points
of intersections of the lines with the sphere of radius $R$ centered at $o$, i.e., $S \cap S(o, R)$. Given the query point it is enough to project $q$ onto the sphere to get $q'$ and find the closest point in the ANN data structure.

For the general configuration of lines this intuition translates to the following module. For any set of lines in $S$, let $S(o, r)$ be any sphere which intersects all lines in $S$, i.e., $\forall \ell \in S : S(o, r) \cap \ell \neq \emptyset$. We construct an instance of ANN on the points obtained by intersecting each line with the larger sphere $S(o, R)$, where $R = \frac{r}{\varepsilon \delta}$ for some value of $\delta \leq \varepsilon$. That is, we construct $\text{ANN}(S \cap S(o, R), \varepsilon)$. Given any query $q$ outside of ball $B(o, R)$, let $q'$ be the projection of $q$ onto the sphere $S(o, R)$. The data structure and query processing algorithms are shown in Algorithms 1 and 2. Note that the input to Algorithm 1 is the smaller ball.

Let $p$ be the point returned by the $\text{ANN}_{S \cap S(o, R)}(q')$. Also let $\ell_p$ be the line in $S$ which corresponds to the point $p$, i.e., $p \in S(o, R) \cap \ell_p$. Then the following lemma holds.

```
Algorithm 1: Unbounded Module Data Structure

Input set of lines $S$
Input $B(o, r)$ which intersects all lines in $S$
Input parameter $\delta \leq \varepsilon$

1: $R \leftarrow \frac{r}{\varepsilon \delta}$
2: $P \leftarrow \emptyset$
3: for $\ell \in S$ do
4:   Add two intersections of $\ell \cap S(o, R)$ to $P$
5: end for
6: construct $\text{ANN}(P, \varepsilon)$
```

Lemma 3.3.1. Let $\ell_{\text{opt}} = \ell^*_S$ be the closest line to $q$. Then for sufficiently small $\varepsilon$ one of the following holds: either $\text{dist}(q, \ell_p)$ is within $1 + O(\varepsilon)$ factor of the closest distance $\text{dist}(q, \ell_{\text{opt}})$, or the closest line $\ell_{\text{opt}}$ is strictly $\delta$-close to $\ell_p$.

Proof. The intuition behind the proof is as follows. When the lines extend further than $\Theta(R) = \Theta(\frac{r}{\varepsilon \delta})$ from the origin $o$, if two lines are $\delta$-far from each other, then the
Algorithm 2: Unbounded Module Query Processing

<table>
<thead>
<tr>
<th>Input</th>
<th>a query point ( q ) such that ( q \notin B(o, R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>a line which is either an approximate closest line to ( q ), or is strictly ( \delta )-close to the closest line</td>
</tr>
</tbody>
</table>

1. \( q' \leftarrow \) projection of \( q \) onto the sphere \( S(o, R) \)
2. \( p \leftarrow \text{ANN}_P(q') \)
3. \( \ell_p \leftarrow \) the line in \( S \) which corresponds to \( p \)
4. Return \( \ell_p \)

Distance between them becomes larger than \( \Theta(R\delta) = \Theta(r/\varepsilon) \). This intuitively means that if we translate the lines inside the smaller ball \( S(o, r) \) such that they intersect the origin \( o \), this only changes their distance by a factor of \( O(r/(r/\varepsilon)) = O(\varepsilon) \). So this translation does not add too much error. Thus we can assume that all lines are crossing the origin \( o \). In this case projecting the query point on to the ball \( S(o, R) \) and computing the approximate point nearest neighbor on the ball results in an approximate nearest line. Now we state the formal proof.

We start by defining a set of notations as shown in Figure 3-4. Let \( r_q = \text{dist}(q, o) \) be the distance of the query point to the center of the balls \( o \), and let \( q' \) be the projection of \( q \) on to the sphere \( S(o, R) \). Let \( p \) be the point returned by the ANN and let \( \ell_p \) be the corresponding line to \( p \). Let \( u \) be the intersection of \( \ell_p \) with the sphere \( S(o, r_q) \) such that both \( p \) and \( u \) fall on the same side of \( S(o, r) \). Also let \( \ell_{\text{opt}} \) denote the actual closest line to the query point \( q \) and let \( s \) and \( t \) be its intersection with \( S(o, R) \) and \( S(o, r_q) \) respectively. Furthermore, let \( \ell'_p \) and \( \ell'_{\text{opt}} \) be the lines parallel to \( \ell_p \) and \( \ell_{\text{opt}} \) that pass through the origin, and let \( p', s', u', t' \) be their intersections with spheres \( S(o, R) \) and \( S(o, r_q) \). Finally, let \( \alpha_1 = \angle ou', \alpha_2 = \angle ot' \) be the angles between the line \( oq \) and the lines \( \ell_p \) and \( \ell_{\text{opt}} \) and let \( \alpha = \angle uot' \) be the angle between \( \ell_p \) and \( \ell_{\text{opt}} \) (or equivalently between \( \ell'_p \) and \( \ell'_{\text{opt}} \)).

Note that we can always choose \( t \) (and therefore the corresponding \( s \), \( t' \) and \( s' \)) such that \( \alpha_2 \leq \pi/2 \). It is easy to see that this holds for \( \ell_p \) approximately, i.e.,
\( \alpha_1 \leq 2\pi/3 \) for small enough \( \varepsilon \), otherwise the ANN would have chosen the other point of the intersection \( \ell_p \cap S(o, R) \) instead of choosing \( p \).

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure3-4.png}
\caption{Figure of Lemma 3.3.1}
\end{figure}

We assume that \( \ell_{opt} \) is \( \delta \)-far from \( \ell_p \), and prove that \( \frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq (1 + O(\varepsilon)) \). We achieve this via the following sequence of steps:

\[
\frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq \frac{\text{dist}(q, \ell_p')}{\text{dist}(q, \ell_{opt}')} (1 + O(\varepsilon)) \quad \text{(by Lemma 3.3.3)}
\]
\[
\leq \frac{\text{dist}(q, u')}{\text{dist}(q, t')} (1 + O(\varepsilon)) \quad \text{(by Lemma 3.3.6)}
\]
\[
\leq \frac{\text{dist}(q', p')}{\text{dist}(q', s')} (1 + O(\varepsilon)) \quad \text{(immediate)}
\]
\[
\leq (1 + O(\varepsilon)) \quad \text{(by Lemma 3.3.7)}
\]

Before proving the lemmas, we start by the following observation.

**Observation 3.3.2.** \( \text{dist}(p, p') \leq 2r \), (and similarly \( \text{dist}(u, u') \), \( \text{dist}(s, s') \), \( \text{dist}(t, t') \) \leq 2r).

**Proof.**

\[
\text{dist}(p, p') = 2R \sin \left( \frac{\text{pop}'}{2} \right) \leq 2R \sin \left( \text{pop} \right) \leq 2r
\]
Lemma 3.3.3. \[ \frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq \frac{\text{dist}(q, \ell'_p)}{\text{dist}(q, \ell'_{opt})}(1 + O(\varepsilon)) \]

Proof. Since the distance between the two pairs of parallel lines is at most \( r \), by triangle inequality
\[ \frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq \frac{\text{dist}(q, \ell'_p) + r}{\text{dist}(q, \ell'_{opt}) - r} \]

The following two claims prove that the value \( r \) in the numerator and denominator does not affect the ratio significantly.

Claim 3.3.4. \( \text{dist}(q, \ell'_p) \geq \frac{r}{3\varepsilon} \) for sufficiently small value of \( \varepsilon \).

Proof. By definition, \( \text{dist}(q, \ell_{opt}) \leq \text{dist}(q, \ell_p) \) and therefore by triangle inequality
\[ \text{dist}(q, \ell'_p) \leq \text{dist}(q, \ell_{opt}) + r \leq \text{dist}(q, \ell_p) + r \leq \text{dist}(q, \ell'_p) + 2r \tag{3.1} \]

Furthermore,
\[ \text{dist}(q, \ell'_p) + \text{dist}(q, \ell'_{opt}) = \text{dist}(q, u') \cos(\alpha_1/2) + \text{dist}(q, t') \cos(\alpha_2/2) \]
\[ \geq \frac{\text{dist}(q, u')}{2} + \frac{\text{dist}(q, t')}{2} \quad \text{(since } \alpha_1, \alpha_2 \leq 2\pi/3) \]
\[ \geq \frac{\text{dist}(t', u')}{2} \quad \text{(by triangle inequality)} \]
\[ \geq r \sin \alpha \geq R \sin \alpha \geq \frac{r}{\delta \varepsilon} \geq r/\varepsilon \quad \text{(noting that } \ell'_p \text{ and } \ell'_{opt} \text{ are } \delta\text{-far)} \]

Therefore, using Equation 3.1 we can infer that \( \text{dist}(q, \ell'_p) + 2r + \text{dist}(q, \ell'_p) \geq r/\varepsilon \) and thus \( \text{dist}(q, \ell'_p) \geq r/(3\varepsilon) \) for sufficiently small value of \( \varepsilon \).

Claim 3.3.5. \( \text{dist}(q, \ell'_{opt}) \geq \frac{r}{3\varepsilon} \) for sufficiently small value of \( \varepsilon \).

Proof. Suppose that, to the contrary, we have \( \text{dist}(q, \ell'_{opt}) < \frac{r}{3\varepsilon} \). Then
\[ \text{dist}(q, t') = \frac{\text{dist}(q, \ell'_{opt})}{\cos(\alpha_2/2)} \leq 2\text{dist}(q, \ell'_{opt}) \leq \frac{r}{4\varepsilon} \]
and thus, \( \text{dist}(q', s') \leq \frac{r}{4\varepsilon} \). However, this means that firstly, by triangle inequality and Observation 3.3.2, and for sufficiently small \( \varepsilon \)

\[
\text{dist}(q', s) \leq \text{dist}(q', s') + \text{dist}(s, s') \leq \frac{r}{4\varepsilon} + 2r \leq \frac{3r}{8\varepsilon}
\]

and secondly

\[
\text{dist}(q', p') \geq \text{dist}(p', s') - \text{dist}(q', s') \geq R \sin \alpha - \frac{r}{4\varepsilon} \quad \text{(since } \ell_p \text{ and } \ell_{opt} \text{ are } \delta\text{-far)} \Rightarrow
\]

\[
\text{dist}(q', p) \geq \text{dist}(q', p') - \text{dist}(p', p) \geq \frac{3r}{4\varepsilon} - 2r \geq \frac{2r}{3\varepsilon}
\]

which is a contradiction because \( \text{ANN} \) have chosen \( p \) over \( s \) and therefore it must be true that \( \text{dist}(q', p) \leq \text{dist}(q', s)(1 + \varepsilon) \), however we just proved that \( \text{dist}(q', p) \) is larger than \( \text{dist}(q', s) \) by a constant factor. This is a contradiction for sufficiently small value of \( \varepsilon \). Therefore the claim holds.

\[
\square
\]

Using the above two claims, we have

\[
\frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq \frac{\text{dist}(q, \ell_p) + r}{\text{dist}(q, \ell_{opt})} - r \leq \frac{\text{dist}(q, \ell_p)(1 + O(\varepsilon))}{\text{dist}(q, \ell_{opt})(1 - O(\varepsilon))} \leq \frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})}(1 + O(\varepsilon))
\]

\[
\square
\]

Lemma 3.3.6. \( \frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq \frac{\text{dist}(q, u')}{\text{dist}(q, t')}(1 + O(\varepsilon)) \)

\[
\text{Proof.}
\]

\[
\frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} = \frac{\text{dist}(q, u') \cos(\alpha_1/2)}{\text{dist}(q, t') \cos(\alpha_2/2)}
\]

Therefore it is enough to prove \( \frac{\cos(\alpha_1/2)}{\cos(\alpha_2/2)} \leq (1 + O(\varepsilon)) \). As proved earlier in Equation
dist\((q, ℓ_{opt})\) ≤ dist\((q, ℓ'_{p})\) + 2r ⇒

\(r_q \sin \alpha_2 ≤ r_q \sin \alpha_1 + 2r ⇒\)

\(\sin \alpha_2 - \sin \alpha_1 ≤ 2\varepsilon \delta ⇒\)

\((\sin \alpha_2 - \sin \alpha_1)(\sin \alpha_2 + \sin \alpha_1) ≤ 4\varepsilon \delta ⇒\)

\(\cos^2 \alpha_1 - \cos^2 \alpha_2 ≤ 4\varepsilon \delta ⇒\)

\((\cos \alpha_1 - \cos \alpha_2)(\cos \alpha_1 + \cos \alpha_2) ≤ 4\varepsilon \delta\)

Note that if \(\alpha_2 ≤ \alpha_1\), the bound on the ratio trivially holds. Otherwise, it is enough to prove that

\[
\frac{\cos(\alpha_1/2)}{\cos(\alpha_2/2)} = \sqrt{\frac{1 + \cos \alpha_1}{1 + \cos \alpha_2}} \leq \frac{1 + \cos \alpha_1}{1 + \cos \alpha_2} \leq 1 + O(\varepsilon)
\]

Now since \(\alpha_1 ≤ \alpha_2 ≤ \pi/2\), then \(\cos \alpha_1 + \cos \alpha_2 ≥ 0\). Therefore

\[
\cos \alpha_1 - \cos \alpha_2 ≤ \frac{4\varepsilon \delta}{\cos \alpha_1 + \cos \alpha_2}
\]

Now one of the two cases can occur. First, if \((\cos \alpha_1 + \cos \alpha_2) ≤ \delta\) then since \(0 ≤ \cos \alpha_2 ≤ \cos \alpha_1 ≤ 1\) and their sum is at most \(\delta\), then it means that their difference is also at most \(\delta\), i.e., \(\cos \alpha_1 - \cos \alpha_2 ≤ \delta ≤ \varepsilon\). Second if \((\cos \alpha_1 + \cos \alpha_2) > \delta\), then \(\cos \alpha_1 - \cos \alpha_2 ≤ \frac{4\varepsilon \delta}{\delta} ≤ 4\varepsilon\). Therefore in both cases we have that \(\cos \alpha_1 - \cos \alpha_2 ≤ 4\varepsilon\) and thus

\[
1 + \cos \alpha_1 ≤ 1 + \cos \alpha_2 + 4\varepsilon ⇒
\]

\[
\frac{1 + \cos \alpha_1}{1 + \cos \alpha_2} ≤ 1 + \frac{4\varepsilon}{1 + \cos \alpha_2} \leq 1 + 4\varepsilon
\]

\(\square\)

Lemma 3.3.7. \(\frac{\text{dist}(q', p')}{\text{dist}(q, s')} ≤ (1 + O(\varepsilon))\)
Proof. First note that

\[
\frac{\text{dist}(q', p')}{\text{dist}(q', s')} \leq \frac{\text{dist}(q', p) + \text{dist}(p, p')}{\text{dist}(q', s) - \text{dist}(s, s')} \leq \frac{\text{dist}(q', p) + 2r}{\text{dist}(q', s) - 2r}
\]

Claim 3.3.8. \(\text{dist}(q', s) \geq \frac{r}{2\varepsilon} (1 - O(\varepsilon))\)

Proof.

\[\text{dist}(q', p) + \text{dist}(q', s) \geq \text{dist}(q', p') - \text{dist}(p, p') + \text{dist}(q', s') - \text{dist}(s, s') \quad \text{(by triangle inequality)}\]
\[\geq \text{dist}(p', s') - 4r \quad \text{(by triangle inequality and Observation 3.3.2)}\]
\[\geq R \sin \alpha - 4r \geq \frac{r}{\varepsilon} - 4r \geq \frac{r}{\varepsilon} (1 - O(\varepsilon))\]

Also since the result of calling \(\text{ANN}\) is \(p\), it means that \(\text{dist}(q', p) \leq \text{dist}(q', s)(1 + \varepsilon)\) and therefore

\[\text{dist}(q', s)(2 + \varepsilon) \geq \text{dist}(q', p) + \text{dist}(q', s) \geq \frac{r}{\varepsilon} (1 - O(\varepsilon)) \quad \Rightarrow\]
\[\text{dist}(q', s) \geq \frac{r}{2\varepsilon(1 + O(\varepsilon))} (1 - O(\varepsilon)) \geq \frac{r}{2\varepsilon} (1 - O(\varepsilon))\]

\(\blacksquare\)

Using the above claim and the fact that \(\text{ANN}\) has chosen \(p\) over \(s\), i.e., \(\text{dist}(q', p) \leq \text{dist}(q', s)(1 + \varepsilon)\), we have

\[\frac{\text{dist}(q', p) + 2r}{\text{dist}(q', s) - 2r} \leq \frac{\text{dist}(q', s)(1 + \varepsilon) + 2r}{\text{dist}(q', s) - 2r} \quad \text{(Since \(\text{ANN}\) has chosen \(p\) over \(s\))}\]
\[\leq \frac{1 + \varepsilon + 2r/\text{dist}(q', s)}{1 - 2r/\text{dist}(q', s)}\]
\[\leq \frac{1 + \varepsilon + 4r}{1 - 2r/\text{dist}(q', s)} \leq \frac{1 + \varepsilon + 4\varepsilon/(1 - O(\varepsilon))}{1 - 4\varepsilon/(1 - O(\varepsilon))} \quad \text{(by Claim 3.3.8)}\]
\[\leq 1 + O(\varepsilon)\]

\(\blacksquare\)

Remark 3.3.9. By Definition 3.2.1, this module uses space \(S_{\text{ANN}}(m, \varepsilon)\) and has \(T_{\text{ANN}}(m, \varepsilon)\) query time, where \(m = 2|S|\) is the total number of points in the \(\text{ANN}\)
3.3.2 Net Module

This module is based on sampling points from the lines and uses the samples in an ANN data structure to find the approximate closest line. We state how the accuracy of the line we find depends on how finely we sample the lines. This module can only be used for the bounded regions, otherwise it needs infinite number of points to correctly represent the lines. Here, we only consider the bounded regions which are balls, because later in our algorithms we build net module for the bounded regions surrounded by balls.

For any set of lines $S$, let $B(o, r)$ be a ball which intersects each of the lines in $S$. We build an instance of ANN as follows. For each line $\ell \in S$, let $s_\ell = \ell \cap B(o, 2r)$ be the segment of the line $\ell$ which lies inside the twice larger ball $B(o, 2r)$. We then sample regularly spaced points from the segment $s_\ell$. The separation length (the distance of two consecutive samples on the segment) is equal to $x$, where $x$ is an arbitrary parameter. Let $P$ be the union of all the samples of the segments. During the preprocessing we construct $\text{ANN}(P, \varepsilon)$. Given a query $q \in B(o, r)$, we find its approximate nearest neighbor $p$ among the samples by calling $\text{ANN}_P(q)$. The data structure and query processing algorithm are shown in Algorithms 3 and 4.

Let $\ell_p$ be the line corresponding to $p$. Then the following lemma holds.

**Lemma 3.3.10.** Let $\ell_{opt} = \ell'_S$ be the closest line to $q$. Then for sufficiently small value of $\varepsilon$, either $\text{dist}(q, \ell_p) \leq \text{dist}(q, \ell_{opt})(1 + O(\varepsilon))$, or $\text{dist}(q, \ell_p) < x/\varepsilon$.

*Proof.* Suppose that $\text{dist}(q, \ell_p) \geq x/\varepsilon$. Let $q_{opt}$ be the projection of $q$ onto $\ell_{opt}$ and let $p_{opt}$ be the closest sampled point of $\ell_{opt}$ to $q_{opt}$. It can easily be checked that $q_{opt}$
**Algorithm 3: Net Module Data Structure**

<table>
<thead>
<tr>
<th>Input</th>
<th>set of lines $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$B(o,r)$ which intersects all lines in set $S$</td>
</tr>
<tr>
<td>Input</td>
<td>separation parameter $x$</td>
</tr>
</tbody>
</table>

1. $P \leftarrow \emptyset$
2. **for** $\ell \in S$ **do**
3. $s_\ell \leftarrow \ell \cap B(o, 2r)$
4. Sample the points from $s_\ell$ with separation $x$ and add them to $P$.
5. **end for**
6. construct $ANN(P, \varepsilon)$

**Algorithm 4: Net Module Query Processing**

<table>
<thead>
<tr>
<th>Input</th>
<th>a query point $q$ such that $q \in B(o, r)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>a line which is either an approximate closest line, or its distance to query is at most $x/\varepsilon$</td>
</tr>
</tbody>
</table>

1. $p \leftarrow ANN_P(q)$
2. $\ell_p \leftarrow$ the line in $S$ which corresponds to $p$
3. Return $\ell_p$

is within the ball $B(o, 2r)$ and therefore $\text{dist}(p_{opt}, q_{opt}) \leq x$. Then we have that

\[
\frac{x}{\varepsilon} \leq \text{dist}(q, \ell_p) \leq \text{dist}(q, p) \leq \text{dist}(q, p_{opt})(1 + \varepsilon) \quad (\text{since } ANN \text{ has chosen } p \text{ over } p_{opt})
\]

\[
\leq [\text{dist}(q, q_{opt}) + x](1 + \varepsilon) \quad (\text{by triangle inequality})
\]

\[
\leq \text{dist}(q, \ell_{opt})(1 + O(\varepsilon)) \quad (\text{since by previous line, } \text{dist}(q, q_{opt}) \geq \frac{x/\varepsilon}{(1+\varepsilon)} - x)
\]

and therefore, $\text{dist}(q, \ell_p) \leq \text{dist}(q, \ell_{opt})(1 + O(\varepsilon))$.

\[\square\]

**Remark 3.3.11.** By Definition 3.2.1, the space bound of this module is $S_{ANN}(m, \varepsilon)$ and its query time is $T_{ANN}(m, \varepsilon)$ where $m = |P| = |S| \left\lceil \frac{4r}{x} \right\rceil$ is the total number of
3.3.3 Parallel Module

*Intuition:* Suppose that all lines in $S$ are parallel. Then it is easy to find the closest line to the query point. Take an arbitrary hyperplane $g$ which is perpendicular to the lines. Then build an ANN data structure for the points of intersections of the lines with the hyperplane. Given the query point, it is enough to project $q$ onto the hyperplane $g$ to get $q'$. Then find the closest point in the ANN data structure and find the corresponding line. The following module generalizes this for the case where the lines are not exactly parallel but the case where all of them are $\delta$-close to some base line $\ell_b$.

Let $S$ be a set of lines, and let the base line $\ell_b \in S$ be a line in the set, such that each line $\ell \in S$ is $\delta$-close to $\ell_b$ for some parameter $\delta \leq \varepsilon$. Let $g$ be any hyperplane perpendicular to $\ell_b$. Moreover let $P$ denote the set of points which is the intersection of the lines in $S$ with the hyperplane $g$. The preprocessing proceeds by constructing $\text{ANN}(P, \varepsilon)$. Given the query point $q$, we project it on to the hyperplane to get the point $q'$. We then use the existing ANN data structure to get an approximate nearest neighbor $p$ of the point $q'$. The data structure and query processing algorithms are shown in Algorithm 5 and 6.

![Figure 3-5: An illustration of Lemma 3.3.12](image)

Let $\ell_p$ be the corresponding line to $p$. We then have the following lemma.
Algorithm 5: Parallel Module Data Structure

**Input** set of lines $S$
**Input** base line $\ell_b \in S$ s.t. all lines in $S$ are $\delta$-close to $\ell_b$
**Input** hyperplane $g$ which is perpendicular to $\ell_b$

1: $P \leftarrow \emptyset$
2: for $\ell \in S$ do
3: Add the point $\ell \cap g$ to the set $P$.
4: end for
5: construct $\text{ANN}(P, \varepsilon)$

Algorithm 6: Parallel Module Query Processing

**Input** a query point $q$ such that $\text{dist}(q, g) \leq D\varepsilon/\delta$
**Output** a line which is either the approximate closest line, or that $\text{dist}(q, \ell_p) \leq D$

1: $q' \leftarrow$ Projection of $q$ onto $g$
2: $p \leftarrow$ result of $\text{ANN}_{P}(q')$
3: $\ell_p \leftarrow$ the line in $S$ which corresponds to $p$
4: Return $\ell_p$

Lemma 3.3.12. Let $\ell_{opt} = \ell^*_S$ be the closest line to $q$ in the set $S$. Then for any value of $D$, if $\text{dist}(q, g) \leq D\varepsilon/\delta$, then either $\text{dist}(q, \ell_p) \leq \text{dist}(q, \ell_{opt})(1 + O(\varepsilon))$, or $\text{dist}(q, \ell_p) \leq D$.

**Proof.** Suppose that $\text{dist}(q, \ell_p) > D$, then we prove $\text{dist}(q, \ell_p)$ is within $(1 + O(\varepsilon))$ of the optimum. Let $h$ be the hyperplane parallel to $g$ which passes through $q$. Let $a$ and $b$ be the closest points on $\ell_p$ and $\ell_{opt}$ to $q$ respectively. Let $a_h, a_g, b_h, b_g$ be the intersection of the lines $\ell_p$ and $\ell_{opt}$ with hyperplanes $h$ and $g$ as shown in Figure 3-5. Furthermore, let $a_p, b_p$ be the projections of $a_h, b_h$ on to the hyperplane $g$. Since $\sin \angle(\ell_p, \ell_b) \leq \delta$ then

\[
\text{dist}(a_p, a_g) = \text{dist}(a_h, a_p)(\tan \angle(\ell_p, \ell_b)) \leq D\varepsilon/\delta \times \frac{\delta}{\sqrt{1 - \delta^2}} \leq \frac{D\varepsilon}{\sqrt{1 - \varepsilon^2}} \leq 2D\varepsilon
\]
for sufficiently small value of $\varepsilon$. For the same reason we have $\text{dist}(b_p, b_g) \leq 2D\varepsilon$. Also we have

$$\frac{\text{dist}(q, b_h)}{\text{dist}(q, b)} \leq \frac{1}{\cos(b_q b_h)} \leq \frac{1}{\sqrt{1 - \delta^2}} \leq (1 + \varepsilon) \quad (3.2)$$

again for sufficiently small value of $\varepsilon$. Now

$$D < \text{dist}(q, a) \quad (\text{by what we assumed})$$

$$\leq \text{dist}(q, a_h) \quad (\text{since } a \text{ is the closest point on } \ell_p \text{ to } q)$$

$$\leq \text{dist}(q', a_g) + 2D\varepsilon \quad (\text{by triangle inequality and that } \text{dist}(q, a_h) = \text{dist}(q', a_p))$$

$$\leq \text{dist}(q', a_g)(1 + O(\varepsilon)) \quad (\text{since by previous line, } \text{dist}(q', a_g) \geq D(1 - 2\varepsilon))$$

$$\leq \text{dist}(q', b_g)(1 + O(\varepsilon)) \quad (\text{since } a_g \text{ is the result of ANN})$$

$$\leq [\text{dist}(q, b_h) + 2D\varepsilon](1 + O(\varepsilon)) \quad (\text{by triangle inequality and that } \text{dist}(q, b_h) = \text{dist}(q', b_p))$$

$$\leq \text{dist}(q, b_h)(1 + O(\varepsilon)) \quad (\text{since by previous line, } \text{dist}(q, b_h) \geq D(1 - O(\varepsilon)))$$

$$\leq \text{dist}(q, b)(1 + O(\varepsilon)) \quad (\text{by Equation } 3.2)$$

Therefore, $\text{dist}(q, \ell_p)$ is within an $(1 + O(\varepsilon))$ factor away from the optimal distance $\text{dist}(q, \ell_{opt})$.

\[ \square \]

**Remark 3.3.13.** By Definition 3.2.1, the space bound of this module is $S_{\text{ANN}}(m, \varepsilon)$ and its query time is $T_{\text{ANN}}(m, \varepsilon)$ where $m = |P| = |S|$ is the total number of points we build the ANN data structure for.

### 3.4 General Case of NLS

This section provides a data structure and an algorithm which answers NLS queries in poly-logarithmic time. The algorithm is recursive. In each level of recursion with input lines $S$, the algorithm first constructs $T$ by sampling half of the lines in $S$ and recursively builds a data structure for $T$. After $O(\log N)$ levels of recursion, the
number of lines is small enough to be searched by brute force. At each level with high enough probability, \( T \) contains one of the \( \log n \) closest lines in \( S \) to the query point (where \( n = |S| \)). Then we start with the line returned as answer for \( T \) and for \( \log n \) rounds we try to find a closer line to \( q \). The second step is called *improvement* step.

The improvement step is as follows. Suppose the current line \( \ell \) has distance \( x \) to the query point. Then the closest line is either strictly \( \varepsilon \)-close to \( \ell \) or \( \varepsilon \)-far from \( \ell \). In the first case, the algorithm makes a call to Almost Parallel Nearest Line Search (APNLS) which finds the approximate closest line among a set of lines that are almost parallel. This case will be discussed in Section 3.6, for which we present an algorithm with poly-logarithmic time and polynomial space.

In the second case, each line \( \ell' \) which is closer to \( q \) than \( \ell \), must intersect \( B(q, x) \). As we show later, this means that \( \text{dist}(\ell, \ell') \leq 2x \) and that \( \text{Cl}_{\ell \rightarrow \ell'} \) is within \( O(x/\varepsilon) \) of the projection of \( q \) on \( \ell \). Thus we build a data structure which can retrieve all lines satisfying these two properties using binary search. We then prove that the closest ball touching all these lines has radius \( O(x/\varepsilon) \). Given the query point \( q \), if it lies "far enough" from this ball (that is, outside of a sufficiently larger ball \( B \)), then we can solve the problem using the unbounded module around \( B \). If the query lies inside \( B \), then we can use the net module. The corresponding line is either an approximate nearest line or its distance to \( q \) decreases by a factor of \( O(\varepsilon) \). Therefore, at most \( \log n \) such improvement steps are needed to obtain an approximate nearest line.

The analysis of the algorithms in this section are presented in Section 3.5

### 3.4.1 Data Structure

Algorithm 7 shows the preprocessing and the data structure construction procedure of NLS algorithm. The preprocessing procedure is randomized. First it selects half of the input lines randomly, and builds the same data structure recursively for the selected lines. Then, for each line \( \ell \) in the input set \( S \), it sorts the other lines in \( S \) by their distance to \( \ell \). This will help us later to retrieve the set of lines which are within a distance at most \( 2x \) from the given line \( \ell \). Let \( S_i \) be the set of the \( i \) lines in \( S \setminus \{\ell\} \) that are closest to \( \ell \). The algorithm sorts the lines in \( S_i \) based on the position of the
point of $\ell$ that is closest to them. That is, we orient $\ell$ arbitrarily in one of the two
two ways, and for all lines $\ell' \in S$, we sort them based on the position of $C_{\ell \to \ell'}$ along $\ell$
towards the chosen direction. This will help us retrieve all lines whose closest point
on $\ell$ lies within some range. Further, let $A$ be any sequence of consecutive lines (an
interval) in $S$. Let $A^-$ be the union of $\ell$ and the set of lines in $A$ that are $\varepsilon$-far from
$\ell$. We then find the smallest ball with a center in $\ell$ that intersects all lines in $A^-$. Denote this ball by $B(o_A, Y_A)$.

We now use the basic modules. First, we build an unbounded module for the set
$A^-$, the ball $B(o_A, Y_A)$ and the parameter $\varepsilon$ (note that, this module is used to handle
queries outside of the larger ball $B(o_A, Y_A/\varepsilon^2)$). Second, we build a net module for the
same set of lines $A^-$ and the larger ball $B(o_A, Y_A/\varepsilon^2)$ with the separation parameter $Y_A\varepsilon^3$. Finally, for each line $\ell$, we find all lines that are strictly $\varepsilon$-close to it (and
therefore strictly $(2\varepsilon)$-close to each other) and create the APNLS data structure for
them as described in Algorithm 9.

**Lemma 3.4.1.** The data structure uses $(N + d)^{O(1/\varepsilon^2)}$ space.

**Proof.** First of all we need $O(Nd)$ space to store the lines. At each step of the recursion
with $n = |S|$ points, there are a total of $n$ candidates for $\ell$ and for each of them there
are at most $n$ different $S_i$. For each $S_i$ there are $O(n^2)$ intervals $A$. For any such
$A$, the space used by the unbounded module is at most $S_{\text{ANN}}(2n, \varepsilon)$ (Remark 3.3.9 ),
and the space used by the net module is at most $S_{\text{ANN}}(n^4Y_A/\varepsilon^2, \varepsilon) = S_{\text{ANN}}(4n/\varepsilon^5, \varepsilon)$
(Remark 3.3.11). Therefore the total space used is $O(n^4S_{\text{ANN}}(4n/\varepsilon^5, \varepsilon))$. Moreover,
as will be proved later in Lemma 3.6.4, the space needed to build the APNLS data structure for each set is
$O(n^7/\varepsilon^3 \times S_{\text{ANN}}(n, \varepsilon))$ and since we call it for each line $\ell \in S$,
the total space used at each level of recursion is $O(n^4S_{\text{ANN}}(4n/\varepsilon^5, \varepsilon) + n^8/\varepsilon^3 \times S(n, \varepsilon))$. The recursion formula for the space at each level of recursion is

$$S(n) \leq S(n/2) + n^4S_{\text{ANN}}(4n/\varepsilon^5, \varepsilon) + O(n^8/\varepsilon^3 \times S_{\text{ANN}}(n, \varepsilon))$$

and therefore, the total space used by the algorithm is $O(N^4S_{\text{ANN}}(4N/\varepsilon^5, \varepsilon) + N^8/\varepsilon^3 \times S_{\text{ANN}}(N, \varepsilon))$. Since $S_{\text{ANN}}(N, \varepsilon)$ is of the form $O(N + d)^{O(1/\varepsilon^2)}$, therefore the lemma
**Algorithm 7: NLS Data Structure Construction**

**Input** a set of lines $S$

1: $T \leftarrow |S|/2$ lines chosen at random from $S$.
2: Recursively build the NLS data structure for $T$.
3: for each line $\ell \in S$ do
4: Sort the lines in $S \setminus \{\ell\}$ based on their distance from $\ell$ in a non-decreasing order.
5: for $i = 1$ to $|S| - 1$ do
6: $S_i \leftarrow$ the closest $i$ lines to $\ell$ in $S \setminus \{\ell\}$
7: Sort all lines in $S_i$ based on the position of the closest point $Cl_{\ell \rightarrow \ell'}$ on the line $\ell$.
8: for each of the $\binom{|S|}{2}$ intervals of lines $A$ do
9: $A^- \leftarrow \ell \cup$ (all lines $\ell' \in A$ that are $\varepsilon$-far from $\ell$)
10: $B(o_A, Y_A) \leftarrow$ the smallest ball that intersects all lines in $A^-$ and that $o_A \in \ell$
11: $UM_A \leftarrow$ Build unbounded module for lines in $A^-$, the ball $B(o_A, Y_A)$ and the parameter $\varepsilon$
12: $NM_A \leftarrow$ Build net module for lines in $A^-$, the ball $B(o_A, Y_A/\varepsilon^2)$, and the separation parameter $Y_A\varepsilon^3$
13: end for
14: end for
15: Build APNLS data structure (described in Section 3.6) for the set of lines $S_{\ell, \varepsilon}$
16: end for
Lemma 3.4.2. The data structure can be constructed in time $(N + d)^{O(1/\varepsilon^2)}$.

Proof. First note that each computation takes $O(d)$ time. As in the previous lemma, there are $n^4$ candidates for $A$, where $n = |S|$. For each such $A$, the smallest ball $B(o_A, Y_A)$ can be found using sweep-line technique in $O(n^2 \log n)$ time (the sweep proceeds along the line $L$ and the squared distance to other lines have quadratic form). Therefore using the same arguments as the previous lemma, the construction time is $O(dN^6 \log N + N^4 \mathcal{CT}_{\text{ANN}}(4N/\varepsilon^5, \varepsilon))$. Since the construction time of the APNLS by Lemma 3.6.5 is $O(N^7/\varepsilon^3 \times \mathcal{CT}_{\text{ANN}}(n, \varepsilon))$, the total construction time of NLS is $O(N^4 \mathcal{CT}_{\text{ANN}}(4N/\varepsilon^5, \varepsilon) + N^8/\varepsilon^3 \times \mathcal{CT}_{\text{ANN}}(N, \varepsilon))$ which given that $\mathcal{CT}_{\text{ANN}}(N, \varepsilon) = O(N + d)^{O(1/\varepsilon^2)}$, it is of the form $(N + d)^{O(1/\varepsilon^2)}$. □

3.4.2 Query Procedure

Given a query point $q$, Algorithm 8 describes how to find an approximate nearest line using the aforementioned data structure. The algorithm first checks whether the problem can be solved by brute force. That is, if the total number of points in the input set $S$ is at most $\log^3 N$ (where $N = |L|$ is the total number of points in the database), then the algorithm simply computes the distances from the query to all lines in $S$, finds the closest line and reports it.

Otherwise, it first runs the procedure recursively for the subset $T$, and takes the reported line $L$ as a starting point. The algorithm performs $\log |S|$ improvement steps, replacing the current line with a closer one. During each iteration, the algorithm first finds the approximately closest line to $q$ among the lines that are strictly $\varepsilon$-close to $L$, i.e., $S_{L, \varepsilon}$ and stores it in $L_{\text{close}}$. This is done by invoking APNLS$(S_{L, \varepsilon}, q)$.

Let $x$ be the distance from the query to the current line $L$ and let $q_L$ be the projection of $q$ onto $L$. The algorithm retrieves all potential candidate lines for improvement in the set $A$. That is, it gets all lines $L'$ which are no farther than $2x$ from $L$ and that the closest point of $L$ to them, i.e., $\text{Cl}_{L, \varepsilon} L'$, lies within a distance of $3x/\varepsilon$ of $q_L$. The algorithm then uses one of the two modules corresponding to the set of lines in $A$.
It checks whether the unbounded module can be used to retrieve the approximate nearest line among \( A^- \) to \( q \), i.e., \( q \notin B(o_A, Y_A/\varepsilon^2) \). If so, it uses the module followed by a call to the APNLS of the found line and sets the value of \( \ell_{far} \). Otherwise the point \( q \) lies inside the ball \( B(o_A, Y_A/\varepsilon^2) \) and thus it uses the net module in order to retrieve a better line and updates the value of \( \ell_{far} \) with it. Note that \( \ell_{far} \) shows the candidate among the set of lines in \( A \) which are \( \varepsilon \)-far from \( \ell \). Then at the end of the iteration, the value of \( \ell \) is updated with the best of \( \ell_{close} \) and \( \ell_{far} \) if any of them can improve \( \ell \).

**Lemma 3.4.3.** The query processing algorithm runs in time \((\log N + d + 1/\varepsilon)^O(1)\).

**Proof.** First note that each computation takes \( O(d) \) time. Also note that in the whole execution of the NLS algorithm, we call the APNLS at most \( 2\log^2 N \) times (\( 2\log N \) times per recursion level). Therefore the total running time of APNLS by Lemma 3.6.6 is at most

\[ O(d \log^5 N + \log^4 N \times T_{ANN}(4N/\varepsilon^4, \varepsilon)) \]

Now, at a recursion level with \( n \) lines (\( n = |S| \)), if we have \( n \leq \log^3 N \), the running time is \( d \log^3 N \). Otherwise, the longest path of the execution of the algorithm consists of \( \log n \) iterations and for each of them we run the net module which by Remark 3.3.11 takes \( T_{ANN}(4n/\varepsilon^5, \varepsilon) \) time. The recursion formula for the running time at each level of recursion is

\[ T(n) \leq T(n/2) + \log n \times T_{ANN}(4n/\varepsilon^5, \varepsilon), \quad T(\log^3 N) = d \log^3 N \]

and therefore since \( T_{ANN}(4n/\varepsilon^5, \varepsilon) \) is poly-logarithmic in \( n \) and polynomial in \( 1/\varepsilon \) and \( d \), the total running time of this part is

\[ d \log^3 N + \log^2 N \times T_{ANN}(4N/\varepsilon^5, \varepsilon) \]
Algorithm 8: NLS Query Processing

**Input** a set of lines $S$, and the query point $q$

**Output** the approximate nearest line $\ell \in S$ to the query point $q$

1: if $|S| \leq \log^3 N$ then
2: return the nearest line found by the brute force.
3: end if
4: $T \leftarrow$ the set of lines sampled during the preprocessing stage
5: $\ell \leftarrow$ the result of executing NLS($T$, $q$)
6: for $(\log |S|)$ times do
7: $\ell_{close} \leftarrow$ the result of running APNLS($S_{\ell,\varepsilon}$, $q$)
8: $x \leftarrow \text{dist}(q, \ell)$
9: $q_{\ell} \leftarrow$ the projection of $q$ on to $\ell$
10: Use binary search to find the largest $i$ such that for all lines $\ell' \in S_i$, we have $\text{dist}(\ell', \ell) \leq 2x$
11: Use binary search to find the largest interval of lines $A$ in $S_i$, s.t. $\forall \ell' \in A: \text{dist}(q_{\ell}, \text{Cl}_{\ell \rightarrow \ell'}) \leq 3x/\varepsilon$
12: if $A$ is not empty then
13: if $q \notin B(o_A, Y_A/\varepsilon^2)$ then
14: $\ell_r \leftarrow$ result of the unbounded module $UM_A(q)$
15: $\ell_{far} \leftarrow$ the best of $\ell_r$ and APNLS($S_{\ell,\varepsilon}$, $q$).
16: else
17: $\ell_{far} \leftarrow$ result of the net module $NM_A(q)$.
18: end if
19: end if
20: $\ell \leftarrow$ the best of $\ell$, $\ell_{close}$ and $\ell_{far}$
21: end for
22: return $\ell$
Therefore the total running time is

\[ O(d \log^5 N + \log^4 N \times T_{\text{ANN}}(4N/\varepsilon^4, \varepsilon) + \log^2 N \times T_{\text{ANN}}(4N/\varepsilon^5, \varepsilon)) \]

Since \( T_{\text{ANN}}(N, \varepsilon) \) is of the form \((\log N + d + 1/\varepsilon)^{O(1)}\), the lemma holds.

\[ \square \]

### 3.5 Analysis of NLS Algorithm

**Definition 3.5.1.** An invocation \( \text{NLS}(L, q) \) is called successful if it satisfies the following two properties. First, in all recursive calls involving a set of lines \( S \), at least one of the \( \log |S| \) closest lines to \( q \) are included in the set \( T \). Second, all calls that the algorithm makes to the subroutine \( \text{APNLS} \) in any level of recursion succeeds, i.e., they correctly report an approximate closest line.

**Lemma 3.5.2.** The probability that a given invocation is successful is at least \( 1 - \frac{6}{\log N} \) where \( N = |L| \).

**Proof.** At any level of recursion with the set of lines \( S \), let us rename the set of lines \( \ell_1, \cdots, \ell_n \) (with \( n = |S| \)), based on their distance to the query point \( q \), in non-decreasing order. Then since \( T \) is a random subset of size \( n/2 \) of the set \( S \), the probability that none of the lines \( \ell_1, \cdots, \ell_{\log n} \) is sampled in the set \( T \) is at most \((1/2)^{\log n} = 1/n\). That is, with probability at least \( 1-1/n \), one of the lines \( \ell_1, \cdots, \ell_{\log n} \) is in the set \( T \). By the union bound the probability that this holds in each level of recursion when we run the algorithm on the set \( L \), is at least

\[
1 - \frac{1}{N} - \frac{1}{N^2} - \frac{1}{N^4} - \cdots - \frac{1}{\log^3 N} \geq 1 - \frac{2}{\log^3 N}
\]

Also in each level of recursion, the algorithm makes at most \( 2 \log N \) calls to the \( \text{APNLS} \) subroutine. By Theorem 3.6.10, the probability of failure of each call to \( \text{APNLS} \) is at most \( \frac{2}{\log^4 N} \). Since there are at most \( \log N \) recursive levels, the total number of times we call \( \text{APNLS} \) is at most \( 2 \log^2 N \), and thus the probability that all calls to \( \text{APNLS} \) are successful is at least \( 1 - \frac{4 \log^2 N}{\log^3 N} \). So the probability that the invocation is
Lemma 3.5.3. Suppose that an invocation of NLS\((L, q)\) is successful. Let \(S\) be the set of lines at some level of the recursion and let \(\ell^* = \ell^*_S\) be the closest line in \(S\) to the query point. Then, at the end of each iteration (improvement step), either \(\ell\) is an approximate nearest line \((\text{dist}(q, \ell) \leq \text{dist}(q, \ell^*)(1 + c\varepsilon))\), or its distance to the query \(q\) has been decreased by a factor of \(4\varepsilon\).

Proof. First observe that if \(\ell\) is a \((1 + c\varepsilon)\)-approximate nearest line at the beginning of the iteration, it will remain a \((1 + c\varepsilon)\)-approximate nearest line in all the following iterations as well. This holds since we only perform an update if the new line improves over \(\ell\). Suppose that \(\ell\) is not yet the approximate closest line. Then one of two cases can occur. The first case is that \(\ell^*\) is strictly \(\varepsilon\)-close to the line \(\ell\). Since we assume the invocation of the algorithm is successful, the call to APNLS in line 7 correctly finds \(\ell_{\text{close}}\), which is an approximate closest line (so we should set \(c \geq c_{\text{aplnn}}\), where \(c_{\text{aplnn}}\) is defined by Theorem 3.6.10). Thus \(\ell\) will be updated with an approximate solution at the end of the iteration and the lemma is proved.

Now suppose that \(\ell^*\) is \(\varepsilon\)-far from \(q\). We know that \(\text{dist}(q, \ell^*) < \text{dist}(q, \ell) = x\) and therefore by triangle inequality, \(\text{dist}(\ell, \ell^*) \leq 2x\). Thus, by running the binary search in line 10 of the algorithm, we make sure that \(\ell^* \in S_i\). The following claim proves that the line \(\ell^*\) is included in the set \(A\) that the algorithm finds in line 11.

Claim 3.5.4. If \(\ell^*\) is \(\varepsilon\)-far from the line \(\ell\), then \(\text{dist}(\text{Cl}_{\ell \rightarrow \ell^*}, q_\ell) \leq 3x/\varepsilon\) for sufficiently small value of \(\varepsilon\).

Proof. Let \(q_\ell\) be the projection of \(q\) onto \(\ell\) and let \(b\) be the projection of \(q\) onto \(\ell^*\). Let \(b_\ell\) denote the projection of \(b\) onto \(\ell\) (see Figure 3-6). It is easy to see that

\[
\text{dist}(b, b_\ell) \leq \text{dist}(b, q) + \text{dist}(q, \ell) \leq 2x
\]  

(3.3)

Let \(a = \text{Cl}_{\ell^* \rightarrow \ell}\) and let \(a_\ell = \text{Cl}_{\ell \rightarrow \ell^*}\) be its projection on \(\ell\), i.e., \(a\) and \(a_\ell\) are the closest pair on the two lines and thus the line \(\overline{aa_\ell}\) is perpendicular to both \(\ell\) and \(\ell^*\). Let \(H\) and \(H'\) be the perpendicular hyperplanes to \(\ell\) which pass through \(q\) and \(b\).
respectively. Also let \( \ell' \) denote the line parallel to \( \ell^* \) which passes through \( a_{\ell} \) and let \( b' \) be its intersection with \( H' \).

![Figure 3-6: Figure of Claim 3.5.4](image)

Now, suppose that \( \text{dist}(q_{\ell}, a_{\ell}) > 3x/\varepsilon \). Then since \( q\ellq' \) and \( b'b_{\ell} \) are perpendicular to \( \ell \), then we have \( \text{dist}(b_{\ell}, q_{\ell}) \leq \text{dist}(b, q) \leq x \). Hence by triangle inequality \( \text{dist}(b_{\ell}, a_{\ell}) > (3x/\varepsilon - x) > 2x/\varepsilon \). Therefore using the fact that the two lines are \( \varepsilon \)-far, we have

\[
\text{dist}(b_{\ell}, b') = \text{dist}(a_{\ell}, b_{\ell}) \tan(\overrightarrow{b_{\ell}a_{\ell}}b') > \frac{2x}{\varepsilon} \cdot \frac{\varepsilon}{\sqrt{1 - \varepsilon^2}} > 2x
\]  \hspace{1cm} (3.4)

Now note that since \( \overrightarrow{aa_{\ell}} \) is perpendicular to \( \ell \), therefore we have that \( \overrightarrow{bb'} \) is parallel to \( \overrightarrow{aa_{\ell}} \) and that \( \text{dist}(a, a_{\ell}) = \text{dist}(b, b') \). Also since \( \overrightarrow{aa_{\ell}} \) is perpendicular to \( \ell^* \), it is also perpendicular to \( \ell' \) and therefore, \( \overrightarrow{bb'} \) is perpendicular to \( \ell' \). Moreover, since \( \overrightarrow{bb'} \in H' \), then \( \overrightarrow{bb'} \) is also perpendicular to the projection of line \( \ell' \) onto \( H' \). Thus \( \overrightarrow{bb'} \) is perpendicular to \( b'b_{\ell} \). Therefore, \( \text{dist}(b, b_{\ell}) \geq \text{dist}(b_{\ell}, b') > 2x \). However this contradicts Equation 3.3 and thus the claim holds. \( \square \)

Thus the optimal line \( \ell^* \) is contained in the set \( A \) which is retrieved by the algorithm in line 11 and therefore \( A \) is not empty. Then the following claim completes the proof of the lemma.

**Claim 3.5.5.** If \( \ell^* \) is contained in the set \( A \) and is \( \varepsilon \)-far from \( \ell \), then \( \ell_{f_{\varepsilon}} \) is either a \((1 + c\varepsilon)\)-approximate closest line or its distance to query is at most \( 4\varepsilon x \).

**Proof.** If \( \ell^* \) is in the set \( A \) and is \( \varepsilon \)-far from \( \ell \) then it is also contained in the set \( A^\perp \). Then one of the following two cases can occur.
**Case 1:** \( q \) is outside of the ball \( B(o_A, Y_A/\varepsilon^2) \). In this case, by Lemma 3.3.1, it suffices to consider two scenarios. If \( \text{dist}(q, \ell) \leq \text{dist}(q, \ell^*)(1 + c_1 \varepsilon) \) (where \( c_1 \) is determined by Lemma 3.3.1), then \( \ell^* \) and thus \( \ell_{\text{far}} \) will be a \((1 + c_1 \varepsilon)\)-approximate nearest line. So we should set \( c \geq c_1 \). Second if \( \ell^* \) is strictly \( \varepsilon \)-close to \( \ell \), then in this case since we have a successful invocation, APNLS finds an approximate nearest line and thus \( \ell_{\text{far}} \) is a \((1 + c \varepsilon)\)-approximate closest line. For this we need to set \( c \geq c_{\text{aplnn}} \) where \( c_{\text{aplnn}} \) is determined by Theorem 3.6.10. Note that APNLS reports the \((1 + c_{\text{aplnn}} \varepsilon)\)-approximate closest line.

**Case 2:** \( q \) is inside the ball \( B(o_A, Y_A/\varepsilon^2) \). Since \( o_A \in \ell \), we get that \( x = \text{dist}(q, \ell) \leq \text{dist}(q, o_A) < Y_A/\varepsilon^2 \). Also, by triangle inequality, one can show that the distance of \( q \) to any line \( \ell' \in \mathbb{A}^- \) is at most

\[
\text{dist}(q, \ell') \leq \text{dist}(q, \text{Cl}_{\ell \rightarrow \ell'}) + \text{dist}(\text{Cl}_{\ell \rightarrow \ell'}, \ell') \leq 3x/\varepsilon + \text{dist}(\ell, \ell') \leq 3x/\varepsilon + 2x \leq 4x/\varepsilon
\]

where we have used the fact that \( \ell' \in S_i \) and thus \( \text{dist}(\ell, \ell') \leq 2x \) (by the line 10 of the Algorithm 8). Therefore, since \( B(q, 4x/\varepsilon) \) touches all lines in \( \mathbb{A}^- \), then we have that \( Y_A \leq 4x/\varepsilon \). Thus, the separation parameter of the net module is \( Y_A \varepsilon^3 \leq 4x \varepsilon^2 \). Then by Lemma 3.3.10 one of the following two cases can occur. First if \( \text{dist}(q, \ell_{\text{far}}) \leq \text{dist}(q, \ell^*)(1 + c_2 \varepsilon) \) (where \( c_2 \) is determined by Lemma 3.3.10), then \( \ell_{\text{far}} \) is updated with an approximate solution (so we should set \( c \geq c_2 \)). On the other hand, if \( \text{dist}(q, \ell_{\text{far}}) < 4x \varepsilon \), then we are done. \( \square \)

Therefore, we update \( \ell \) with a line which is either a \((1 + c \varepsilon)\)-approximate closes line or its distance to query is at most \( 4x \varepsilon \). \( \square \)

**Theorem 3.5.6.** Let \( L \) be a set of \( N \) lines in Euclidean space \( \mathbb{R}^d \). Then for any sufficiently small fixed \( \varepsilon > 0 \), there exists a data structure using \( O(N + d)^{O(1/\varepsilon^2)} \) space, and an algorithm that given a query point \( q \), reports a \((1 + c \varepsilon)\)-approximate nearest line with probability at least \( 1 - \frac{6}{\log N} \) in time \( O(1/\varepsilon^2) \).

**Proof.** The proof follows from Lemmas 3.4.3, 3.4.1, 3.5.2 and 3.5.3 and the following argument.
Suppose that we have a successful invocation, which by Lemma 3.5.2 occurs with probability \(1 - \frac{6}{\log N}\). We prove the theorem by induction. At each level of recursion with the set of lines \(S\), one of the \(\log |S|\) closest lines in \(S\) is sampled in the set \(T\). Denote this line by \(\ell_{\log n}\) and let the solution returned by the next level of recursion \((T)\) be \(\ell_T\). By induction we know that \(\ell_T\) is a \((1 + c\varepsilon)\)-approximate closest line in the set \(T\), and thus \(\text{dist}(q,\ell_T) \leq \text{dist}(q,\ell_{\log n})(1 + c\varepsilon)\).

Let \(c = \max\{c_1, 8\}\) where \(c_1\) is determined by Lemma 3.5.3 and let \(\varepsilon < 1/c \leq 1/8\). Also let \(\ell\) be as defined in Algorithm 8. Then by the end of the first improvement step, by Lemma 3.5.3 we have that either \(\ell\) is a \((1 + c_1\varepsilon)\)-approximate (and thus \((1 + c\varepsilon)\)-approximate) closest line in the set \(S\) (since \(c_1 \leq c\)), or

\[
\text{dist}(q,\ell) \leq \text{dist}(q,\ell_T)4\varepsilon \leq \text{dist}(q,\ell_{\log n})(1 + c\varepsilon)(4\varepsilon) < \text{dist}(q,\ell_{\log n})
\]

So by the end of the first iteration, we have improved not only over the line \(\ell_T\) but also on the line \(\ell_{\log n}\). In the following \(\log |S| - 1\) iterations, by Lemma 3.5.3, if a \((1 + c_1\varepsilon)\)-approximate line is not found, we have at least found a line with a closer distance to the query. Since there are at most \(\log n\) such lines, by the end of the algorithm we have found a \((1 + c\varepsilon)\)-approximate nearest line to the query.

\[
\square
\]

3.6 Almost Parallel Nearest Line Search (APNLS)

This section presents a solution to NLS in the special case where all lines in the input set are almost parallel. The algorithm is randomized. It starts by splitting the input set of lines into two halves randomly, then its solves the problem recursively on one of the halves, and uses the solution as the starting line in an iterative improvement process. Suppose we want to improve over a line \(\ell\) with distance \(x\) to the query point \(q\). To this end, we identify all lines with distance at most \(x\) to the query point in the following way. We partition the space into "slabs" using a set of hyperplanes which are perpendicular to \(\ell\). For each slab there is a unique ordering of the set of lines
by their projective distance from $\ell$. This enables the query procedure to identify all lines within a distance $O(x)$ to $\ell$. Specifically, we sort the lines in each slab based on their projective distance to $\ell$ and for each prefix of those lines we build a separate data structure.

Let $B(o, r)$ be a ball that intersects all lines. Suppose that all lines are $\delta$-close to $\ell$. Then (by Parallel module Lemma 3.3.12) the required density of parallel modules is proportional to $\delta$. On the other hand, if all lines are $\delta$-far from $\ell$, then by Lemma 3.3.1 we can find an approximate closest line using the unbounded module for queries that are farther than $r/\varepsilon \delta$.

Let $\varepsilon = \delta_0 > \delta_1 > \delta_2 > \cdots > \delta_t$ be all the sin values of the pairwise angles between the lines. Create a set of concentric balls $B_0, B_1, B_2, \cdots, B_t$, with radii $r/\varepsilon \delta_0, r/\varepsilon \delta_1, \cdots, r/\varepsilon \delta_t$ centered at $o$. Then if the query point falls between $B_j$ and $B_{j+1}$, using unbounded module lemma, we can distinguish between the lines that are $\delta_j$-far, returning a line $\ell'$. We are left with a set of lines that are strictly $\delta_j$-close to $\ell'$ and therefore $\delta_{j+1}$-close to $\ell'$. It now suffices to cover the inner part of $B_{j+1}$ with parallel modules with density proportional to $\delta_{j+1}$ as opposed to $\delta_j$. This enables us to bound the space. After finding $\ell'$ the algorithm uses the parallel module to find the approximate nearest line.

### 3.6.1 Data Structure

The construction procedure for the APNLS data structure is shown in Algorithm 9. As stated earlier, the data structure randomly partitions the set of input lines $S$ (with $n = |S|$) into two halves and builds the data structure recursively for one of the halves. Then for each line $\ell$ in the set $S$, we build a data structure that supports an iterative improvement procedure.

**Definition 3.6.1.** Let $H(\ell, p)$ denote the hyperplane perpendicular to a line $\ell$ which passes through a point $p$. Furthermore, for any sets of points $a$ and $b$, let $\text{dist}_H(a, b) = \text{dist}(H \cap a, H \cap b)$ be the distance between the two sets on the hyperplane $H$ (note that the intersection should not be empty). It is clear that $\text{dist}_H(a, b) \geq \text{dist}(a, b)$.

To construct the data structure we partition the line $\ell$ into the smallest number of
segments \([o_0, o_1], [o_1, o_2], \ldots, [o_{k-1}, o_k]\) with \(o_0 = -\infty\) and \(o_k = \infty\), such that each segment has the strong monotonicity property defined as follows.

**Definition 3.6.2.** We say that a segment \([o_i, o_{i+1}]\) has strong monotonicity property for the set of lines \(S\) if the following two properties hold. Suppose that we continuously move a point \(p \in \ell\) from \(o_i\) to \(o_{i+1}\). The distance between any pair of lines on the hyperplane \(H(\ell, p)\) should be monotone. Furthermore, the ordering of the distances from the lines in \(S\) to \(\ell\) on the hyperplane \(H(\ell, p)\) should remain constant. That is, for any \(\ell_1, \ell_2 \in S\), either it is always the case that \(\text{dist}_{H(\ell, p)}(\ell, \ell_1) \geq \text{dist}_{H(\ell, p)}(\ell, \ell_2)\), or it is always the case that \(\text{dist}_{H(\ell, p)}(\ell, \ell_1) \leq \text{dist}_{H(\ell, p)}(\ell, \ell_2)\).

**Lemma 3.6.3.** There exists a partitioning with at most \(O(n^2)\) segments.

*Proof.* For simplicity suppose that \(\ell\) is identified with the \(X\)-axis. Given two lines \(\ell_1\) and \(\ell_2\) which are strictly \(O(\varepsilon)\)-close to the \(X\)-axis, let \(p_1(x)\) and \(p_2(x)\) be the points on \(\ell_1\) and \(\ell_2\) with their \(X\)-coordinate equal to \(x\). Clearly the other coordinates of \(p_1(x)\) and \(p_2(x)\) can be written as a linear function of \(x\), and therefore the distance between the two lines is of the form \(D(x) = \sqrt{ax^2 + bx + c}\), with the derivative \(D'(x) = \frac{ax + b}{2\sqrt{ax^2 + bx + c}}\). Therefore it is enough to partition \(\ell\) according to the roots of \((ax + b)\) and \((ax^2 + bx + c)\). This contributes at most three segment splits per each pair of lines.

As for for the second property, it is enough to find the roots of the equation \(D_{\ell, \ell_1}(x) - D_{\ell, \ell_2}(x) = 0\), which is of the form \(\sqrt{ax^2 + bx + c} - \sqrt{a'x^2 + b'x + c'} = 0\). Clearly the equation has at most four roots. Therefore the total number of required segments is at most \(7\left(\binom{n}{2}\right) \leq 4n^2\).

It suffices now to build a data structure for each segment \([o_m, o_{m+1}]\). We sort the lines based on their distance to \(\ell\) on the hyperplane \(H(\ell, p)\) for an arbitrary \(p \in [o_m, o_{m+1}]\). Note that this ordering is well-defined and does not depend on the choice of \(p\) because of the strong monotonicity property (so we just let \(p = o_m\)). Then for each prefix \(S_i\) of lines with respect to this order, we proceed as follows. First, since the value of \(\text{dist}_{H(\ell, p)}(\ell, \ell_i)\) is either increasing or decreasing as a function of \(p \in [o_m, o_{m+1}]\), its value achieves minimum at either \(o_m\) or \(o_{m+1}\). Denote the minimizer endpoint by \(o\).
Algorithm 9: APNLS Data Structure

**Input** a set of lines $S$ that are strictly $(2\varepsilon)$-close to each other

1: $T \leftarrow$ half of the lines in $S$ chosen randomly.
2: Recursively build the APNLS data structure for $T$.
3: for $\ell \in S$ do
   4: Partition $\ell$ into segments $[o_0 = -\infty, o_1], [o_1, o_2], \cdots, [o_{k-1}, o_k = \infty]$, s.t.
each segment has strong monotonicity property.
5: for $0 \leq m \leq k - 1$ do
   6: sort all lines $\ell' \in S$ by $\text{dist}_{H(\ell,o_m)}(\ell, \ell')$ in a non-decreasing order
7: for $i = 1$ to $|S|$ do
   8: $S_i \leftarrow$ the first $i$ lines in $S$
   9: $o \leftarrow \arg\min_{o \in \{o_m, o_{m+1}\}} \text{dist}_{H(\ell,o)}(\ell, \ell_i)$
10: $r \leftarrow \text{dist}_{H(\ell,o)}(\ell, \ell_i)$
11: let $\delta_0 > \delta_1 > \cdots > \delta_t > \delta_{t+1} = \{\delta < \varepsilon \mid \exists \ell', \ell'' \in S_i, \delta = \sin\angle(\ell', \ell'')\} \cup \{0, \varepsilon\}$ be the sin values (between 0 and $\varepsilon$) of the
    pairwise angles.
12: $\forall 0 \leq j \leq t : R_j \leftarrow \frac{r}{\varepsilon \delta_j}$
13: $NM_0 \leftarrow$ Build the net module for $S_i$, the ball $B(o, R_0)$, and separation $r\varepsilon^2$
14: for $1 \leq j \leq t + 1$ do
15: $UM_{j-1} \leftarrow$ Build the unbounded module for lines $S_i$ and ball $B(o, r)$,
   and parameter $\delta_{j-1}$
16: for $\ell' \in S_i$ do
17: if $j = t + 1$ then
18: Let $G'_{\ell', j}$ be the two hyperplanes perpendicular to $\ell'$ at positions
   $\ell' \cap \mathcal{S}(o, R_{j-1})$.
19: else
20: Let $G_{\ell', j}$ be a set of hyperplanes perpendicular to $\ell'$ with separation $R_{j+1}\varepsilon^3$, on the segment $\ell' \cap B(o, 2R_j)$,
21: end if
22: for each hyperplane $g \in G_{\ell', j}$ do
23: $PM_{\ell', g} \leftarrow$ Parallel Module for the set of lines $(S_i)_{\ell', \delta_{j-1}}$, base
   line $\ell'$, and hyperplane $g$
24: end for
25: end for
26: end for
27: end for
28: end for
29: end for
Second, for all other lines $\ell' \in S_i$, we have $\text{dist}_{H(\ell,o)}(\ell, \ell') \leq \text{dist}_{H(\ell,o)}(\ell, \ell_i)$. Therefore, the ball $B(o, r)$ intersects all lines in the prefix set $S_i$, where $r = \text{dist}_{H(\ell,o)}(\ell, \ell_i)$.

We describe now how to create the modules. Let $\delta_0 > \delta_1 > \cdots > \delta_t > \delta_{t+1}$ be the set of the sin values of all pairwise angles between the lines in $S_i$. Also we add the extra condition that $\delta_0 = \varepsilon$ and $\delta_{t+1} = 0$, and we ignore those values that are not between $0$ and $\varepsilon$. Note that the total number of such angles is at most $t \leq n^2$. We then define a set of radii, corresponding to the angles as follows. Let $R_j = r/(\varepsilon \delta_j)$ for all $j \leq t$ and consider the set of concentric balls $B(o, R_j)$. First, we build a net module for the ball $B(o, R_0)$ for the set of lines $S_i$ and with separation parameter $R\varepsilon^2$. Then for each $j \leq t$, we create an unbounded module $UM_j$ for the ball $B(o, r)$, the set of lines $S_i$ and parameter $\delta_j$.

Consider any line $\ell' \in S_i$ and each $1 \leq j \leq t$. We create a set of Parallel modules such that given a query inside the ball $B(o, R_j)$, it can distinguish between the set of lines that are $\delta_j$-close to $\ell'$. Note that since there are no other angles between $\delta_j$ and $\delta_{j-1}$, it means that this set of lines is equal to the set of lines that are strictly $\delta_{j-1}$-close to $\ell'$, i.e., $(S_i)_{\ell', \delta_{j-1}}$. We take a set of regularly spaced hyperplanes perpendicular to $\ell'$, separated by $R_j \varepsilon^3$, that cover the part of $\ell'$ that lies inside the slightly larger ball $B(o, 2R_j)$. Note, that for $j = t + 1$ this means that, all lines in $(S_i)_{\ell', \delta_{j-1}}$ are 0-close, i.e., parallel, to $\ell'$. Therefore it is enough to have only one hyperplane perpendicular to $\ell'$ in order to find the approximate nearest line among them. For each such hyperplane $g$, we build a Parallel module on the set of lines $(S_i)_{\ell', \delta_{j-1}}$, base line $\ell'$ and the hyperplane $g$.

**Lemma 3.6.4.** The data structure for a set of lines $L' \subset L$ uses $(|L'| + d)^{O(1/\varepsilon^2)}$ space.

**Proof.** Consider the data structure for $L' \subset L$. At each step of the recursion with $n$ lines, there are a total of $n$ candidates for $\ell$ and for each of them at most $O(n^2)$ segments and for each of the segments, there are $n$ different $S_i$. For each $S_i$ there are at most $O(n^2)$ different values of $\delta_j$. For each of them, the unbounded module uses space $S_{\text{ANN}}(2n, \varepsilon)$ (by Remark 3.3.9). Also for each such $\delta_j$ and each line $\ell'$, the
number of hyperplanes \(|G_{\ell,j}|\) is at most \(\frac{4R_j}{R_j \varepsilon^3} = O(1/\varepsilon^3)\). For each such hyperplane, we build a parallel module with at most \(n\) lines. Therefore the total space of each step of recursion (using Remark 3.3.13) will be \(O(n^7/\varepsilon^3 \times S_{\text{ANN}}(n, \varepsilon))\). The recursion formula for the space at each level of recursion is

\[S(n) \leq S(n/2) + 4n^7/\varepsilon^3 \times S_{\text{ANN}}(n, \varepsilon)\]

and therefore, the total space used by the algorithm is \(O(|L'|^7/\varepsilon^3 \times S_{\text{ANN}}(|L'|, \varepsilon))\).

Note that for storing the lines we can use \(O(|L'|d)\) space or just use the \(|L'|\) pointers to the lines stored globally. Now since \(S_{\text{ANN}}(|L'|, \varepsilon)\) is of the form \(O(|L'| + d)^{O(1/\varepsilon^2)}\), the space of this data structure is of this form as well.

**Lemma 3.6.5.** The data structure for a set of lines \(L' \subset L\) can be constructed in time \((|L'| + d)^{O(1/\varepsilon^2)}\).

**Proof.** The same as the previous lemma, one can show that it takes \(O(|L'|^7/\varepsilon^3 \times \mathcal{C}T_{\text{ANN}}(|L'|, \varepsilon))\). Thus the total construction time is of the form \((|L'| + d)^{O(1/\varepsilon^2)}\).

### 3.6.2 Query Procedure

The query procedure of APNLS is sketched in Algorithm 10. The algorithm proceeds as follows. Given the query point \(q\), it first checks whether the problem can be solved by brute force. That is, if the total number of points in the input set \(S\) is at most \(\log^3 N\), where \(N = |L|\) is the total number of points in the database. Then it computes the distance of the query to all lines in \(S\), finds the closest line and reports it.

Otherwise, it first solves the problem on the half of lines \(T\), for which a data structure has been constructed. Let \(\ell\) be the approximate nearest line in the sampled set \(T\). The algorithm starts with \(\ell\) and keeps improving it so that the distance from the query to \(\ell\) in each of the \(\log |S|\) iterations decreases until an approximate closest line is found. Since the probability that one of the \(\log |S|\) closest lines to \(q\) have been sampled in the set \(T\) is at least \(1 - 1/|S|\), \(\log |S|\) iterations are sufficient to find the approximate nearest line.
The improvement step is as follows. Let $x$ be the distance from $q$ to the line $\ell$ and let $q_\ell$ be the projection of $q$ onto the line $\ell$. Then using binary search we find the segment $[o_m, o_{m+1}]$ that $q_\ell$ falls into and proceed to the corresponding data structure. As the optimum line should have distance less than $x$ to the query point, its distance to $\ell$ is at most $2x$ and its distance to $\ell$ on the hyperplane $H(\ell, q_\ell)$ should be at most $3x$. Note that, because of the strong monotonicity property, the ordering of the lines inside the interval $[o_m, o_{m+1}]$ is constant. Therefore we can use binary search to retrieve all lines $S_i$ that have distance at most $3x$ from $\ell$ on the hyperplane $H(\ell, q_\ell)$. Let $o$ be the endpoint of the segment (either $o_m$ or $o_{m+1}$) for which the distance between $\ell_i$ and $\ell$ is minimized on the hyperplane $H(\ell, o)$, and denote this distance by $r = \text{dist}_{H(\ell, o)}(\ell, \ell_i)$.

Now the following two cases can occur. If $q$ is inside the smallest ball $B(o, R_0)$, then we use the net module $NM_0$ and update the line. Otherwise we find the largest $j \leq t$ such that $q$ is outside of the ball $B(o, R_j)$. We use the unbounded module $UM_j$ to get the line $\ell_{um}$. Then we know that either $\ell_{um}$ is an approximate nearest line or the nearest line is strictly $\delta_j$-close to $\ell_{um}$, or equivalently $\delta_{j+1}$-close to it. Therefore we get the closest hyperplane $g \in G_{\ell_{um}, j+1}$, and use the corresponding Parallel module to find the line $\ell_{pm}$. We update $\ell$ with this line $\ell_{pm}$ (if it provides an improvement). Finally, after all iterations, we report the line $\ell$ as the approximate nearest line. Figure 3-7 shows an example of the data structure and the query point.

**Lemma 3.6.6.** The query processing algorithm on the set $L' \subset L$ runs in time $(\log |L'| + d + 1/\varepsilon)^{O(1)}$.

**Proof.** First note that each computation takes $O(d)$ time. At each step of the recursion with $n$ lines, if we have $n \leq \log^3 N$, the the running time is $O(d \log^3 N)$. Otherwise, the longest path of the execution of the algorithm consists of $\log n$ iterations and for each of them we run the net module which by Remark 3.3.11 takes $\frac{4R_{\text{run}}}{r \varepsilon^2} = O(n/\varepsilon^4)$. Therefore, the total time taken is $\log n \times T_{\text{ANN}}(4n/\varepsilon^4, \varepsilon)$. The recursion formula for the running time at each level of recursion is

$$T(n) \leq T(n/2) + \log n \times T_{\text{ANN}}(4n/\varepsilon^4, \varepsilon), \quad T(\log^3 N) = \log^3 N$$
Algorithm 10: APNLS Query Processing

Input a set of lines $S$ that are strictly $2\varepsilon$-close to each other, and the query point $q$

Output the approximate nearest line $\ell \in S$ to the query point $q$

1: if $|S| \leq \log^3 N$ then
2: return the nearest line found by the brute force.
3: end if
4: $T \leftarrow$ the set of lines sampled during the preprocessing stage.
5: $\ell \leftarrow$ the result of executing APNLS$(T, q)$
6: for $(\log |S|)$ times do
7: $x \leftarrow \text{dist}(q, \ell)$
8: $q_{\ell} \leftarrow$ the projection of $q$ onto $\ell$.
9: Use binary search to find the segment $[o_m, o_{m+1}]$ containing $q_{\ell}$.
10: Use binary search to find the largest $i$ such that for all lines $\ell' \in S_i$, we have $\text{dist}_{H(\ell, q)}(\ell', \ell) \leq 3x$
11: $o \leftarrow \text{arg min}_{o \in \{o_m, o_{m+1}\}} \text{dist}_{H(\ell, o)}(\ell, \ell_i)$
12: $r \leftarrow \text{dist}_{H(\ell, o)}(\ell, \ell_i)$
13: if $q \in B(o, R_0)$ then
14: $\ell \leftarrow$ the best of $\ell$ and $\ell_{nm}$, where $\ell_{nm} = NM_0(q)$ is the output of the net module.
15: else
16: $j \leftarrow$ the largest value in the set $\{0, 1, \cdots, t\}$ such that $q \not\in B(o, R_j)$
17: $\ell_{um} \leftarrow$ the output of $UM_j(q)$
18: $g \leftarrow$ the closest hyperplane to $q$ in the set $G_{\ell_{um, j+1}}$
19: $\ell_{pm} \leftarrow$ the result of $PM_{\ell_{um, g}}(q)$
20: update $\ell$ with the best of $\ell$ and $\ell_{pm}$
21: end if
22: end for
23: return $\ell$
and therefore since $T_{\text{ANN}}(n/\varepsilon^4, \varepsilon)$ is poly-logarithmic in $n$ and polynomial in $1/\varepsilon$, the total running time of the algorithm is $O(d \log^3 N + \log^2 |L'| \times T_{\text{ANN}}(4|L'|/\varepsilon^4, \varepsilon)) = O(d \log^3 N + \log^2 N \times T_{\text{ANN}}(4N/\varepsilon^4, \varepsilon))$ which is poly-logarithmic in $N$ and polynomial in $1/\varepsilon$ and $d$.

3.6.3 Analysis of APNLS Algorithm

Definition 3.6.7. An invocation $\text{APNLS}(L', q)$ (for $L' \subset L$) is called successful if in all recursive calls involving a set of lines $S$, at least one of the $\log |S|$ closest lines to $q$ is included in the set $T$.

Lemma 3.6.8. The probability that a given invocation on set $L' \subset L$ is successful is at least $1 - \frac{2}{\log^2 N}$ where $N = |L| \geq |L'|$.

Proof. Consider any level of recursion with the set of lines $S$. Without the loss of generality we can assume that those lines $\ell_1, \cdots, \ell_n$, $n = |S|$, are order based on their distance to the query point $q$, in a non-decreasing order. Since $T$ is a random subset of the set $S$ of size $n/2$, the probability that none of the lines $\ell_1, \cdots, \ell_{\log n}$ is included in the set $T$ is at most $(1/2)^{\log n} = 1/n$. That is, with probability at least $1 - 1/n$, one of the lines $\ell_1, \cdots, \ell_{\log n}$ is in the set $T$. By the union bound, the probability that this holds in each level of the recursion when we run the algorithm on the set $L'$ is at least

$$1 - \frac{1}{|L'|} - \frac{1}{|L'|/2} - \frac{1}{|L'|/4} - \cdots - \frac{1}{\log^2 N} \geq 1 - \frac{1}{N} - \frac{1}{N/2} - \frac{1}{N/4} - \cdots - \frac{1}{\log^2 N} \geq 1 - \frac{2}{\log^2 N}$$

Lemma 3.6.9. Suppose that the invocation of the $\text{APNLS}(L', q)$ is successful for $L' \subset L$. Let $S$ be the set of lines at some level of the recursion and let $\ell^*_S$ be the closest line in $S$ to the query point. Then by the end of each iteration, either, $\ell$ is an approximate nearest line ($\text{dist}(q, \ell) \leq \text{dist}(q, \ell^*_S)(1 + c\varepsilon)$), or its distance to the query $q$ has been decreased by a factor of $3\varepsilon$. 81
Proof. First observe that if $\ell$ is a $(1 + c\varepsilon)$-approximate nearest line at the beginning of the iteration, it will remain a $(1 + c\varepsilon)$-approximate nearest line in the following iterations as well. This is because we only perform an update if the new line improves over $\ell$. Suppose that $\ell$ is not yet the approximate closest line.

Since $\text{dist}(q, \ell) = x$, we know that $\text{dist}(q, \ell^*) \leq x$ and therefore we have $\text{dist}(q_\ell, \ell^*) \leq 2x$. Hence, since $\ell$ and $\ell^*$ are strictly $2\varepsilon$-close to each other, we have that $\text{dist}_{H(\ell,q)}(\ell, \ell^*) \leq \frac{2x}{\sqrt{1-4\varepsilon^2}} \leq 3x$ for sufficiently small value of $\varepsilon$. Therefore, the set of lines $S_i$ that we retrieve in line 10 contains the optimal line $\ell^*$. Furthermore, the value of $\text{dist}_{H(\ell,p)}(\ell, \ell_i)$ increases as we distance $p$ from $o$ inside the segment $[o_m, o_{m+1}]$.

Specifically for $p = q_\ell$, it means that $r \leq 3x\varepsilon$. Now, if $q \in B(o, R_0)$, then by the net module Lemma 3.3.10 one of the two cases can occur. The first case occurs if $\text{dist}(q, \ell_{nm}) \leq \text{dist}(q, \ell^*)(1 + c_1\varepsilon)$ (where $c_1$ is defined by Lemma 3.3.10), which means that the value $\ell$ is updated with the approximate nearest line and thus the algorithm reports the approximate nearest line (we should just set $c \geq c_1$). The second case occurs if $\text{dist}(q, \ell_{nm}) \leq r\varepsilon^2/\varepsilon \leq 3x\varepsilon$, which means that we have reduced the distance $\text{dist}(q, \ell)$ by a factor of $3\varepsilon$.

Otherwise, we are using the unbounded module of $UM_j$. This means that by Lemma 3.3.1, either $\text{dist}(q, \ell_{um}) \leq \text{dist}(q, \ell^*)(1 + c_2\varepsilon)$ (where $c_2$ is determined by Lemma 3.3.1), or $\ell^* \in (S_i)_{\ell_{um}, \delta_j}$, i.e. the optimal line is strictly $\delta_j$-close to the line $\ell_{um}$. However, in both cases we know that there exists a line $\ell'$ ($\ell_{um}$ in the first case and $\ell^*$ otherwise) that is strictly $\delta_j$-close to $\ell_{um}$ and that $\text{dist}(q, \ell') \leq \text{dist}(q, \ell^*)(1 + c_2\varepsilon)$. Since there is no other pairwise angle $\alpha$ such that $\delta_{j+1} = \sin \alpha < \delta_j$, it means that $\ell'$ is $\delta_{j+1}$-closet to $\ell_{um}$. Now, note that if $j = t$, then the set $(S_i)_{\ell_{um}, \delta_j}$ are all parallel to $\ell_{um}$, and therefore, using any Parallel module perpendicular to $\ell_{um}$ returns an approximate nearest line with approximation factor $(1 + c_2\varepsilon)(1 + \varepsilon) \leq (1 + (c_2 + 2)\varepsilon)$. Thus, we have $c \geq c_2 + 2$.

Otherwise if $j < t$, we use Lemma 3.3.12 to prove the correctness of the algorithm. Taking into account that $q \in B(o, R_j)$ and that the hyperplanes cover the range of the $\ell_{um} \cap B(o, 2R_j)$, we have that $q$ lies inside these hyperplanes and thus its distance to the closest hyperplane is at most $R\varepsilon^2 = \frac{\varepsilon^2}{\delta_j}$ and therefore Lemma 3.3.12.
applies. Using the lemma one can see that either
\[ \text{dist}(q, \ell_{pm}) \leq \text{dist}(q, \ell')(1 + c_3 \varepsilon) \leq \text{dist}(q, \ell^*)(1 + (c_2 + c_3 + 1)\varepsilon) \] (where \( c_3 \) is defined by Lemma 3.3.12), in which case by
the end of the iteration \( \ell \) is an approximate nearest line (we should set \( c \geq c_2 + c_3 + 1 \)),
or \[ \text{dist}(q, \ell_{pm}) \leq r\varepsilon \leq 3x\varepsilon \] which means that we have decreased the value of \( \text{dist}(q, \ell) \)
by a factor of \( 3\varepsilon \).

**Theorem 3.6.10.** Let \( L' \subset L \) be a set of lines in Euclidean space \( \mathbb{R}^d \) that are almost
parallel. Then for any sufficiently small fixed \( \varepsilon > 0 \), there exists a data structure
using \( O(|L'| + d)^{O(1/\varepsilon^2)} \) space, and an algorithm that given a query point \( q \), reports
a \((1 + c\varepsilon)\)-approximate nearest line with probability at least \( 1 - \frac{2}{\log^4 N} \) in time \((d + \log |L'| + 1/\varepsilon)^{O(1)}\).

**Proof.** The proof follows from Lemmas 3.6.6, 3.6.4, 3.6.8 and 3.6.9 and the following
argument which is analogous to the proof of Theorem 3.5.6

Suppose that we have a successful invocation (which by Lemma 3.6.8 happens
with probability \( 1 - \frac{2}{\log^4 N} \)). Then we prove the theorem by induction. At each level
of recursion with the set of lines \( S \), one of the \( \log n \) (where \( n = |S| \)) closest lines in \( S \)
is included in the set \( T \). Let this line be \( \ell_{\log n} \) and let the solution returned by the next
level of recursion \( (T) \) be \( \ell_T \). By induction we know that \( \ell_T \) is a \((1 + c\varepsilon)\)-approximate
closest line in the set \( T \), and thus \( \text{dist}(q, \ell_T) \leq \text{dist}(q, \ell_{\log n})(1 + c\varepsilon) \).

Let \( c = \max\{c_1, 6\} \) where \( c_1 \) is determined by Lemma 3.6.9 and let \( \varepsilon < 1/c \leq 1/6 \).
Also let \( \ell \) be as defined in Algorithm 10. Then by the end of the first improvement
step, by Lemma 3.6.9 we have that either \( \ell \) is a \((1 + c_1\varepsilon)\)-approximate (and thus a
\((1 + c\varepsilon)\)-approximate) nearest line in the set \( S \) or

\[ \text{dist}(q, \ell) \leq \text{dist}(q, \ell_T)3\varepsilon \leq \text{dist}(q, \ell_{\log n})(1 + c\varepsilon)(3\varepsilon) < \text{dist}(q, \ell_{\log n}) \]

So by the end of the first iteration, we have improved not only over the line \( \ell_T \) but
also over the line \( \ell_{\log n} \). In the following \( \log n - 1 \) iterations, by Lemma 3.6.9, if a
\((1 + c_1\varepsilon)\)-approximate line is not found, we have found at least a line with a closer
distance to the query. Since there are at most \( \log n \) such lines, by the end of the
algorithm we have found a \((1 + c\varepsilon)\)-approximate nearest line to the query.
Figure 3-7: Example: There are only three lines with sin values of $2\varepsilon, \delta_1$ and $\delta_2$. $B(o, r)$ touches all lines. The red dots show some of the sampled points in the net module inside $B(o, r/\varepsilon^2)$. The blue dots on the ball $B(o, r/\varepsilon^2)$ show some of the points in the unbounded module $UM_0$ which is used for the query point $q$. Suppose that the output of this module is $\ell_{um}$. Then we take the closest hyperplane $g$ to $q$ which is perpendicular to $\ell_{um}$. The points of the parallel module on $g$ are depicted by green dots. The output of the parallel module is denoted by $\ell_{pm}$. Note that not all the hyperplanes are shown in the picture.
Chapter 4

Sparse Linear Regression: Nearest Induced Flat

4.1 Introduction and Background

The goal of the Sparse Linear Regression (SLR) problem is to find a sparse linear model explaining a given set of observations. Formally, we are given a matrix $M \in \mathbb{R}^{d \times n}$, and a vector $q \in \mathbb{R}^d$, and the goal is to find a vector $\tau$ that is $k$-sparse (has at most $k$ non-zero entries) and that minimizes $\|q - M\tau\|_2$. The problem also has a natural query/online variant where the matrix $M$ is given in advance (so that it can be preprocessed) and the goal is to quickly find $\tau$ given $q$.

Various variants of SLR has been extensively studied, in a wide range of fields including (i) statistics and machine learning [Tib96, Tib11], (ii) compressed sensing [Don06], and (iii) computer vision [WYG+09]. The query/online variant is of particular interest in the application described by Wright et al. [WYG+09], where the matrix $M$ describes a set of image examples with known labels and $q$ is a new image that the algorithm wants to label.

If the matrix $M$ is generated at random or satisfies certain assumptions, it is known that a natural convex relaxation of the problem finds the optimum solution in polynomial time [CRT06, CDS98]. However, in general the problem is known to be NP-HARD [Nat95, DMA97], and even hard to approximate up to a polynomial
factor [FKT15] (see below for a more detailed discussion). Thus, it is highly likely
that any algorithm for this problem that guarantees "low" approximation factor must
run in exponential time. A simple upper bound for the offline problem is obtained
by enumerating \( \binom{n}{k} \) possible supports of \( \tau \) and then solving an instance of \( d \times k \)
least squares problem. This results in \( n^k(d+k)^{O(1)} \) running time, which (to the best
of our knowledge) constitutes the fastest known algorithm for this problem. At the
same time, one can test whether a given set of \( n \) points in a \( d \)-dimensional space is
degenerate by reducing it to \( n \) instances of SLR with sparsity \( d \). The former problem
is conjectured to require \( \Omega(n^d) \) time [ES95]—this is the \textit{affinely degenerate conjecture}.
This provides a natural barrier for any running time improvements (we elaborate on
this below in 4.1.1).

In this thesis, we study the complexity of the problem in the case where the
sparsity parameter \( k \) is constant. In addition to the formulation above, we also
consider two more constrained variants of the problem. First, we consider the \textbf{Affine SLR}
where the vector \( \tau \) is required to satisfy \( \|\tau\|_1 = 1 \), and second, we consider
the \textbf{Convex SLR} where additionally \( \tau \) should be non-negative. We focus on the
approximate version of these problems, where the algorithm is allowed to output a
\( k \)-sparse vector \( \tau' \) such that \( \|M\tau' - q\|_2 \) is within a factor of \( 1 + \varepsilon \) of the optimum.

It is not hard to see that the SLR problem is equivalent to the \textit{Nearest Linear
Induced Flat} (NLIF) problem defined as follows. Given a set \( P \) of \( n \) points in \( d \)
dimensions and a \( d \)-dimensional vector \( q \), the task is to find a \( k \)-dimensional flat
spanning a subset \( B \) of \( k \) points in \( P \) and the origin, such that the (Euclidean) distance
from \( q \) to the flat is minimized. The Affine and Convex variants of SLR respectively
correspond to finding the \textit{Nearest Induced Flat} (NIF) and the \textit{Nearest Induced Simplex}
(NIS) problems, where the goal is to find the closest \( (k-1) \)-dimensional flat/simplex
spanned by a subset of \( k \) points in \( P \) to the query.

**Motivation for the problems studied.** Given a large set of items (e.g., images),
one would like to store them efficiently for various purposes. One option is to pick a
relatively smaller subset of representative items (i.e., support vectors), and represent
all items as a combination of this supporting set. Note, that if our data-set is diverse and is made out of several distinct groups (say, images of the sky, and images of children), then naturally, the data items would use only some of the supporting set for representation (i.e., the representation over the supporting set is naturally sparse). As such, it is natural to ask for a sparse representation of each item over the (sparse but still relatively large) supporting set. (As a side note, surprisingly little is known about how to choose such a supporting set in theory, and the problem seems to be surprisingly hard even for points in the plane.)

Now, when a new item arrives to the system, the task is to compute its best sparse representation using the supporting set, and we would like to do this as fast as possible (which admittedly is not going to be that fast, see below for details).

### 4.1.1 Results

In this thesis, we present upper and lower bounds\(^1\) for the aforementioned problems. Our results fall into the following classes:

\(\text{(A) Data-structures.}\) We present data-structures to solve the online variants of the SLR, Affine SLR and Convex SLR problems, for general value of \(k\). Our algorithms use a provided approximate nearest-neighbor (ANN) data-structure as a black box. The new results are summarized in 4.1. See [HPIM16] for further improvements of this result for the special case of \(k = 2\).

\(^1\)Some of our results involve randomized algorithms. See the statements of theorems for the details.

---

### Table 4.1: Summary of results

<table>
<thead>
<tr>
<th>Comment</th>
<th>Space</th>
<th>Query</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLR</td>
<td>(n^{k-1}S_{\text{ANN}}(n, \varepsilon))</td>
<td>(n^{k-1}T_{\text{ANN}}(n, \varepsilon))</td>
<td>Theorem 4.3.7</td>
</tr>
<tr>
<td>Affine SLR</td>
<td>(n^{k-1}S_{\text{ANN}}(n, \varepsilon))</td>
<td>(n^{k-1}T_{\text{ANN}}(n, \varepsilon))</td>
<td>Theorem 4.3.6</td>
</tr>
<tr>
<td>Convex SLR</td>
<td>(n^{k-1}S_{\text{ANN}}(n, \varepsilon) \log^k n)</td>
<td>(n^{k-1}T_{\text{ANN}}(n, \varepsilon) \log^k n)</td>
<td>Lemma 4.4.15</td>
</tr>
</tbody>
</table>
(B) Conditional lower bound. We complement the data-structures, by a conditional lower bound of $\Omega(n^{k/2}/(e^k \log^{O(1)} n))$, for the offline variant of all three problems. This implies that one can not hope to do much better. Improving this lower bound further, for the case of $k = 4$, would imply a nontrivial lower bound for the famous Hopcroft's problem. See Section 4.5.1 for the description.

Our algorithms offer tradeoffs that notably improve the running time (for the offline version) or the space/query time (for the online/query version) over the aforementioned naive algorithm for small values of $k$. Moreover, our algorithms involve formulating and solving several interesting geometric subproblems, which we believe to be of independent interest.

Detecting affine degeneracy.

Given a point set $P$ in $\mathbb{R}^d$ (here $d$ is conceptually small), it is natural to ask if the points are in general position – that is, all subsets of $d + 1$ points are affinely independent. The affinely degenerate conjecture states that this problem requires $\Omega(n^d)$ time to solve [ES95]. This can be solved by building the arrangement of hyperplanes in the dual, and detecting any vertex that have $d + 1$ hyperplanes passing through it. This problem is also solvable using our data-structure. Indeed, instantiate Theorem 4.3.6), for $k = d - 1$, and using a low-dimensional $(1 + \varepsilon)$-ANN data-structure of Arya et al. [AMN+98]. Such an ANN data-structure uses $S_{\text{ANN}}(n, \varepsilon) = O(n)$ space, $O(n \log n)$ preprocessing time, and $O(\log n + 1/\varepsilon^d) = O(\log n)$ query time (for $\varepsilon$ a fixed constant $< 1$). Detecting affine degeneracy then reduces to solving for reach point of $q \in P$, the problem of finding the closest $(d - 1)$-dimensional induced flat of $P \setminus \{q\}$ to $q$. It is easy to show that this can be solved using our data-structure with an extra log factor\footnote{The details are somewhat tedious – one generates $O(\log n)$ random samples of $P$ where each point is picked with probability half. Now, we build the data-structure for each of the random samples. With high probability, for each of the query point $q \in P$, one of the samples contains, with high probability, the $d$ points defining the closest flat, while not containing $q$.}. Thus, up to polylogarithmic factor, the data-structure of Theorem 4.3.6 is optimal under the affinely degenerate conjecture.
4.1.2 Related work

The computational complexity of the approximate sparse linear regression problem has been studied, e.g., in [Nat95, DMA97, FKT15]. In particular, the last paper proved a strong hardness result, showing that the problem is hard even if the algorithm is allowed to output a solution with sparsity $k' = k2^{2\log^{1-d}\delta/n}$ whose error is within a factor of $n^c m^{1-\alpha}$ from the optimum, for any constants $\delta, \alpha > 0$ and $c > 1$.

The query/online version of the Affine SLR problem can be reduced to the Nearest $k$-flat Search Problem, where the database consists of a set of $k$-flats (affine subspaces) of size $N$ and the goal is to find the closest $k$-flat to a given query point $q$. Let $P$ be a set of $n$ points in $\mathbb{R}^d$ that correspond to the columns of $M$. The reduction proceeds by creating a database of all $\binom{n}{k}$ possible $k$-flats that pass through $k$ points of $P$. For the case of $k = 2$, it is known that there exists an algorithm that uses $(dn/\varepsilon)^{O(1)}$ space and $(1/\varepsilon + d + \log n)^{O(1)}$ query time. This yields an algorithm with space usage $\frac{n^{14}}{\varepsilon^2} S_{\text{ANN}}(n^2, \varepsilon)$ and query time of $\log^2 n T_{\text{ANN}}\left(\frac{n^2}{\varepsilon^2}, \varepsilon\right)$ [Mah15]. Similar results can be achieved for the other variants.

4.1.3 Techniques and Sketch of the Algorithms

Affine SLR (nearest flat). To solve this problem, we first fix a subset $B \subseteq P$ of $k - 1$ points, and search for the closest $(k - 1)$-flat among those that contain $B$. Note, that there are at most $n - k + 1$ such flats. Each such flat $f$, as well as the query flat $Q_{\text{flat}}$ (containing $B$ and the query $q$), has only one additional degree of freedom, which is represented by a vector $v_H$ ($v_Q$, resp.) in a $d - k + 1$ space. The vector $v_H$ that is closest to $v_Q$ corresponds to the flat that is closest to $q$. This can be found approximately using standard ANN data structure, resulting in an algorithm with running time $O(n^{k-1} \cdot T_{\text{ANN}}(n, \varepsilon))$. Similarly, by adding the origin to the set $B$, we could solve the SLR problem in a similar way.

\footnote{The exact exponent is not specified in the main theorem of [Mah15] and it was obtained by an inspection of the proofs in that paper.}
**Convex SLR (nearest simplex).** This case requires an intricate combination of low and high dimensional data structures, and it is the most challenging part of this work. To find the closest $(k - 1)$-dimensional induced simplex, one approach would be to fix $B$ as before, and find the closest corresponding flat. This will not work, however, if the projection of the query point onto the closest flat falls outside of the corresponding simplex. Because of that, we need to restrict our search to the flats of feasible simplices, i.e., the simplices $S$ such that the projection of the query point onto the corresponding flat falls inside $S$. If we manage to find this set, we can use the algorithm for affine SLR to find the closest one. Note that finding the distance of the query to the closest non-feasible simplex can easily be computed in time $n^{k-1}$ as the closest point lies on a lower dimensional object.

Let $S$ be a simplex which is obtained from $B$ and an additional point $x$. Then we can determine whether $S$ is feasible or not only by looking at (i) the relative positioning of $x$ with respect to $B$, that is, how the simplex looks like in the flat going through $S$, (ii) the relative positioning of $q$ with respect to $B$, and (iii) the distance between the query and the flat of the simplex. Thus, if we were given a set of simplices such that all their flats were at a distance $r$ from the query, we could build a single data structure for retrieving all the feasible flats. This could be done by mapping all of them in advance onto a unified $(k - 1)$ dimensional space (the “parameterized space”), and then using $k-1$ dimensional orthogonal range-searching trees in that space.

However, this distance $r$ is not known in general. Fortunately, we show that the feasibility property is monotone in the distance: the farther the flat of the simplex is from the query point, the weaker constraints it needs to satisfy. Thus, given a threshold value $r$, our algorithm retrieves the simplices satisfying the restrictions they need to satisfy if they were at a distance $r$ from the query. This allows us to use binary search for finding the right value of $r$ by random sampling. The final challenge is that, since our access is to an approximate NN data structure (and not an exact one), the above procedure yields a superset of feasible simplices. The algorithm then finds the closest flat corresponding to the simplices in this superset. We show that
although the reported simplex may not be feasible (the projection of \( q \) on to the corresponding flat does not fall inside the simplex), its distance to the query is still approximately at most \( r \).

**Lower bound.** Our reduction from \( k \)-sum is randomized, and based on the following idea. First observe that by testing whether the solution to SLR is zero or non-zero we can test whether there is a subset of \( k \) numbers and a set of associated \( k \) weights such that the *weighted* sum is equal to zero. In order to solve the \( k \)-sum, however, we need to force the weights to be equal to 1. To this end, we lift the numbers to vectors, by extending each number by a vector selected at random from the standard basis \( e_1 \ldots e_k \). We then ensure that in the selected set of \( k \) numbers, each element from the basis is represented exactly once and with weight 1. This ensures that the solution to SLR yields a feasible solution to \( k \)-sum.

### 4.2 Preliminaries

#### 4.2.1 Notations

Throughout this chapter, we assume \( P \subseteq \mathbb{R}^d \) is the set of input points which is of size \( n \). For simplicity, we assume that the point-sets are non-degenerate, however this assumption is not necessary for the algorithms. We use the notation \( X \subset_i B \) to denote that \( X \) is a subset of \( B \) of size \( i \), and use 0 to denote the origin. Furthermore, for two points \( a,b \in \mathbb{R}^d \), we denote the segment formed by the two points using \( ab \) and the line formed by them using \( \overline{ab} \).

**Definition 4.2.1.** For a set of points \( S \), let \( f_S = \text{aff}(S) = \left\{ \sum_{i=1}^{k} \alpha_i x_i \bigg| k > 0, x_i \in S, \text{ and } \sum_{i=1}^{k} \alpha_i = 1 \right\} \) be the \((|S| - 1)\)-dimensional flat (or \((|S| - 1)\)-flat for short) passing through the points in the set \( S \) (aka the affine hull of \( S \)). The \((|S| - 1)\)-dimensional simplex \((k-1)\)-simplex for short) that is formed by the convex-hull of the points of \( S \) is denoted by \( \triangle_S \). We denote the *interior* of a simplex \( \triangle_S \) by \( \text{int}(\triangle_S) \).
Definition 4.2.2 (distance and nearest-neighbor). For a point \( q \in \mathbb{R}^d \), and a point \( x \in \mathbb{R}^d \), we use \( d(q, x) = \| q - x \|_2 \) to denote the distance between \( q \) and \( x \). For a closed set \( X \subseteq \mathbb{R}^d \), we denote by \( d(q, X) = \min_{x \in X} \| q - x \|_2 \) the distance between \( q \) and \( X \). The point of \( X \) realizing the distance between \( q \) and \( X \) is the nearest neighbor to \( q \) in \( X \), denoted by \( \text{nn}(q, X) \). We sometimes refer to \( \text{nn}(q, X) \) as the projection of \( q \) onto \( X \).

More generally, given a finite family of such sets \( \mathcal{G} = \{ X_i \subseteq \mathbb{R}^d \mid i = 1, \ldots, m \} \), the distance of \( q \) from \( \mathcal{G} \) is \( d(q, \mathcal{G}) = \min_{X \in \mathcal{G}} d(q, X) \). The nearest-neighbor \( \text{nn}(q, \mathcal{G}) \) is defined analogously to the above.

Assumption 4.2.3. Throughout the chapter, we assume we have access to a data structure that can answer \((1 + \varepsilon)\)-ANN queries on a set of \( n \) points in \( \mathbb{R}^d \). We use \( S_{\text{ANN}}(n, \varepsilon) \) to denote the space requirement of this data structure, and by \( T_{\text{ANN}}(n, \varepsilon) \) to denote the query time.

Induced stars, bouquets, books, simplices and flats.

Definition 4.2.4. Given a point \( b \) and a set \( P \) of points in \( \mathbb{R}^d \), the \textit{star} of \( P \), with the base \( b \), is the set of segments \( \text{star}(b, P) = \{ bx \mid x \in P \setminus \{b\} \} \). Similarly, given a set \( B \) of points in \( \mathbb{R}^d \), with \( |B| = k - 1 \leq d \), the \textit{book} of \( P \), with the base \( B \), is the set of simplices \( \Delta(B, P) = \{ \Delta_{B \cup \{x\}} \mid x \in P \setminus B \} \). Finally, the set of flats induced by these simplices, is the \textit{bouquet} of \( P \), denoted by \( \text{bqt}(B, P) = \{ f_{B \cup \{x\}} \mid x \in P \setminus B \} \).

If \( B \) is a single point, then a book is a star, and a bouquet is a set of lines all passing through a point.

Definition 4.2.5. For a set \( P \subseteq \mathbb{R}^d \), let \( L_k(P) = \{ f_{S \cup \emptyset} \mid S \subseteq_k P \} \) be the set of all linear \( k \)-flats induced by \( P \), and \( \mathcal{F}_k(P) = \{ f_S \mid S \subseteq_k P \} \) be the set of all \((k - 1)\)-flats induced by \( P \). Similarly, let \( \Delta_k(P) = \{ \Delta_S \mid S \subseteq_k P \} \) be the set of all \((k - 1)\)-simplices induced by \( P \).
4.2.2 Problems

In the following, we are given a set $P$ of $n$ points in $\mathbb{R}^d$, a query point $q$ and parameters $k$ and $\varepsilon > 0$. We are interested in the following problems:

I. **NLIF** (Nearest Linear Induced Flat): Compute $\text{nn}(q, \mathcal{L}_k(P))$.

II. **ANLIF** (Approximate Nearest Linear Induced Flat): Compute a $k$-flat $f \in \mathcal{L}_k(P)$, such that $d(q, f) \leq (1 + \varepsilon)d(q, \mathcal{L}_k(P))$.

III. **NIF** (Nearest Induced Flat): Compute $\text{nn}(q, \mathcal{F}_k(P))$.

IV. **ANIF** (Approximate Nearest Induced Flat): Compute a $(k-1)$-flat $f \in \mathcal{F}_k(P)$, such that $d(q, f) \leq (1 + \varepsilon)d(q, \mathcal{F}_k(P))$.

V. **NIS** (Nearest Induced Simplex): Compute $\text{nn}(q, \Delta_k(P))$.

VI. **ANIS** (Approximate Nearest Induced Simplex): Compute a $(k-1)$-simplex $\Delta \in \Delta_k(P)$, such that $d(q, \Delta) \leq (1 + \varepsilon)d(q, \Delta_k(P))$.

Here, the parameter $k$ corresponds to the sparsity of the solution.

**Remark 4.2.6.** We note that the solutions of NLIF, NIF, and NIS respectively correspond to the solutions of the SLR, Affine SLR and the Convex SLR problems.

4.3 Approximating the Nearest Induced and Nearest Linear Induced Flats

In this section, we show how to solve approximately the online variants of NLIF and NIF problems. The results in this section are further used in 4.4. We start with the simplified case of the uniform star.

4.3.1 Approximating the Nearest Neighbor in a Uniform Star

**Input & task.** We are given a base point $b$, a set $P$ of $n$ points in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. We assume that $\|b - x\| = 1$, for all $x \in P$. The task is to build a data structure that can report quickly, for a query point $q$ that is also at distance one from $b$, the closest $(1 + \varepsilon)$-ANN segment to $q$ in $\text{star}(b, P)$. 

93
**Preprocessing.** The algorithm computes the set \( V = \{ x - b \mid x \in P \setminus \{ b \} \} \), which lies on a unit sphere in \( \mathbb{R}^d \). Next, the algorithm builds a data structure \( DS_V \) for answering \((1 + \varepsilon)\)-ANN queries on \( V \).

**Answering a query.** For a query point \( q \), the algorithm does the following:

(A) Compute \( \tau = q - b \).

(B) Compute \((1 + \varepsilon)\)-ANN to \( \tau \) in \( V \), denoted by \( u \) using \( DS V \).

(C) Let \( y \) be the point in \( P \) corresponding to \( u \).

(D) Return \( \min(d(q, by), 1) \).

**Analysis.**

**Lemma 4.3.1.** Consider a base point \( b \), and a set \( P \) of \( n \) points in \( \mathbb{R}^d \) all on \( S(b, 1) \), where \( S = S(b, 1) \) is the sphere of radius 1 centered at \( b \). Given a query point \( q \in S \), the above algorithm reports correctly the \((1 + \varepsilon)\)-ANN in \( \text{star}(b, P) \). The query time is dominated by the time to perform a single \((1 + \varepsilon)\)-ANN query.

**Proof.** If \( d(q, \text{star}(b, P)) = 1 \) the correctness is obvious. Otherwise, \( d(q, \text{star}(b, P)) < 1 \), and assume for the sake of simplicity of exposition that \( b \) is the origin. Let \( x \) be the nearest neighbor to \( q \) in \( P \), and let \( x' \) be the nearest point on \( bx \) to \( q \). Similarly, let \( y \) be the point returned by the ANN data structure for \( q \), and let \( y' \) be the nearest point on \( by \) to \( q \). Moreover, let \( \alpha = \angle qbx \) and \( \beta = \angle qby \). As \( d(q, \text{star}(b, P)) < 1 \), we can also conclude that \( \alpha \) is smaller than \( \pi/2 \).

We have that \( \|q - x\| \leq \|q - y\| \leq (1 + \varepsilon)\|q - x\| \). Observe that \( \|q - y\| = 2 \sin(\beta/2) \), and \( \|q - x\| = 2 \sin(\alpha/2) \). As such, by the monotonicity of the sin function in the range \([0, \pi/2]\), we conclude that \( \alpha \leq \beta \). This readily implies that \( x \) is the point of \( P \) that minimizes the angle \( \angle qbx \) (i.e., \( \alpha \)), which in turn minimizes the distance to \( q \) on the induced segment. As such, we have \( d(q, \text{star}(b, P)) = \|q - x'\| \).

As for the quality of approximation, first suppose that \( \beta \leq \pi/2 \). Then we have

\[
\frac{\|q - y'\|}{\|q - x'\|} = \frac{\|q - y\| \cos(\beta/2)}{\|q - x\| \cos(\alpha/2)} \leq 1 + \varepsilon, \text{ since } \|q - y\| / \|q - x\| \leq 1 + \varepsilon \text{ and } \alpha/2 \leq \beta/2 \leq \pi/2,
\]

which in turn implies that \( \cos(\beta/2) \leq \cos(\alpha/2) \).
Otherwise, we know that $\beta > \pi/2$ and thus $d(q, b) = 1$. Now if $1 \leq (1 + \varepsilon) \|q - x'\|$ we are done. Thus, we assume that $\|q - x'\| \leq 1/(1 + \varepsilon)$. Also, we have that $\|q - x\| \geq \|q - y\|/(1 + \varepsilon) > \sqrt{2}/(1 + \varepsilon)$ as $\beta > \pi/2$. This would imply that $\sqrt{2}/(1 + \varepsilon) < \|q - x\| = \|q - x'\|/\cos(\alpha/2) \leq 2/\sqrt{2}(1 + \varepsilon)$, as $\alpha \leq \pi/2$, which is a contradiction. Hence, the lemma holds.

\[\square\]

4.3.2 Approximating the Nearest Flat in a Bouquet

**Definition 4.3.2.** For a set $X$ and a point $x$ in $\mathbb{R}^d$, let $x' = \text{nn}(x, X)$. We use $\text{dir}(X, x)$ to denote the unit vector $(x - x')/\|x - x'\|$, which is the direction of $x$ in relation to $X$.

**Input & task.** We are given sets $B$ and $P$ of $k - 1$ and $n$ points, respectively, in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. The task is to build a data structure that can report quickly, for a query point $q$, the $(1 + \varepsilon)$-ANN flat to $q$ in $\text{bqt}(B, P)$, see 4.2.4.

**Preprocessing.** Let $F = f_B$. The algorithm computes the set $V = \{\text{dir}(F, x), -\text{dir}(F, x) \mid x \in P \setminus B\}$, which lies on a $d-k+2$ dimensional unit sphere in $\mathbb{R}^{d-k+1}$, and then builds a data structure $\text{DS}_V$ for answering (standard) ANN queries on $V$. 

95
Answering a query. For a query point \(q\), the algorithm does the following:

(A) Compute \(\tau = \text{dir}(F, q)\).

(B) Compute ANN to \(\tau\) in \(V\), denoted by \(u\) using the data structure \(\text{DS}_V\).

(C) Let \(x\) be the point in \(P\) corresponding to \(u\).

(D) Return the distance \(d(q, f_{B \cup \{x\}})\).

Analysis.

Definition 4.3.3. For two sets \(X, Y \subseteq \mathbb{R}^d\), let \(\text{proj}_X(Y) = \{\text{nn}(q, X) \mid q \in Y\}\) be the projection of \(Y\) on \(X\).

Observation 4.3.4. Consider two affine subspaces \(F \subseteq G\) with a base point \(b \in F\), and the orthogonal complement affine subspace \(F^\perp = \{b + \tau \mid \langle \tau, u - v \rangle = 0 \text{ for all } u, v \in F, \tau \in \mathbb{R}^d\}\). For an arbitrary point \(q \in \mathbb{R}^d\), let \(q^\perp = \text{proj}_{F^\perp}(q)\). We have that \(d(q, G) = d(q^\perp, \text{proj}_{F^\perp}(G))\).

Using the notation of Assumption 4.2.3 and Definition 4.2.4, we have the following:

Lemma 4.3.5 (ANN flat in a bouquet). Given sets \(B\) and \(P\) of \(k - 1\) and \(n\) points, respectively, in \(\mathbb{R}^d\), and a parameter \(\varepsilon > 0\), one can preprocess them, using a single ANN data structure, such that given a query point, the algorithm can compute an \((1 + \varepsilon)\)-ANN to the closest \((k - 1)\)-flat in \(\text{bqt}(B, P)\). The algorithm space and preprocessing time is \(O(S_{\text{ANN}}(n, \varepsilon))\), and the query time is \(O(T_{\text{ANN}}(n, \varepsilon))\).

Proof. The construction is described above, and the space and query time bounds follow directly from the algorithm description. As for correctness, pick an arbitrary base point \(b \in B\), and let \(F^\perp\) be the orthogonal complement affine subspace to \(F\) passing through \(b\). Let \(P^\perp = \text{proj}_{F^\perp}(P)\), and observe that \(\text{proj}_{F^\perp}(B)\) is the point \(b\). In particular, the projection of \(B = \text{bqt}(B, P)\) to \(F^\perp\) is the bouquet of lines \(B^\perp = \text{bqt}(\{b\}, P^\perp)\). Applying 4.3.4 to each flat of \(B\), implies that \(d(q, B) = d(q^\perp, B^\perp)\), where \(q^\perp = \text{proj}_{F^\perp}(q)\). Let \(S\) be the sphere of radius \(r = \|q^\perp - b\|\) centered at \(b\). Clearly, the closest line in the bouquet, is the closest point to the uniform star formed by clipping the lines of \(B^\perp\) to \(S\). Since all the lines of \(B^\perp\) pass through \(b\),
scaling space around \( b \) by some factor \( \alpha \), just scales the distances between \( q \) and \( B^\perp \) by \( x \). As such, scale space so that \( r = 1 \). But then, this is a uniform star with radius 1, and the algorithm of 4.3.1 applies (which is exactly what the current algorithm is doing). Thus, it correctly identifies a line in the bouquet that realizes the desired approximation, implying the correctness of the query algorithm. 

4.3.3 The Result

Here, we show simple algorithms for the ANIF and the ANLIF problems by employing 4.3.5. We assume \( \varepsilon > 0 \) is a prespecified approximation parameter.

Approximating the NIF. As discussed earlier, the goal is to find an approximately closest \((k-1)\)-dimensional flat that passes through \( k \) points of \( P \), to the query. To this end, we enumerate all possible \( k-1 \) subsets of points of \( B \subset_k P \), and build for each such base set \( B \), the data structure of 4.3.5. Given a query, we compute the ANN flat in each one of these data structures, and return the closest one found.

**Theorem 4.3.6.** The aforementioned algorithm computes an \((1 + \varepsilon)\)-ANN to the closest \((k-1)\)-flat in \( \mathcal{F}_k(P) \), see 4.2.5.

The space and preprocessing time is \( O(n^{k-1}S_{\text{ANN}}(n, \varepsilon)) \), and the query time is \( O(n^{k-1}T_{\text{ANN}}(n, \varepsilon)) \).

Approximating the NLIF. The goal here is to find an approximately closest \( k \)-dimensional flat that passes through \( k \) points of \( P \) and the origin \( 0 \), to the query. We enumerate all possible \( k-1 \) subsets of points of \( B' \subset_{k-1} P \), and build for each base set \( B = B' \cup \{0\} \), the data structure of 4.3.5. Given a query, we compute the ANN flat in each one of these data structures, and return the closest one found.

**Theorem 4.3.7.** The aforementioned algorithm computes an \((1 + \varepsilon)\)-ANN to the closest \( k \)-flat in \( \mathcal{L}_k(P) \), see 4.2.5, with space and preprocessing time of \( O(n^{k-1}S_{\text{ANN}}(n, \varepsilon)) \), and the query time of \( O(n^{k-1}T_{\text{ANN}}(n, \varepsilon)) \).
4.4 Approximating the Nearest Induced Simplex

In this section we consider the online variant of the ANIS problem. Here, we are given the parameter $k$, and the goal is to build a data structure, such that given a query point $q$, it can find an $(1 + \varepsilon)$-ANN induced $(k - 1)$-simplex.

As before, we would like to fix a set $B$ of $k - 1$ points and look for the closest simplex that contains $B$ and an additional point from $P$. The plan is to filter out the simplices for which the projection of the query on to them falls outside of the interior of the simplex. Then we can use the algorithm of the previous section to find the closest flat corresponding to the feasible simplices (the ones that are not filtered out).

First we define a canonical space and map all these simplices and the query point to a unique $(k + 1)$-dimensional space. As it will become clear shortly, the goal of this conversion is to have a common lower dimensional space through which we can find all feasible simplices using range searching queries.

4.4.1 Simplices and Distances

Canonical realization.

In the following, we fix a sequence $B = (x_1, \ldots, x_{k-1})$ of $k - 1$ points in $\mathbb{R}^d$. We are interested in arguing about simplices induced by $k + 1$ points, i.e., $B$, an additional input point $x_k$, and a query point $q$. Since the ambient dimension is much higher (i.e., $d$), it would be useful to have a common canonical space, where we can argue about all entities.

**Definition 4.4.1.** For a given set of points $B$, let $F = f_B$. Let $x \notin F$ be a given point in $\mathbb{R}^d$, and consider the two connected components of $f_{B \cup \{x\}} \setminus F$, which are **halfflats**. The halfflat containing $x$ is the **positive halfflat**, and it is denoted by $f^+(B, x)$.

Fix some arbitrary point $s^* \in \mathbb{R}^d \setminus F$, and let $G = f^+(B, s^*)$ be a canonical such halfflat. Similarly, for a fixed point $s^{**} \in \mathbb{R}^d \setminus f_{B \cup \{s^*\}}$, let $H = f^+(B \cup s^*, s^{**})$. Conceptually, it is convenient to consider $H = \mathbb{R}^{k-2} \times \mathbb{R} \times \mathbb{R}^+$, where the first $k - 2$
coordinates correspond to $F$, and the first $k - 1$ coordinates correspond to $G$ (this can be done by applying a translation and a rotation that maps $H$ into this desired coordinates system). This is the **canonical parameterization** of $H$.

First, the following observation formalizes this simple intuition: given a $(k - 1)$ dimensional halfflat $G$ passing through $B$, a point on $G$ is uniquely identified by its distances from the points in $B$.

**Observation 4.4.2.** Given a sequence of distances $\ell = (\ell_1, \ldots, \ell_{k-1})$, there might be only one unique point $x = x_G(\ell) \in G$, such that $\|x - x_i\| = \ell_i$, for $i = 1, \ldots, k - 1$. Such a point might not exist at all\(^4\).

Next, given $G$ and $H$, a point $q$ and a value $\ell < d(q, F)$, we aim to define the points $q_G(\ell)$ and $q_H(\ell)$. Consider a point $q \in \mathbb{R}^d \setminus F$ (not necessarily the query point), and consider any positive $(k - 1)$-halfflat $g$ that contains $B$, and is in distance $\ell$ from $q$. Furthermore assume that $\ell = d(q, g) < d(q, F)$. Let $q_g$ be the projection of $q$ to $g$. Observe that, by the Pythagorean theorem, we have that $d_i = \|q_g - x_i\| = \sqrt{\|q - x_i\|^2 - \ell^2}$, for $i = 1, \ldots, k - 1$. Thus, the above observation implies, that the canonical point $q_G(\ell) := x_G(d_1, \ldots, d_{k-1})$ (see 4.4.2) is uniquely defined. Note that this is somewhat counterintuitive as the flat $g$ and thus the point $q_g$ are not uniquely defined. Similarly, there is a unique point $q_H(\ell) \in H$, such that: (i) the projection of $q_H(\ell)$ to $G$ is the point $q_G(\ell)$, (ii) $\|q_H(\ell) - q_G(\ell)\| = \ell$, and these two also imply that (iii) $\|q_H(\ell) - x_i\| = \|q - x_i\|$, for $i = 1, \ldots, k - 1$.

Therefore, given $G$ and $H$, a point $q$ and a value $\ell < d(q, F)$, the points $q_G(\ell)$ and $q_H(\ell)$ are uniquely defined. Intuitively, for a halfflat that passes through $B$ and is at distance $\ell$ from the query, $q_G(\ell)$ models the position of the projection of the query onto the halfflat, and $q_H(\ell)$ models the position of the query point itself with respect to this halfflat. Next, we prove certain properties of these points.

\(^4\) **Trilateration** is the process of determining the location of $x \in G$ given $\ell$. **Triangulation** is the process of determining the location when one knows the angles (not the distances).
Orbits.

Definition 4.4.3. For a set of points $B$ in $\mathbb{R}^d$, define $\Phi_B$ to be the open set of all points in $\mathbb{R}^d$, such that their projection into $F$ lies in the interior of the simplex $\triangle_B = \text{ConvexHull}(B)$. The set $\Phi_B$ is a prism.

Consider a query point $q \in \Phi_B$, and its projection $q_B = \text{nn}(q, F)$. Let $r = r_B(q) = \|q - q_B\|$ be the radius of $q$ in relation to $B$. Using the above canonical parameterization, we have that $q_G(0) = (q_B, r)$, and $q_H(0) = (q_G(0), 0) = (q_B, r, 0)$. More generally, for $\ell \in [0, r]$, we have

$$q_G(\ell) = (q_B, \sqrt{r^2 - \ell^2}) \quad \text{and} \quad q_H(\ell) = (q_B, \sqrt{r^2 - \ell^2}, \ell). \quad (4.1)$$

The curve traced by $q_H(\ell)$, as $\ell$ varies from 0 to $r$, is the orbit of $q$ – it is a quarter circle with radius $r$. Next, a monotonicity property is proved in the following lemma which is used as a basis for binary search over the value of $\ell$, and is described later in this section.

Lemma 4.4.4. (i) Define $\hat{q}(\ell) := (\sqrt{r^2 - \ell^2}, \ell)$, and consider any point $x = (x, 0)$, where $x \geq 0$. Then, the function $d(\ell) = \|\hat{q}(\ell) - x\|$ is monotonically increasing for $\ell \in [0, r]$.

(ii) For any point $x$ in the halfflat $G$, the function $\|q_H(\ell) - x\|$ is monotonically increasing.

Proof. (i) $D(\ell) := (d(\ell))^2 = \left(\sqrt{r^2 - \ell^2} - x\right)^2 + \ell^2 = r^2 - 2x\sqrt{r^2 - \ell^2} + x^2$, and clearly this is a monotonically increasing function for $\ell \in [0, r]$.

(ii) Let $x_B = \text{nn}(x, F)$, and let $x \geq 0$ be a number such that in the canonical representation, we have that $x = (x_B, x)$. Using 4.1, we have

$$D(\ell) := \|q_H(\ell) - x\|^2 = \|q_B, \sqrt{r^2 - \ell^2}, x - \ell\|^2 = \|q_B - x_B\|^2 + (d(\ell))^2,$$

and the claim readily follows from (i) as $q_B$ and $x_B$ do not depend on $\ell$. □
Distance to a simplex via distance to the flat.

**Definition 4.4.5.** Given a point \( q \), and a distance \( \ell \), let \( \triangle_G(q, \ell) \) be the unique simplex in \( G \), having the points of \( B \) and the point \( q_G(\ell) \) as its vertices. Similarly, let \( \triangle_G(q) = \triangle_G(q, 0) \).

The following lemma which is the heart of our datastructure intuitively provides the necessary and sufficient conditions for a simplex to be feasible.

**Lemma 4.4.6.** Given a query point \( q \in \Phi_B \), and a point \( x_k \in P \setminus B \), for a number \( 0 < x \leq d(q, F) \) we have

\[
(A) \quad q_G(x) \in \triangle_G(x_k) \quad \text{and} \quad d(q, f^+(B, x_k)) \leq x \implies d(q, \triangle_{B \cup \{x_k\}}) \leq x.
\]

\[(B) \quad d(q, \triangle_{B \cup \{x_k\}}) \leq x \quad \text{and} \quad q \in \Phi_{B \cup \{x_k\}} \implies q_G(x) \in \triangle_G(x_k) \quad \text{and} \quad d(q, f^+(B, x_k)) \leq x.
\]

**Proof.** (A) Let \( q^* = \text{nn}(q, f^+(B, x_k)) \). If \( q^* \in F \) then \( q^* \in \text{int}(\triangle_B) \) (see 4.2.1), because \( q \in \Phi_B \). But then, \( d(q, \triangle_{B \cup \{x_k\}}) = \|q - q^*\| \leq x \), as desired.

Observe that by definition, the projection of \( q_H(x) \) to \( G \) is the point \( q_G(x) \), and since \( q_G(x) \in \triangle_G(x_k) \), it follows that \( d(q_H(x), \triangle_G(x_k)) = \|q_G(x) - q_H(x)\| = x \). As such, let \( \ell = d(q, f^+(B, x_k)) \), and observe that by definition of the parameterization \( d(q, \triangle_{B \cup \{x_k\}}) = d(q_H(\ell), \triangle_G(x_k)) \). However, since \( \ell \leq x \) and by 4.4.4 (ii), we have that \( d(q_H(\ell), \triangle_G(x_k)) \leq d(q_H(x), \triangle_G(x_k)) = x \) which completes the proof.

(B) Let \( \ell := d(q, f^+(B, x_k)) \), and note that since \( q \in \Phi_{B \cup \{x_k\}} \), we have that \( \ell = d(q, \triangle_{B \cup \{x_k\}}) \leq x \), proving one part. Also, by definition of the parameterization, we have that \( d(q_H(\ell), \triangle_G(x_k)) = d(q, \triangle_{B \cup \{x_k\}}) \) which is equal to \( \ell \). As by definition \( q_G(\ell) \) is the projection of \( q_H(\ell) \) onto \( G \) and that \( \|q_H(\ell) - q_G(\ell)\| = \ell \), we get that \( q_G(\ell) \) is in fact equal to \( \text{nn}(q_H(\ell), \triangle_G(x_k)) \), and therefore \( q_G(\ell) \in \triangle_G(x_k) \).

Moreover, 4.1 implies that \( q_G(y) \), for \( y \in [0, r_B(q)] \), moves continuously and monotonically on a straight segment starting at \( q_G(0) \), and as \( y \) increases, it moves towards \( q_G(r) = q_B = \text{nn}(q, \triangle_B) \). As we proved here \( \ell \leq x \), and by the assumption of the lemma \( x \leq d(q, F) = r \). Thus we infer that \( q_G(x) \) lies on the segment \( q_B q_G(\ell) \). As \( q \in \Phi_B \) we know that \( q_B \in \triangle_G(x_k) \), and we also proved here that \( q_G(\ell) \in \triangle_G(x_k) \). Thus, by the convexity of \( \triangle_G(x_k) \), we have that \( q_G(x) \in q_B q_G(\ell) \subseteq \triangle_G(x_k) \). \( \square \)
4.4.2 Approximating the Nearest Page in a Book

Definiton 4.4.7. We are given a set $P$ of $n$ points in $\mathbb{R}^d$, and a sequence $B = \{\}^{k-1}$ points. We are interested in the set of simplices having $B$ and one additional point from $P$; that is,

$$\Delta = \Delta(B, P) = \{\triangle_{B \cup \{x\}} \mid x \in P \setminus B\}.$$ 

We refer to the set $\Delta$ as the book induced by $(B, P)$, and to a single simplex in this book, as a page.

The task at hand, is to preprocess $\Delta$ for ANN queries, as long as (i) the nearest point lies in the interior of one of these simplices and (ii) $q \in \Phi_B$ (we will clarify it later in 4.4.3 how we will take care of these conditions). To this end, we consider the canonical representation of this set of simplices

$$\Delta_G = \{\triangle_G(x) \mid x \in P \setminus B\}. \quad (4.2)$$

Idea. The algorithm follows 4.4.6 (A). Given a query point, using standard range-searching techniques, we extract a small number of canonical sets of the points, that in the parametric space, their simplex contains the parameterized query point. This is described in 4.4.2. For each of these canonical sets, we use the data structure of 4.3.5 to quickly query each one of these canonical sets for their nearest positive flat (see 4.4.12 below). This would give us the desired ANN.

Reporting all simplices containing a point.

Definiton 4.4.8. Let $B = (x_1, \ldots, x_{k-1})$ be a sequence of $k - 1$ points in $\mathbb{R}^d$. For a point $x \in \mathbb{R}^d$, consider the $(k-1)$-simplex $\triangle_{B \cup \{x\}}$, which is a full dimensional simplex in the flat $f_{B \cup \{x\}}$ (see 4.2.1). The base angles of $x$ (with respect to $B$), is the $(k-1)$-tuple $\alpha_B(x) = (\alpha_1(x), \ldots, \alpha_{k-1}(x))$, where $\alpha_i(x)$ is the dihedral angle between the facet $\triangle_{B \cup \{x\}\setminus \{x_i\}}$ and the base facet $\triangle_B$. See the figure on the right, where $k = 3$. 

102
**Observation 4.4.9** (Inclusion and base angles). Let $B$ be a set of $k - 1$ points in $\mathbb{R}^{k-1}$ all with their $(k-1)$th coordinate being zero, and let $x$ be an additional point with its $(k-1)$th coordinate being a positive number. Then, for a point $q \in \mathbb{R}^{k-1}$, we have that $q \in \triangle_{B \cup \{x\}} \iff \alpha_B(q) \leq \alpha_B(x)$ (i.e., $(\forall i : \alpha_i(q) \leq \alpha_i(x)$).

**Lemma 4.4.10.** Given a set $n$ of $(k-1)$-simplices $\Delta_G$ in $\mathbb{R}^{k-1}$, that all share common $k - 1$ vertices, one can build a data structure of size $O(n \log^{k-1} n)$, such that given a query point $q \in \mathbb{R}^{k-1}$, one can compute $O(\log^{k-1} n)$ disjoint canonical sets, such that the union of these sets, is the set of all simplices in $\Delta_G$ that contain $q$. The query time is $O(\log^{k-1} n)$.

**Proof.** By a rigid transformation of the space, we can assume that $F$ is the hyperplane $x_{k-1} = 0$, and furthermore, all the vertices of $\Delta_G$ have $x_{k-1} \geq 0$ (we can handle the simplices $x_{k-1} \leq 0$ in a similar separate data structure). Let $P$ be the vertices of $\Delta_G$ not lying on $F$. We generate the corresponding set of base angles $P_L = \{\alpha_B(x) \mid x \in P\}$. Preprocess this set for orthogonal range searching, say, using range-trees [BCKO08]. Given a query point $q$, by 4.4.9, the desired simplices correspond to all points in $x \in P$, such that $\alpha_B(x) \geq \alpha_B(q)$, which is an unbounded box query in the range tree of $P_L$, with the aforementioned performance. \hfill \Box

**Remark 4.4.11.** The data structure of 7.4.4 can be used to report all simplices that contain a specific point $x$, and do not contain another point $x'$, which is vertically above $x$ (i.e., the same point with larger $(k-1)$th coordinate). This corresponds to $k$ (possibly unbounded) box queries instead of quadrant query in the orthogonal data structure. The query time and number of canonical sets will be multiplied by at most...
The space bound remains the same. Moreover, we ensure these set of $k$ boxes are disjoint.

Proof. If $x'$ is vertically above $x$, we have that $\alpha_B(x') \geq \alpha_B(x)$. Let us denote $\alpha_B(x)$ by $\alpha$ and $\alpha_B(x')$ by $\alpha'$. We will then for each $1 \leq i \leq k-1$ ask the following box query

$$[[\alpha'_i, \infty], [\alpha_{i-1}, \infty], [\alpha_i, \alpha'_i], [\alpha_{i+1}, \infty], \ldots, [\alpha'_{k-1}, \infty]]$$

It is easy to show that these boxes are disjoint and their union will correspond to all simplices that contain $x$ but not $x'$. As each of these boxes correspond to poly-logarithmic number of canonical sets, in total we will get $O(k \log^{k-1} n)$ canonical sets.

\qed

Data structure and correctness.

Remark 4.4.12. For a set of points $P$ and a base set $B$, consider the set of positive halfspaces (the positive bouquet) $\text{bqt}^+(B, P) = \{f^+(B, x) \mid x \in P \setminus B\}$. We can preprocess such a set for ANN queries readily, by using the data structure of 4.3.5. The only modification is that for every positive flat we assign one vector (in the positive direction), instead of two vectors in both directions which we put in the data structure of 4.3.2.

Preprocessing. The algorithm computes the set of canonical simplices $\Delta_G$, see 4.2. Next, the algorithm builds the data structure of 7.4.4 for this set of simplices. For each canonical set $V$ in this data structure, for the corresponding set of original points, we build the data structure of 4.4.12 to answer ANN queries on the positive bouquet $\text{bqt}^+(B, V)$. (Observe that the total size of these canonical sets is $O(n \log^{k-1} n)$.)

Answering a query. Given a query point $q \in \Phi_B$, the algorithm computes its projection $q_B = \text{nn}(q, F)$, where $F = f_B$. Let $r = \|q - q_B\|$ be the radius of $q$. The desired ANN distance is somewhere in the interval $[0, r]$, and the algorithm maintains
an interval \([\alpha, \beta]\) where this distance lies, and uses binary search to keep pruning away on this interval, till reaching the desired approximation.

Observe that for every point \(x \in P\), there is a critical value \(\gamma(x)\), such that for \(x \geq \gamma(x)\), the parameterized point \(q_G(x)\) is inside the simplex \(\triangle_G(x)\), and is outside if \(x < \gamma(x)\), see 4.4.5. Note that this statement only holds for queries in \(\Phi_B\) (otherwise it could have been false on simplices \(\triangle_{B \cup \{x\}}\) with obtuse angles, see 4.4.14 for handling the case of \(q \notin \Phi_B\)).

Now, by 4.4.11, we can compute a polylogarithmic number of canonical sets, such that the union of these sets, are (exactly) all the points with critical values in the range \([\alpha, \beta]\). As long as the number of critical values is at least one, we randomly pick one of these values (by sampling from the canonical sets – one can assume each canonical set is stored in an array), and let \(\gamma\) be this value. We have to decide if the desired ANN is smaller or larger than \(\gamma\). To this end, we compute a representation, by polylogarithmic number of canonical sets, of all the points of \(P\) such that their simplex contain the parameterized point \(q_G(\gamma)\), using 7.4.4. For each such canonical set, we compute the approximate closest positive halfflat by 4.4.12. Let \(\tau\) be the minimum distance of such a halfflat computed. If this distance is smaller than \(\gamma\), then the desired ANN is smaller than \(\gamma\), and the algorithm continues the search in the interval \([\alpha, \gamma]\), otherwise, the algorithm continues the search in the interval \([\gamma, \beta]\).

After logarithmic number of steps, in expectation, we have an interval \([\alpha', \beta']\), that contains no critical value in it, and the desired ANN distance lies in this interval. We compute the ANN positive flats for all the points that their parameterized simplex contains \(q_G(\beta')\), and we return this as the desired ANN distance.

**Correctness.** For the sake of simplicity, we first assume the ANN data structure returns the exact nearest-neighbor. 4.4.6 (A) readily implies that whatever simplex is being returned, its distance from the query point is the closest, as claimed by the data structure. The other direction is more interesting – consider the unknown point \(x_k\), such that the (desired) nearest point to the query point lies in the interior of the simplex \(\triangle_{B \cup \{x_k\}}\) (we made this assumption at the beginning of 4.4.2). 4.4.6
(B) implies that the distance to this simplex is always going to be inside the active interval, as the algorithm searches (if not, then the algorithm had found even closer simplex, which is a contradiction).

To adapt the proof to the approximate case, suppose that the data structure of 4.4.12 returns a \((1 + \varepsilon)\) approximate nearest neighbor. Consider some iteration of the algorithm. Let \(P \subseteq P\) be the set of all points \(x\) such that the simplex \(\triangle_G(x)\) contains \(q_G(\gamma)\), and let \(A\) be the set of simplices \(\triangle_{B \cup \{x\}}\) corresponding to the points in \(P\), and \(F\) be the set of half-flats \(f^+(B, x)\) corresponding to the points in \(P\).

Suppose that \(\ell^*\) is the minimum distance of the half-flats in \(F\) to the query, and let \(\tau\) be the distance of the half-flat reported by the ANN data structure to the query. Thus, we have \(\tau \leq \ell^*(1 + \varepsilon)\). Note that, if \(\tau < \gamma\), we know that the optimal distance \(\ell^*\) is also less than \(\gamma\) and recursing on the interval \([\alpha, \gamma)\) works as it satisfies the precondition. However, if \(\tau \geq \gamma\), we know that either \(\ell^* \geq \gamma\) as well, in which case the recursion would work for the same reason, or \(\ell^* < \gamma \leq \tau \leq \ell^*(1 + \varepsilon)\).

In the latter case, let \(p\) be the reported point corresponding to \(\tau\). We know that the distance of the query to the half-flat \(f^+(B, x)\) is \(\tau\) which is at most \(\ell^*(1 + \varepsilon)\). Now, if the simplex \(\triangle_G(x)\) contains the point \(q_G(\tau)\), as well, then the distance of the query to the simplex \(\triangle_{B \cup \{x\}}\) is equal to its distance to the half-flat \(f^+(B, x)\) which is \(\tau \leq \ell^*(1 + \varepsilon)\). Therefore, we can assume that \(q_G(\tau)\) is outside of the simplex \(\triangle_G(x)\). However in this case, we get that the distance of the query to the simplex \(\triangle_{B \cup \{x\}}\) is as follows.

\[
    d(q, \triangle_{B \cup \{x\}}) = d(q_H(\tau), \triangle_G(x)) \leq d(q_H(\tau), q_G(\gamma)) = \sqrt{\tau^2 + d(q_G(\tau), q_G(\gamma))^2} = \sqrt{\tau^2 + (\sqrt{\tau^2 - \gamma^2} - \sqrt{\tau^2 - \tau^2})^2} \leq \sqrt{\tau^2 + \tau^2 - \gamma^2} \leq \ell^* \sqrt{2(1 + \varepsilon)^2 - 1} < \ell^*(1 + 2\varepsilon)
\]

Where we use the fact that \(q_G(\gamma) \in \triangle_G(x)\), and 4.1, and the fact that \(\gamma \leq \tau \leq r\). The above equation means that the distance of the query to the simplex \(\triangle_{B \cup \{x\}}\) is itself approximately the closest. Thus, it is enough to change the algorithm so that at each iteration, it checks the distance of simplex it finds to the query and reports the best one found in all iterations.
**Query time.** In order to sample a value from \([\alpha, \beta]\) the algorithm uses 4.4.11 which has running time of \(O(k \log^{k-1} n)\). Moreover, the algorithm performs \(O(\log^{k-1} n)\) ANN queries in each iteration of the search corresponding to 7.4.4. As the search takes \(O(\log n)\) iterations in expectation (and also with high-probability), the query time is \(O(T_{ANN}(n, \varepsilon) \log^k n)\). Note that this assumes \(k\) is smaller than \(T_{ANN}(n, \varepsilon)\) which holds as we are working in a low sparsity regime.

**Lemma 4.4.13** (Approximate nearest induced page). Given a set \(P\) of \(n\) points in \(\mathbb{R}^d\), and a set \(B\) of \(k - 1\) points, and a parameter \(\varepsilon > 0\), one can preprocess them, such that given a query point, the algorithm can compute an \((1+\varepsilon)\)-ANN to the closest page in \(\Delta(B, P)\), see 4.4.7. This assumes that (i) the nearest point to the query lies in the interior of the nearest page, and (ii) \(q \in \Phi_B\). The algorithm space and preprocessing time is \(O(S_{ANN}(n, \varepsilon) \log^k n)\), and the query time is \(O(T_{ANN}(n, \varepsilon) \log^k n)\).

**4.4.3 Result: Nearest Induced Simplex**

The idea is to use brute-force to handle the distance of the query to the \(\leq (k - 2)\)-simplices induced by the given point set which takes \(O(n^{k-1})\) time. As such, the remaining task is to handle the \((k - 1)\)-simplices, and thus we can assume that the nearest point to the query lies in the interior of the nearest simplex, as desired by 4.4.13. To this end, we generate the \(\binom{n}{k-1} = O(n^{k-1})\) choices for \(B \subseteq P\), and for each one of them we build the data structure of 4.4.13, and query each one of them, returning the closest one found.

**Remark 4.4.14.** Note that for a set of \(k\) points \(A \subset_k P\), if the projection of the query onto the simplex \(\triangle_A\) falls inside the simplex, i.e. \(q \in \Phi_A\), then there exists a subset of \(k - 1\) points \(B \subset_{k-1} A\) such that the projection of the query onto the simplex \(\triangle_B\) falls inside the simplex, i.e., \(q \in \Phi_B\). Therefore, either the brute-force component of the algorithm finds an ANN, or there exists a set \(B\) for which the corresponding data structure reports the correct ANN.

We thus get the following result.
Theorem 4.4.15 (Convex SLR). Given a set $P$ of $n$ points in $\mathbb{R}^d$, and parameters $k$ and $\varepsilon > 0$, one can preprocess them, such that given a query point, the algorithm can compute an $(1 + \varepsilon)$-ANN to the closest $(k - 1)$-simplex in $\Delta_k(P)$, see 4.2.5. The algorithm space and preprocessing time is $O(n^{k-1}S_{\text{ANN}}(n,\varepsilon) \log^k n)$, and the query time is $O(n^{k-1}T_{\text{ANN}}(n,\varepsilon) \log^k n)$.

4.5 Lower bound

In this section, we reduce the $k$-sum problem to offline variant of all of our problems, ANLIF, ANIF and ANIS problems, providing an evidence that the time needed to solve these problems is $\tilde{\Omega}(n^{k/2}/e^k)$. In the $k$-sum problem, we are given $n$ integer numbers $a_1, \ldots, a_n$. The goal is to determine if there exist $k$ numbers among them $a_{i_1}, \ldots, a_{i_k}$ such that their sum equals zero. The problem is conjectured to require $\Omega(n^{[k/2]}/\log^\Theta(1)n)$ time, see [PW10], Section 5.

We reduce this problem as follows. Let $P = \{v_1, \ldots, v_n\}$ be a set of $n$ vectors of dimension $k+1$. More precisely, each $v_i \in \mathbb{R}^{k+1}$ has its first coordinate equal to $a_i$ and all the other coordinates are 0 except for one coordinate chosen uniformly at random from $\{2, \ldots, k+1\}$, whose value we set to 1. The query is also a vector of dimension $(k+1)$ and is of the form $q = [0, 1/k, 1/k, \ldots, 1/k]^T$. We query the point $q$ and let $v_{i_1}, \ldots, v_{i_k}$ be the points corresponding to the approximate closest flat/simplex reported by the algorithm. We then check if $\sum_{j=1}^k a_{i_j} = 0$, and if so, we report $\{a_{i_1}, \ldots, a_{i_k}\}$. Otherwise, we report that no such $k$ numbers exist. Next, we prove the correctness via the following two lemmas.

Lemma 4.5.1. If there is no solution to the $k$-sum problem, that is if there is no set $\{i_1, \ldots, i_k\}$ such that $\sum_{j=1}^k a_{i_j} = 0$, then the distance of the query to the closest flat/simplex is non-zero.

Proof. Suppose that the distance of the query to the closest flat/simplex is zero. Thus, there exist $k$ vectors $v_{i_1}, \ldots, v_{i_k}$ and the coefficients $c_1, \ldots, c_k$, such that $c_1v_{i_1} + \cdots + c_kv_{i_k} = q$. Let $t_1, \ldots, t_k$ be the coordinates ($t_i$ has value from 2 to $(k+1)$) such
that \( v_{ij} \) has nonzero value in its \( t_j \)th coordinate. Note that \( q \) has exactly \( k \) nonzero coordinates, and each \( v_{ij} \) has exactly one non-zero coordinate from 2 to \((k + 1)\)th coordinates, and there are \( k \) such vectors. Thus \( t_1, \cdots, t_k \) should be a permutation from 2 to \( k + 1 \). Therefore all \( c_j \)'s should be equal to \( 1/k \). Hence, we also have that 
\[
a_{i_1} + \cdots + a_{i_k} = 0
\]
which is a contradiction. Therefore the lemma holds.

**Lemma 4.5.2.** If there exist \( a_{i_1}, \cdots, a_{i_k} \) such that \( \sum_{j=1}^{k} a_{ij} = 0 \), then with probability \( e^{-k} \) the solution of the NIF/NIS/NLIF would be a flat/simplex which contains the query \( q \).

**Proof.** We consider the sum of \( \frac{1}{k} v_{i_1} + \cdots + \frac{1}{k} v_{i_k} \) and show that it equals \( q \) with probability \( e^{-k} \). Let \( t_j \) be the position (from 2 to \((k + 1)\)) of the coordinate with value 1 in vector \( v_{i_j} \). Then if \( t_1, \cdots, t_k \) is a permutation from 2 to \( k + 1 \), we have that 
\[
\sum_{j=1}^{k} \frac{1}{k} v_{ij} = q
\]
and thus the solution of the NIF/NIS/NLIF would contain \( q \) (notice that the coefficients are positive and they sum to 1, so they satisfy the required constraints). The probability that \( t_1, \cdots, t_k \) is a permutation is 
\[
\frac{k!}{k^k} \approx \frac{\sqrt{2\pi k (k/e)^k}}{k^k} \geq e^{-k}.
\]

Therefore we repeat this process \( e^k \) times, and if any of the reported flats/simplices contained the query point, then we report the corresponding solution \( \{a_{i_1}, \cdots, a_{i_k}\} \). Otherwise, we report that no such \( a_{i_1}, \cdots, a_{i_k} \) exists. This algorithm reports the answer correctly with constant probability by the above lemmas. Moreover as the algorithm needs to detect the case when the optimal distance is zero or not, this lower bound works for any approximation of the problem. Thus we get the following theorem.

**Theorem 4.5.3.** There is an algorithm for the \( k \)-sum problem with the running time bounded by \( O(e^k) \) times the required time to solve any of the three variants of the approximate SLR problem.

### 4.5.1 Connection to Hopcroft’s Problem

In the Hopcroft’s problem, we are given two sets \( U \) and \( V \), each consisting of \( N \) vectors in \( \mathbb{R}^d \) and the goal is to check whether there exists \( u \in U \) and \( v \in V \) such
that $u$ and $v$ are orthogonal.

Given an instance of the Affine SLR with $n$ points and $k = 4$, we proceed as follows. Suppose that the input is a set of $n$ points in $\mathbb{R}^d$ and the query is the origin $0$. Moreover, suppose that $d = 4$. So the goal is to decide whether there exist four points $a, b, c, d$ such that the three dimensional flat that passes through them also passes through the origin. This is equivalent to checking whether the determinant of the matrix which is formed by concatenating these four points as its columns, is zero or not. We can pre-process pairs of points to solve it fast.

Take all $\binom{n}{2}$ pairs of points $a = (a_1, a_2, a_3, a_4)$, $b = (b_1, b_2, b_3, b_4)$ and pre-process them by constructing a vector $u$ in 24 dimensional space such that $u = (a_1b_2, -a_1b_2, -a_1b_3, a_1b_3, \ldots, -a_4b_3, a_4b_3)$ and let $U$ be the set of such vectors $u$. Also, for each pair of points $c = (c_1, c_2, c_3, c_4)$, $d = (d_1, d_2, d_3, d_4)$, we construct $v = (c_3d_4, c_4d_3, c_2d_4, c_4d_2, \ldots, c_1d_2, c_2d_1)$ and let $V$ be the set of $\binom{n}{2}$ such vectors $v$. It is easy to check that the determinant is zero, if and only if the inner product of $u$ and $v$ is zero.

Thus, we have two collections $U$ and $V$ of $N = \binom{n}{2}$ vectors in $\mathbb{R}^{24}$ and would like to check whether there exists two points, one from each collection, that are orthogonal. So any lower bound better than $\omega(n^{k/2}) = \omega(n^2)$ implies a super linear lower bound $\omega(N)$ for the Hopcroft’s problem.
Chapter 5

Simultaneous Nearest Neighbor Search

5.1 Introduction and Background

One of the major drawbacks of the current algorithms for the nearest neighbor problem is their inability to support and exploit structure in query sets that is often present in applications. Specifically, in many applications (notably in computer vision), queries issued to the data structure are not unrelated but instead correspond to samples taken from the same object. For example, queries can correspond to pixels or small patches taken from the same image. To ensure consistency, one needs to impose “compatibility constraints” that ensure that related queries return similar answers. Unfortunately, standard nearest neighbor data structures do not provide a clear way to enforce such constraints, as all queries are processed independently of each other.

To address this issue, we introduce the Simultaneous Nearest Neighbor Search (SNN) problem. Given \( k \) simultaneous query points \( q_1, q_2, \cdots, q_k \), the goal of a SNN data structure is to find \( k \) points (also called labels) \( p_1, p_2, \cdots, p_k \) in \( P \) such that (i) \( p_i \) is close to \( q_i \), and (ii) \( p_1, \cdots, p_k \) are “compatible”. Formally, the compatibility is defined by a graph \( G = (Q, E) \) with \( k \) vertices which is given to the data structure, along with an ordered set of query points \( Q = (q_1, \cdots, q_k) \). Furthermore, we assume
that the data set $P$ is a subset of some space $X$ equipped with a distance function $\text{dist}_X$, and that we are given another metric $\text{dist}_Y$ defined over $P \cup Q$. Given the graph $G$ and the queries $q_1, \cdots, q_k$, the goal of the SNN data structure is to return points $p_1, \cdots, p_k$ from $P$ that minimize the following function:

$$\sum_{i=1}^{k} \kappa_i \text{dist}_Y(p_i, q_i) + \sum_{(i,j) \in E} \lambda_{i,j} \text{dist}_X(p_i, p_j)$$

(5.1)

where $\kappa_i$ and $\lambda_{i,j}$ are parameters defined in advance.

The above formulation captures a wide variety of applications that are not well modeled by traditional NN search. For example, many applications in computer vision involve computing nearest neighbors of pixels or image patches from the same image [FJP02, BVZ01, BSFG09]. In particular, algorithms for tasks such as denoising (removing noise from an image), restoration (replacing a deleted or occluded part of an image) or super-resolution (enhancing the resolution of an image) involve assigning “labels” to each image patch\(^1\). The labels could correspond to the pixel color, the enhanced image patch, etc. The label assignment should have the property that the labels are similar to the image patches they are assigned to, while at the same time the labels assigned to nearby image patches should be similar to each other. The objective function in Equation 5.1 directly captures these constraints.

From a theoretical perspective, SNN generalizes several well-studied computational problems, notably the Aggregate Nearest Neighbor problem [YMP05, LLY+11, LYK11, AEZ12, KK12] and the 0-extension problem [Kar98, FHRT03, CKR05, AFH+04]. The first problem is quite similar to the basic nearest neighbor search problem over a metric $\text{dist}$, except that the data structure is given $k$ queries $q_1 \cdots q_k$, and the goal is to find a data point $p$ that minimizes the sum\(^2\) $\sum_i \text{dist}(q_i, p)$. This objective can be easily simulated in SNN by setting $\text{dist}_Y = \text{dist}$ and $\text{dist}_X = L \cdot \text{uniform}$, where $L$ is a very large number and $\text{uniform}(p,q)$ is the uniform metric. The 0-extension

\(^1\)This problem has been formalized in the algorithms literature as the metric labeling problem [KT02]. The problem considered in this chapter can thus be viewed as a variant of metric labeling with a very large number of labels.

\(^2\)Other aggregate functions, such as the maximum, are considered as well.
problem is a combinatorial optimization problem where the goal is to minimize an objective function quite similar to that in Equation 5.1. The exact definition of 0-extension as well as its connections to SNN are discussed in detail in Section 5.2.1.

5.1.1 Results

In this thesis, we consider the basic case where \( \text{dist}_X = \text{dist}_Z \) and \( \lambda_{i,j} = \kappa_i = 1 \); we refer to this variant as the unweighted case. Our main contribution is a general reduction that enables us to design and analyze efficient data structures for unweighted SNN. The algorithm (called Independent Nearest Neighbors or INN) consists of two steps. In the first (pruning) step, for each query point \( q_i \) we find its nearest neighbor\(^3\) point \( \hat{p}_i \); this can be done efficiently using existing nearest neighbor search data structures. In the second (optimization) step, we run an appropriate (approximation) algorithm for the SNN problem over sets \( q_1, \ldots, q_k \) and \( \hat{p}_1, \ldots, \hat{p}_k \); this can be done efficiently given that \( k \) is much smaller than \( n \). We show that the resulting algorithm satisfies a \( O(b \log k / \log \log k) \)-approximation guarantee, where \( b \) is the approximation factor of the algorithm used in the second step. This can be further improved to \( O(b\delta) \), if the metric space \( \text{dist} \) admits a \( \delta \)-padding decomposition (see Section 5.2 for more detail). The running time incurred by this algorithm is bounded by the cost of \( k \) nearest neighbor search queries in a data set of size \( n \) plus the cost of the approximation algorithm for the 0-extension problem over an input of size \( k \). By plugging in the best nearest neighbor algorithms for \( \text{dist} \) we obtain significant running time savings if \( k \ll n \).

We note that INN is somewhat similar to the belief propagation algorithm for super-resolution described in [FJP02]. Specifically, that algorithm selects 16 closest labels for each \( q_i \), and then chooses one of them by running a belief propagation algorithm that optimizes an objective function similar to Equation 5.1. However, we note that the algorithm in [FJP02] is heuristic and is not supported by approximation guarantees.

---

\(^3\)Our analysis immediately extends to the case where we compute approximate, not exact, nearest neighbors. For simplicity we focus only on the exact case in the following discussion.
We complement our upper bound by showing that the aforementioned reduction inherently yields super-constant approximation guarantee. Specifically, we show that, for an appropriate distance function \( \text{dist} \), queries \( q_1, \ldots, q_k \), and a label set \( P \), the best solution to SNN with the label set restricted to \( \hat{p}_1, \ldots, \hat{p}_k \) can be \( \Theta(\sqrt{\log k}) \) times larger than the best solution with label set equal to \( P \). This means that even if the second step of the problem is solved to optimality, reducing the set of labels from \( P \) to \( \hat{P} \) inherently increases the cost by a super-constant factor.

However, we further show that the aforementioned limitation can be overcome if the compatibility graph \( G \) has pseudoarboricity \( r \) (which means that each edge can be mapped to one of its endpoint vertices such that at most \( r \) edges are mapped to each vertex). Specifically, we show that if \( G \) has pseudoarboricity \( r \), then the gap between the best solution using labels in \( P \), and the best solution using labels in \( \hat{P} \), is at most \( O(r) \). Since many graphs used in practice do in fact satisfy \( r = O(1) \) (e.g., 2D grids, 3D grids or planar graphs), this means that the gap is indeed constant for a wide collection of common compatibility graphs.

Finally, we validate our theoretical results by preliminary experiments comparing our SNN data structure with an alternative (less efficient) algorithm that solves the same optimization problem using the full label set \( P \). In our experiments we apply both algorithms to an image denoising task and measure their performance using the objective function (5.1). In particular, we show that the “empirical gap” incurred by the above approach, i.e, the ratio of objective function values observed in our experiments, is very close to 1.

5.1.2 Techniques

We start by pointing out that SNN can be reduced to 0-extension in a “black-box” manner. Unfortunately, this reduction yields an SNN algorithm whose running time depends on the size of labels \( n \), which could be very large; essentially this approach defeats the goal of having a data structure solving the problem. The INN algorithm overcomes this issue by reducing the number of labels from \( n \) to \( k \). However, the pruning step can increase the cost of the best solution. The ratio between the optimum
cost after pruning to the optimum cost before pruning is called the pruning gap.

To bound the pruning gap, we again resort to existing $0$-extension algorithms, albeit in a “grey box” manner. Specifically, we observe that many algorithms, such as those in [CKR05, AFH+04, FHRT03, LN04], proceed by first creating a label assignment in an “extended” metric space (using a LP relaxation of $0$-extension), and then apply a rounding algorithm to find an actual solution. The key observation is that the correctness of the rounding step does not rely on the fact that the initial label assignment is optimal, but instead it works for any label assignment. We use this fact to translate the known upper bounds for the integrality gap of linear programming relaxations of $0$-extension into upper bounds for the pruning gap. On the flip side, we show a lower bound for the pruning gap by mimicking the arguments used in [CKR05] to lower bound the integrality gap of a $0$-extension relaxation.

To overcome the lower bound, we consider the case where the compatibility graph $G$ has pseudoarboricity $r$. Many graphs used in applications, such as 2D grids, 3D grids or planar graphs, have pseudoarboricity $r$ for some constant $r$. We show that for such graphs the pruning gap is only $O(r)$. The proof proceeds by directly assigning labels in $\hat{P}$ to the nodes in $Q$ and bounding the resulting cost increase. It is worth noting that the “grey box” approach outlined in the preceding paragraph, combined with Theorem 11 of [CKR05], yields an $O(r^3)$ pruning gap for the class of $K_{r,r}$-minor-free graphs, whose pseudoarboricity is $\tilde{O}(r)$. Our $O(r)$ pruning gap not only improves this $O(r^3)$ bound in a quantitative sense, but it also applies to a much broader class of graphs. For example, three-dimensional grid graphs have pseudoarboricity 6, but the class of three-dimensional grid graphs includes graphs with $K_{r,r}$ minors for every positive integer $r$.

Finally, we validate our theoretical results by experiments. We focus on a simple de-noising scenario where $X$ is the pixel color space, i.e., the discrete three-dimensional space space $\{0 \ldots 255\}^3$. Each pixel in this space is parametrized by the intensity of the red, green and blue colors. We use the Euclidean norm to measure the distance between two pixels. We also let $P = X$. We consider three test images: a cartoon with an MIT logo and two natural images. For each image we add some
noise and then solve the SNN problems for both the full color space $P$ and the pruned color space $\hat{P}$. Note that since $P = X$, the set of pruned labels $\hat{P}$ simply contains all pixels present in the image.

Unfortunately, we cannot solve the problems optimally, since the best known exact algorithm takes exponential time. Instead, we run the same approximation algorithm on both instances and compare the solutions. We find that the values of the objective function for the solutions obtained using pruned labels and the full label space are equal up to a small multiplicative factor. This suggests that the empirical value of the pruning gap is very small, at least for the simple data sets that we considered.

5.2 Definitions and Preliminaries

We define the Unweighted Simultaneous Nearest Neighbor problem as follows. Let $(X, \text{dist})$ be a metric space and let $P \subseteq X$ be a set of $n$ points from the space.

**Definition 5.2.1** (Unweighted SNN). In the Unweighted Simultaneous Nearest Neighbor problem, the goal is to build a data structure over a given point set $P$ that supports the following operation. Given an ordered set of $k$ points $Q = \{q_1, \cdots, q_k\}$ in the metric space $X$, along with a graph $G = (Q, E)$ of $k$ nodes, the goal is to report $k$ (not necessarily unique) points from the database $p_1, \cdots, p_k \in P$ which minimize the following cost function:

$$
\sum_{i=1}^{k} \text{dist}(p_i, q_i) + \sum_{(q_i, q_j) \in E} \text{dist}(p_i, p_j) \tag{5.2}
$$

We refer to the first term in sum as the nearest neighbor (NN) cost, and to the second sum as the pairwise (PW) cost. We denote the cost of the optimal assignment from the point set $P$ by $\text{cost}(Q, G, P)$.

In the rest of this paper, simultaneous nearest neighbor (SNN) refers to the unweighted version of the problem (unless stated otherwise). Next, we define the pseudoarboricity of a graph and $r$-sparse graphs.
Definition 5.2.2. Pseudoarboricity of a graph $G$ is defined to be the minimum number $r$, such that the edges of the graph can be oriented to form a directed graph with out-degree at most $r$. In this paper, we call such graphs as $r$-sparse.

Note that given an $r$-sparse graph, one can map the edges to one of its endpoint vertices such that there are at most $r$ edges mapped to each vertex.

The doubling dimension of a metric space is defined as follows.

Definition 5.2.3. The doubling dimension of a metric space $(X, \text{dist})$ is defined to be the smallest $\delta$ such that every ball in $X$ can be covered by $2^\delta$ balls of half the radius.

It is known that the doubling dimension of any finite metric space is $O(\log |X|)$. We then define padding decompositions.

Definition 5.2.4. A metric space $(X, \text{dist})$ is $\delta$-padded decomposable if for every $r$, there is a randomized partitioning of $X$ into clusters $C = \{C_i\}$ such that, each $C_i$ has diameter at most $r$, and that for every $x_1, x_2 \in X$, the probability that $x_1$ and $x_2$ are in different clusters is at most $\delta \text{dist}(x_1, x_2)/r$.

It is known that any finite metric with doubling dimension $\delta$ admits an $O(\delta)$-padding decomposition [GKL03].

5.2.1 0-Extension Problem

The 0-extension problem, first defined by Karzanov [Kar98] is closely related to the SNN problem. In the 0-extension problem, the input is a graph $G(V, E)$ with a weight function $w(e)$, and a set of terminals $T \subseteq V$ with a metric $d$ defined on $T$. The goal is to find a mapping from the vertices to the terminals $f : V \rightarrow T$ such that each terminal is mapped to itself and that the following cost function is minimized:

$$
\sum_{(u,v) \in E} w(u,v) \cdot d(f(u), f(v))
$$

It can be seen that this is a special case of the metric labeling problem [KT02] and thus a special case of the general version of the SNN problem defined by Equation
5.1. To see this, it is enough to let $Q = V$ and $P = T$, and let $\kappa_i = \infty$ for $q_i \in T$, $\kappa_i = 0$ for $q_i \not\in T$, and $\lambda_{i,j} = w(i,j)$ in Equation 5.1.

Calinescu et al. [CKR05] considered the semimetric relaxation of the LP for the 0-extension problem and gave an $O(\log |T|)$ algorithm using randomized rounding of the LP solution. They also proved an integrality ratio of $O(\sqrt{\log |T|})$ for the semimetric LP relaxation.

Later Fakcharoenphol et al. [FHRT03] improved the upper-bound to $O(\log |T|/\log \log |T|)$, and Lee and Naor [LN04] proved that if the metric $d$ admits a $\delta$-padded decomposition, then there is an $O(\delta)$-approximation algorithm for the 0-extension problem. For the finite metric spaces, this gives an $O(\delta)$ algorithm where $\delta$ is the doubling dimension of the metric space. Furthermore, the same results can be achieved using another metric relaxation (earth-mover relaxation), see [AFH+04]. Later Karloff et al. [KKMR09] proved that there is no polynomial time algorithm for 0-extension problem with approximation factor $O((\log n)^{1/4-\epsilon})$ unless $NP \subseteq DTIME(n^{poly(\log n)})$.

SNN can be reduced to 0-extension in a “black-box” manner via the following lemma.

**Lemma 5.2.5.** For $b \geq 1$, any $b$-approximate algorithm for the 0-extension problem yields an $O(b)$-approximate algorithm for the SNN problem.

**Proof.** Given an instance of the SNN problem $(Q, G', P)$, we build an instance of the 0-extension problem $(V, T, G)$ as follows. Let $T = P$ and $V = T \cup Q$. The metric $d$ is the same as dist. However, the graph $G$ of the 0-extension problem requires some modification. Let $G' = (Q, E_{G'})$, then $G = (V, E)$ is defined as follows. For each $q_i, q_j \in Q$, we have the edge $(q_i, q_j) \in E$ iff $(q_i, q_j) \in E_{G'}$. We also include another type of edges in the graph: for each $q_i \in Q$, we add an edge $(q_i, \hat{p}_i) \in E$ where $\hat{p}_i \in P$ is the nearest neighbor of $q_i$. Note that we consider the graph $G$ to be unweighted.

Using the $b$-approximation algorithm for this problem, we get an assignment $\mu$ that maps the non-terminal vertices $q_1, \ldots, q_k$ to the terminal vertices. Suppose $q_i$ is mapped to the terminal vertex $p_i$ in this assignment. Let $p_1^*, \ldots, p_k^*$ be the optimal
SNN assignment. Next, we show that the same mapping $\mu$ for the SNN problem, gives us an $O(b)$ approximate solution. The SNN cost of the mapping $\mu$ is denoted as follows:

$$
\text{cost}^{\text{SNN}}(\mu) = \sum_{i=1}^{k} \text{dist}(q_i, p_i) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i, p_j)
$$

$$
\leq \sum_{i=1}^{k} \text{dist}(q_i, \hat{p}_i) + \sum_{i=1}^{k} \text{dist}(\hat{p}_i, p_i) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i, p_j)
$$

$$
\leq \sum_{i=1}^{k} \text{dist}(q_i, p_i^*) + b \cdot \left[ \sum_{i=1}^{k} \text{dist}(\hat{p}_i, p_i^*) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i^*, p_j^*) \right]
$$

$$
\leq \text{cost}(Q, G', P) + b \cdot \left[ \sum_{i=1}^{k} \text{dist}(\hat{p}_i, q_i) + \sum_{i=1}^{k} \text{dist}(q_i, p_i^*) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i^*, p_j^*) \right]
$$

$$
\leq \text{cost}(Q, G', P) + b \cdot \left[ \sum_{i=1}^{k} \text{dist}(\hat{p}_i, q_i) + \text{cost}(Q, G', P) \right]
$$

$$
\leq \text{cost}(Q, G', P)(2b + 1)
$$

where we have used triangle inequality and the following facts in the above. First, $\hat{p}_i$ is the closest point in $P$ to $q_i$ and thus $\text{dist}(q_i, \hat{p}_i) \leq \text{dist}(q_i, p_i^*)$. Second, by definition we have that $\text{cost}(Q, G', P) = \sum_{i=1}^{k} \text{dist}(q_i, p_i^*) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i^*, p_j^*)$. Finally, since $\mu$ is a $b$ approximate solution for the 0-extension problem, we have that $\sum_{i=1}^{k} \text{dist}(\hat{p}_i, p_i) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i, p_j)$ is smaller than $b$ times the 0-extension cost of any other assignment, and in particular $\sum_{i=1}^{k} \text{dist}(\hat{p}_i, p_i^*) + \sum_{(q_i, q_j) \in E_{G'}} \text{dist}(p_i^*, p_j^*)$. $\square$

By plugging in the known 0-extension algorithms cited earlier we obtain the following:

**Corollary 5.2.6.** There exists an $O(\log n / \log \log n)$ approximation algorithm for the SNN problem with running time $n^{O(1)}$, where $n$ is the size of the label set.

**Corollary 5.2.7.** If the metric space $(X, \text{dist})$ is $\delta$-padded decomposable, then there exists an $O(\delta)$ approximation algorithm for the SNN problem with running time $n^{O(1)}$. For finite metric spaces $X$, $\delta$ could represent the doubling dimension of the metric space (or equivalently the doubling dimension of $P \cup Q$).
Unfortunately, this reduction yields a SNN algorithm with running time depending on the size of labels $n$, which could be very large. In the next section we show how to improve the running time by reducing the label set size from $n$ to $k$. However, unlike the reduction in this section, our new reduction will no longer be “black-box”. Instead, its analysis will use particular properties of the 0-extension algorithms. Fortunately, those properties are satisfied by the known approximation algorithms for this problem.

### 5.3 Independent Nearest Neighbors Algorithm

In this section, we consider a natural and general algorithm for the SNN problem, which we call Independent Nearest Neighbors (INN). The algorithm proceeds as follows. Given the query points $Q = \{q_1, \cdots, q_k\}$, for each $q_i$ the algorithm picks its (approximate) nearest neighbor $\hat{p}_i$. Then it solves the problem over the set $\hat{P} = \{\hat{p}_1, \cdots, \hat{p}_k\}$ instead of $P$. This simple approach reduces the size of search space from $n$ down to $k$.

The details of the algorithm are shown in Algorithm 11.

<table>
<thead>
<tr>
<th>Algorithm 11: Independent Nearest Neighbors (INN) Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong> $Q = {q_1, \cdots, q_k}$, and input graph $G = (Q, E)$</td>
</tr>
<tr>
<td>1: for $i = 1$ to $k$ do</td>
</tr>
<tr>
<td>2: Query the NN data structure to extract a nearest neighbor (or approximate nearest neighbor) $\hat{p}_i$ for $q_i$</td>
</tr>
<tr>
<td>3: end for</td>
</tr>
<tr>
<td>4: Find the optimal (or approximately optimal) solution among the set $\hat{P} = {\hat{p}_1, \cdots, \hat{p}_k}$.</td>
</tr>
</tbody>
</table>

In the rest of the section we analyze the quality of this pruning step. More specifically, we define the pruning gap of the algorithm as the ratio of the optimal cost function using the points in $\hat{P}$ over its value using the original point set $P$.

**Definition 5.3.1.** The pruning gap of an instance of SNN is defined as $\alpha(Q, G, P) = $
We define the pruning gap of the INN algorithm, $\alpha$, as the largest value of $\frac{\text{cost}(Q,G,\hat{P})}{\text{cost}(Q,G,P)}$ over all instances.

First, in Section 5.3.1, by proving a reduction from algorithms for rounding the LP solution of the 0-extension problem, we show that for arbitrary graphs $G$, we have $\alpha = O(\log k / \log \log k)$, and if the metric $(X, \text{dist})$ is $\delta$-padded decomposable, we have $\alpha = O(\delta)$ (for example, for finite metric spaces $X$, $\delta$ can represent the doubling dimension of the metric space). Then, in Section 5.3.2, we prove that $\alpha = O(r)$ where $r$ is the pseudoarboricity of the graph $G$. This would show that for the sparse graphs, the pruning gap remains constant. Finally, in Section 5.4, we present a lower bound showing that the pruning gap could be as large as $\Omega(\sqrt{\log k})$ and as large as $\Omega(r)$ for $r \leq \sqrt{\log k}$. Therefore, we get the following theorem.

**Theorem 5.3.2.** The following bounds hold for the pruning gap of the INN algorithm. First we have $\alpha = O(\frac{\log k}{\log \log k})$, and that if metric $(X, \text{dist})$ is $\delta$-padded decomposable, we have $\alpha = O(\delta)$. Second, $\alpha = O(r)$ where $r$ is the pseudoarboricity of the graph $G$. Finally, we have that $\alpha = \Omega(\sqrt{\log k})$ and $\alpha = \Omega(r)$ for $r \leq \sqrt{\log k}$.

Note that the above theorem results in an $O(b \cdot \alpha)$ time algorithm for the SNN problem where $b$ is the approximation factor of the algorithm used to solve the metric labeling problem for the set $\hat{P}$, as noted in line 4 of the INN algorithm. For example in a general graph $b$ would be $O(\log k / \log \log k)$ that is added on top of $O(\alpha)$ approximation of the pruning step.

### 5.3.1 Bounding the Pruning Gap Using 0-extension

In this section we show upper bounds for the pruning gap ($\alpha$) of the INN algorithm. The proofs use specific properties of existing algorithms for the 0-extension problem.

**Definition 5.3.3.** We say an algorithm $A$ for the 0-extension problem is a $\beta$-natural rounding algorithm if, given a graph $G = (V,E)$, a set of terminals $T \subseteq V$, a metric space $(X,d_X)$, and a mapping $\mu : V \rightarrow X$, it outputs another mapping $\nu : V \rightarrow X$ with the following properties:
\begin{itemize}
  \item ∀t ∈ T : ν(t) = µ(t)
  \item ∀v ∈ V : \exists t ∈ T \text{ s.t. } ν(v) = µ(t)
  \item cost(ν) ≤ βcost(µ), \text{ i.e., } \sum_{(u,v) ∈ E} d_X(ν(u), ν(v)) ≤ β \cdot \sum_{(u,v) ∈ E} d_X(µ(u), µ(v))
\end{itemize}

Many previous algorithms for the 0-extension problem, such as [CKR05, AFH+04, FHRT03, LN04], first create the mapping µ using some LP relaxation of 0-extension (such as semimetric relaxation or earth-mover relaxation), and then apply a β-natural rounding algorithm for the 0-extension to find the mapping ν which yields the solution to the 0-extension problem. Below we give a formal connection between guarantees of these rounding algorithms, and the quality of the output of the INN algorithm (the pruning gap of INN).

**Lemma 5.3.4.** Let A be a β-natural rounding algorithm for the 0-extension problem. Then we can infer that the pruning gap of the INN algorithm is \(O(β)\), that is, \(α = O(β)\).

**Proof.** Fix any SNN instance \((Q, G_S, P)\), where \(G_S = (Q, E_{PW})\), and its corresponding INN invocation.

We construct the inputs to the algorithm A from the INN instance as follows. Let the metric space of A be the same as \((X, \text{dist})\) defined in the SNN instance. Also, let V be a set of 2k vertices corresponding to \(\hat{P} ∪ P^*\) with T corresponding to \(\hat{P}\). Here \(P^* = \{p^*_1, \cdots, p^*_k\}\) is the set of the optimal solutions of SNN, and \(\hat{P}\) is the set of nearest neighbors as defined by INN. The mapping \(µ\) simply maps each vertex from \(V = \hat{P} ∪ P^*\) to itself in the metric \(X\) defined in SNN. Moreover, the graph \(G = (V, E)\) is defined such that \(E = \{(\hat{p}_i, p^*_i) | 1 \leq i \leq k\} \cup \{(p^*_i, p^*_j) | (q_i, q_j) ∈ E_{PW}\}\).

First we claim the following (note that \(cost(µ)\) is defined in Definition 5.3.3, and that by definition \(cost(Q, G_S, P) = cost(Q, G_S, P^*)\))

\[
  cost(µ) ≤ 2cost(Q, G_S, P^*) = 2cost(Q, G_S, P)
\]

We know that \(cost(Q, G_S, P^*)\) can be split into NN cost and PW cost. We can also split \(cost(µ)\) into NN cost (corresponding to edge set \(\{(\hat{p}_i, p^*_i) | 1 \leq i \leq k\}\)) and PW
cost (corresponding to edge set \( \{(p_i^*, p_j^*) | (q_i, q_j) \in E_{PW}\} \)). By definition we know the PW costs of \( \text{cost}(Q, G_S, P) \) and \( \text{cost}(\mu) \) are equal. For NN cost, by triangle inequality, we know \( \text{dist}(\hat{p}_i, p_i^*) \leq \text{dist}(\hat{p}_i, q_i) + \text{dist}(q_i, p_i^*) \leq 2 \cdot \text{dist}(q_i, p_i^*) \). Here we use the fact that \( \hat{p}_i \) is the nearest database point of \( q_i \). Thus, the claim follows.

We then apply algorithm A to get the mapping \( \nu \). By the assumption on A, we know that \( \text{cost}(\nu) \leq \beta \text{cost}(\mu) \). Given the mapping \( \nu \) by the algorithm A, consider the assignment in the SNN instance where each query \( q_i \) is mapped to \( \nu(p_i^*) \), and note that since \( \nu(p_i^*) \in T \), this would map all points \( q_i \) to points in \( \hat{P} \). Thus, by definition, we have that

\[
\text{cost}(Q, G_S, \hat{P}) \leq \sum_{i=1}^{k} \text{dist}(q_i, \nu(p_i^*)) + \sum_{(q_i, q_j) \in E_{PW}} \text{dist}(\nu(p_i^*), \nu(p_j^*)) \\
\leq \sum_{i=1}^{k} \text{dist}(q_i, \hat{p}_i) + \sum_{i=1}^{k} \text{dist}(\hat{p}_i, \nu(p_i^*)) + \sum_{(q_i, q_j) \in E_{PW}} \text{dist}(\nu(p_i^*), \nu(p_j^*)) \\
\leq \sum_{i=1}^{k} \text{dist}(q_i, \hat{p}_i) + \text{cost}(\nu) \\
\leq \text{cost}(Q, G_S, P) + \beta \text{cost}(\mu) \\
\leq (2\beta + 1)\text{cost}(Q, G_S, P)
\]

where we have used the triangle inequality. Therefore, we have that the pruning gap \( \alpha \) of the INN algorithm is \( O(\beta) \), as claimed.

Using the previously cited results, and noting that in the above instance \(|V| = O(k)\), we get the following corollaries.

**Corollary 5.3.5.** The INN algorithm has pruning gap \( \alpha = O(\log k / \log \log k) \).

**Corollary 5.3.6.** If the metric space \((X, \text{dist})\) admits a \( \delta \)-padding decomposition, then the INN algorithm has pruning gap \( \alpha = O(\delta) \). For finite metric spaces \((X, \text{dist})\), \( \delta \) is at most the doubling dimension of the metric space.
5.3.2 Sparse Graphs

In this section, we prove that the INN algorithm performs well on sparse graphs. More specifically, here we prove that when the graph $G$ is $r$-sparse, then $\alpha(Q,G,P) = O(r)$. To this end, we show that there exists an assignment using the points in $\hat{P}$ whose cost function is within $O(r)$ of the optimal solution using the points in the original data set $P$.

Given a graph $G$ of pseudoarboricity $r$, we know that we can map each edge to one of its end points such that the number of edges mapped to each vertex is at most $r$. For each edge $e$, we call the vertex that $e$ is mapped to as the corresponding vertex of $e$. This would mean that each vertex is the corresponding vertex of at most $r$ edges.

Let $p_1^*, \ldots, p_k^* \in P$ denote the optimal solution of SNN. Algorithm 12 shows how to find an assignment $p_1, \ldots, p_k \in \hat{P}$. We show that the cost of this assignment is within a factor $O(r)$ from the optimum.

Algorithm 12: $r$-Sparse Graph Assignment Algorithm

| Input | Query points $q_1, \ldots, q_k$, Optimal assignment $p_1^*, \ldots, p_k^*$, Nearest Neighbors $\hat{p}_1, \ldots, \hat{p}_k$, and the input graph $G = (Q,E)$ |
| Output | An Assignment $p_1, \ldots, p_k \in \hat{P}$ |

1: for $i = 1$ to $k$ do
2: Let $j_0 = i$ and let $q_{j_1}, \ldots, q_{j_t}$ be all the neighbors of $q_i$ in the graph $G$
3: $m \leftarrow \arg \min_{t=0}^{t} \ell \text{dist}(p_i^*, p_{j_t}^*) + \text{dist}(p_{j_t}^*, q_{j_t})$
4: Assign $p_i \leftarrow \hat{p}_{j_m}$
5: end for

Lemma 5.3.7. The assignment defined by Algorithm 12, has $O(r)$ approximation factor.

Proof. For each $q_i \in Q$, let $y_i = \text{dist}(p_i^*, q_i)$ and for each edge $e = (q_i, q_j) \in E$ let $x_e = \text{dist}(p_i^*, p_j^*)$. Also let $Y = \sum_{i=1}^{k} y_i$ and $X = \sum_{e \in E} x_e$. Note that $Y$ is the NN cost and $X$ is the PW cost of the optimal assignment and that $OPT = \text{cost}(Q,G,P) = \sum_{i=1}^{k} y_i$.
Define the variables $y_i', x_e', Y', X'$ in the same way but for the assignment $p_1, \cdots, p_k$ produced by the algorithm. That is, for each $q_i \in Q$, $y_i' = \text{dist}(p_i, q_i)$, and for each edge $e = (q_i, q_j) \in E$, $x_e' = \text{dist}(p_i, p_j)$. Moreover, for a vertex $q_i$, we define the designated neighbor of $q_i$ to be $q_{im}$ for the value of $m$ defined in the line 3 of Algorithm 12 (note that the designated neighbor might be the vertex itself). Fix a vertex $q_i$ and let $q_c$ be the designated neighbor of $q_i$. We can bound the value of $y_i'$ as follows.

$$y_i' = \text{dist}(q_i, p_i) = \text{dist}(q_i, \hat{p}_c)$$

$$\leq \text{dist}(q_i, p_i^*) + \text{dist}(p_i^*, p_c^*) + \text{dist}(p_c^*, q_c) + \text{dist}(q_c, \hat{p}_c) \quad \text{(by triangle inequality)}$$

$$\leq y_i + \text{dist}(p_i^*, p_c^*) + 2\text{dist}(p_c^*, q_c) \quad \text{(since $\hat{p}_c$ is the nearest neighbor of $q_c$)}$$

$$\leq y_i + 2[\text{dist}(p_i^*, p_c^*) + \text{dist}(p_c^*, q_c)]$$

$$\leq 3y_i$$

(by definition of designated neighbor and the value $m$ in line 3 of Algorithm 12)

Thus summing over all vertices, we get that $Y' \leq 3Y$. Now for any fixed edge $e = (q_i, q_s)$ (with $q_i$ being its corresponding vertex), let $q_c$ be the designated neighbor of $q_i$, and $q_z$ be the designated neighbor of $q_s$. Then we bound the value of $x_e'$ as:

$$x_e' = \text{dist}(p_i, p_s) = \text{dist}(\hat{p}_e, \hat{p}_z)$$

(by definition of designated neighbor and line 4 of Algorithm 12)

$$\leq \text{dist}(\hat{p}_e, q_c) + \text{dist}(q_c, p_c^*) + \text{dist}(p_c^*, p_i^*) + \text{dist}(p_i^*, p_s^*)$$

$$+ \text{dist}(p_s^*, p_z^*) + \text{dist}(p_z^*, q_z) + \text{dist}(q_z, \hat{p}_z) \quad \text{(by triangle inequality)}$$

$$\leq 2\text{dist}(q_c, p_c^*) + \text{dist}(p_c^*, p_i^*) + \text{dist}(p_i^*, p_s^*) + \text{dist}(p_s^*, p_z^*)$$

$$+ 2\text{dist}(p_z^*, q_z) \quad \text{(since $\hat{p}_e(\hat{p}_z$ respectively) is a NN of $q_c(q_z$ respectively))}$$

$$\leq 2[\text{dist}(q_c, p_c^*) + \text{dist}(p_c^*, p_i^*)] + \text{dist}(p_i^*, p_s^*) + 2[\text{dist}(p_s^*, p_z^*) + \text{dist}(p_z^*, q_z)]$$

$$\leq 2y_i + x_e + 2[x_e + y_i]$$

(since $q_c(q_z$ respectively) is designated neighbor of $q_i(q_s$ respectively))

$$\leq 4(x_e + y_i)$$
Hence, summing over all the edges, since each vertex $q_i$ is the corresponding vertex of at most $r$ edges, we get that $X' \leq 4X + 4rY$. Therefore we have the following.

$$\text{cost}(Q, G, \hat{P}) \leq X' + Y' \leq 3Y + 4X + 4rY \leq (4r + 3) \cdot \text{cost}(Q, G, P)$$

and thus $\alpha(Q, G, P) = O(r)$. 

\[\square\]

### 5.4 Lower Bound

In this section we prove a lower bound of $\Omega(\sqrt{\log k})$ for the approximation factor of the INN algorithm. Furthermore, the lower bound example presented in this section is a graph (in fact a multi-graph) that has pseudoarboricity equal to $O(\sqrt{\log k})$, showing that in a way, the upper bound of $\alpha = O(r)$ for the $r$-sparse graphs is tight. More specifically, we show that for $r \leq \sqrt{\log k}$, we have $\alpha = \Omega(r)$. We note that the lower bound construction presented in this paper is similar to the approach of [CKR05] for proving a lower bound for the integrality ratio of the LP relaxation for the 0-extension problem.

**Lemma 5.4.1.** For any value of $k$, there exists a set of points $P$ of size $O(k)$ in a metric space $X$, and a query $(Q, G)$ such that $|Q| = k$ and the pruning step induces an approximation factor of at least $\alpha(Q, G, P) = \Omega(\sqrt{\log k})$.

**Proof.** In what follows, we describe the construction of the lower bound example.

Let $H = (V, E)$ be an expander graph with $k$ vertices $V = \{v_1, \cdots, v_k\}$ such that each vertex has constant degree $d$ and the vertex expansion of the graph is also a constant $c$. Let $H' = (V', E', W')$ be a weighted graph constructed from $H$ by adding $k$ vertices $\{u_1, \cdots, u_k\}$ such that each new vertex $u_i$ is a leaf connected to $v_i$ with an edge of weight $\sqrt{\log k}$. All the other edges between $\{v_1, \cdots, v_k\}$ (which were present in $H$) have weight 1. This graph $H'$ defines the metric space $(X, \text{dist})$ such that $X$ is the set of nodes $V'$ and dist is the weight of the shortest path between the nodes in the graph $H'$. Moreover, let $P = V'$ be all the vertices in the graph $H'$.
Let the set of $k$ queries be $Q = V' \setminus V = \{u_1, \ldots, u_k\}$. Then, while running the INN algorithm, the set of candidates $\hat{P}$ would be the queries themselves, i.e., $\hat{P} = Q = \{u_1, \ldots, u_k\}$. Also, let the input graph $G = (Q, E_G)$ be a multi-graph which is obtained from $H$ by replacing each edge $(v_i, v_j)$ in $H$ with $\sqrt{\log k}$ copies of the edge $(u_i, u_j)$ in $G$. This is the input graph given along with the $k$ queries to the algorithm.

Consider the solution $P^* = \{p^*_1, \ldots, p^*_k\}$ where $p^*_i = v_i$. The cost of this solution is

$$\sum_{i=1}^{k} \text{dist}(q_i, p^*_i) + \sum_{(u_i, u_j) \in E_G} \text{dist}(v_i, v_j) = k\sqrt{\log k} + kd\sqrt{\log k}/2$$

Therefore, the cost of the optimal solution $OPT = \text{cost}(Q, G, P)$ is at most $O(k\sqrt{\log k})$. Next, consider the optimal labeling $\hat{P}^* = \{\hat{p}^*_1, \ldots, \hat{p}^*_k\} \subseteq \hat{P}$ using only the points in $\hat{P}$. This optimal assignment has one of the following forms.

**Case 1:** For all $1 \leq i \leq k$, we have $\hat{p}^*_i = u_i$. The cost of $\hat{P}^*$ in this case would be

$$\text{cost}(Q, G, \hat{P}) = \sum_{i=1}^{k} \text{dist}(q_i, u_i) + \sum_{(u_i, u_j) \in E_G} \text{dist}(u_i, u_j) \geq 0 + |E_G| \cdot 2\sqrt{\log k} \geq \frac{dk}{2} \log k$$

Thus the cost in this case would be $\Omega(OPT\sqrt{\log k})$.

**Case 2:** All the $\hat{p}^*_i$'s are equal. Without loss of generality suppose they are all equal to $u_1$. Then the cost would be:

$$\text{cost}(Q, G, \hat{P}) = \sum_{i=1}^{k} \text{dist}(q_i, u_1) + \sum_{(u_i, u_j) \in E_G} \text{dist}(u_1, u_1) \geq \Omega(k \log k) + 0$$

This is true because in an expander graph with constant degree, the number of vertices at distance less than $\frac{\log \frac{k}{2}}{d} \log d$ of any vertex is at most $1 + d + \cdots, d^{\log \frac{k}{2} \leq 2\sqrt{k}}$. Thus $\Theta(k)$ vertices are farther than $\frac{\log \frac{k}{2}}{d} = \frac{\log k}{2 \log d} = \Theta(\log k)$. Thus, again the cost of the assignment $\hat{P}$ in this case would be $\Omega(OPT\sqrt{\log k})$.

**Case 3:** Let $S = \{S_1, \ldots, S_t\}$ be a partition of $[k]$ such that each part corresponds to all the indices $i$ having their $\hat{p}^*_i$ equal. That is, for each $1 \leq j \leq t$, we have $\forall i, i' \in S_j : \hat{p}^*_i = \hat{p}^*_i'$. Now, two cases are possible. First if all the parts $S_j$ have size at
most \( k/2 \). In this case, since the graph \( H \) has expansion \( c \), the total number of edges between different parts would be at least

\[
\left| \{(u_i, u_j) \in E_G | \hat{p}_i^2 \neq \hat{p}_j^2 \} \right| \geq \frac{1}{2} \sum_{j=1}^{t} c |S_j| \sqrt{\log k} \geq kc\sqrt{\log k}/2
\]

Therefore similar to Case 1 above, the PW cost would be at least \( kc\sqrt{\log k}/2 \cdot \sqrt{\log k} = \Omega(k \log k) \). Otherwise, at least one of the parts such as \( S_j \) has size at least \( k/2 \). In this case, similar to Case 2 above, the NN cost would be at least \( \Omega(k \log k) \). Therefore, in both cases the cost of the assignment \( \hat{P}^* \) would be at least \( \Omega(OPT \sqrt{\log k}) \). Hence, the pruning gap of the \textsc{INN} algorithm on this graph is \( \Omega(\sqrt{\log k}) \).

Since the degree of all the vertices in the above graph is \( d\sqrt{\log k} \), the pseudoarboricity of the graph is also \( \Theta(\sqrt{\log k}) \). It is easy to check that if we repeat each edge \( r \) times instead of \( \sqrt{\log k} \) times in \( E_G \) in the above proof, the same arguments hold and we get the following corollary.

**Corollary 5.4.2.** For any value of \( r \leq \sqrt{\log k} \), there exists an instance of \textsc{SNN}(\( Q,G,P \)) such that the input graph \( G \) has arboricity \( O(r) \) and that the pruning gap of the \textsc{INN} algorithm is \( \alpha(Q,G,P) = \Omega(r) \).

### 5.5 Experiments

We consider image denoising as an application of our algorithm. A popular approach to denoising (see e.g. [FPTZ10]) is to minimize the following objective function:

\[
\sum_{i \in V} \kappa_i d(q_i, p_i) + \sum_{(i,j) \in E} \lambda_{i,j} d(p_i, p_j)
\]

Here \( q_i \) is the color of pixel \( i \) in the noisy image, and \( p_i \) is the color of pixel \( i \) in the output. We use the standard 4-connected neighborhood system for the edge set \( E \), and use Euclidean distance as the distance function \( d(\cdot, \cdot) \). We also set all weights \( \kappa_i \) and \( \lambda_{i,j} \) to 1.
When the image is in grey scale, this objective function can be optimized approximately and efficiently using message passing algorithm, see e.g. [FH06]. However, when the image pixels are points in RGB color space, the label set becomes huge \((n = 256^3 = 16,777,216)\), and most techniques for metric labeling are not feasible.

Recall that our algorithm proceeds by considering only the nearest neighbor labels of the query points, i.e., only the colors that appeared in the image. In what follows we refer to this reduced set of labels as the \textit{image color} space, as opposed to the \textit{full color} space where no pruning is performed.

In order to optimize the objective function efficiently, we use the technique of [FPTZ10]. We first embed the original (color) metric space into a tree metric (with \(O(\log n)\) distortion), and then apply a top-down divide and conquer algorithm on the tree metric, by calling the alpha-beta swap subroutine [BK04]. We use the random-split kd-tree for both the full color space and the image color space. When constructing the kd-tree, split each interval \([a, b]\) by selecting a random number chosen uniformly at random from the interval \([0.6a + 0.4b, 0.4a + 0.6b]\).

To evaluate the performance of the two algorithms, we use one cartoon image with MIT logo and two images from the Berkeley segmentation dataset [MFM04] which was previously used in other computer vision papers [FPTZ10]. We use Matlab imnoise function to create noisy images from the original images. We run each instance 20 times, and compute both the average and the variance of the objective function (the variance is due to the random generating process of kd tree).

The results are presented in Figure 5-1 and Table 5.1. In Figure 5-1, one can see that the images produced by the two algorithms are comparable. The full color version seems to preserve a few more details than the image color version, but it

<table>
<thead>
<tr>
<th></th>
<th>Avg cost for full color</th>
<th>Avg cost for image color</th>
<th>Empirical pruning gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT</td>
<td>341878 ± 3.1%</td>
<td>340477 ± 1.1%</td>
<td>0.996</td>
</tr>
<tr>
<td>Snow</td>
<td>9338604 ± 4.5%</td>
<td>9564288 ± 6.2%</td>
<td>1.024</td>
</tr>
<tr>
<td>Surf</td>
<td>8304184 ± 6.6%</td>
<td>7588244 ± 5.1%</td>
<td>0.914</td>
</tr>
</tbody>
</table>

\textbf{Table 5.1:} The empirical values of objective functions for the respective images and algorithms.
Figure 5-1: MIT logo (first column, size 45 × 124), and two images from the Berkeley segmentation dataset [MFM04] (second & third columns, size 321 × 481). The first row shows the original image; the second row shows the noisy image; the third row shows the denoised image using full color space; the fourth row shows the denoised image using image space (our algorithm).

also “hallucinates” non-existing colors to minimize the value of the objective function. The visual quality of the de-noised images can be improved by fine-tuning various parameters of the algorithms. We do not report these results here, as our goal was to compare the values of the objective function produced by the two algorithms, as opposed to developing the state of the art de-noising system.

Note that, as per Table 5.1, for some images the value of the objective function is sometimes lower for the image color space compared to the full color space. This is because we cannot solve the optimization problem exactly. In particular, using the kd tree to embed the original metric space into a tree metric is an approximate process.
5.5.1 De-noising with Patches

To improve the quality of the de-noised images, we run the experiment for patches of the image, instead of pixels. Moreover, we use an improved algorithm \(^4\) which implements not only a pruning step, but also computes the solution directly and achieves approximation factor \(2r + 1\). In this experiment (see Figure 5-2 for a sample of the results), each patch (a grid of pixels) from the noisy image is a query point, and the dataset consists of available patches which we use as a substitute for a noisy patch.

In our experiment, to build the dataset, we take one image from the Berkeley segmentation data set, then add noise to the right half of the image, and try to use the patches from the left half to denoise the right half. Each patch is of size \(5 \times 5\) pixels. We obtain \(317 \times 236\) patches from the left half of the image and use it as the patch database. Then we apply our algorithm to denoise the image. In particular, for each noisy patch \(q_n\) (out of \(317 \times 237\) patches) in the right half of the image, we perform a linear scan to find the closest patch \(p_i\) from the patch database, based on the following cost function:

\[
\text{dist}(q_n, p_i) + \sum_{p_j \in \text{neighbor}(q_n)} \frac{\text{dist}(p_j, p_i)}{5}
\]

where \(\text{dist}(p, q)\) is defined to be the sum of squares of the \(l_2\) distances between the colors of corresponding pixels in the two patches.

After that, for each noisy patch we retrieve the closest patch from the patch database. Then for each noisy pixel \(x\), we first identify all the noisy patches (there are at most 25 of them) that cover it. The denoised color of this pixel \(x\) is simply the average of all the corresponding pixels in those noisy patches which cover \(x\).

Since the nearest neighbor algorithm is implemented using a linear scan, it takes around 1 hour to denoise one image. One could also apply some more advanced techniques like locality sensitive hashing to find the closest patches with much faster

\(^4\)See [IKMY16] for details of the algorithm and its proof of correctness.
Figure 5-2: Two images from the Berkeley segmentation dataset [MFM04] (size 321 * 481). The first column shows the original image; the second column shows the half noisy image; the third column shows the de-noised image using our algorithm for the patches.
Part II

Composable Coresets
Chapter 6

Composable Coresets

6.1 Introduction and Background

In this chapter, we introduce the notion of composable coresets which is a small but efficient representation of the data, and then we show how to compute such representation, for the specific purpose of diversity-aware summarization and search.

**Definition 6.1.1 (Composable Coreset).** Given a universal set $\mathbb{U}$, and a maximization set function $f : 2^\mathbb{U} \rightarrow \mathbb{R}$ that is defined for any subset of $\mathbb{U}$, an $\alpha$-approximate composable coreset, is a mapping $\mu$ that maps any data set $S \subseteq \mathbb{U}$ to a subset of $S$ with the following composability property: for a collection of (possibly overlapping) data sets $S_1, \ldots, S_m \subseteq \mathbb{U}$, the value of the function over the union of the composable coresets, approximates the value of the function over the union of the data sets:

$$\frac{1}{\alpha} \cdot f(\bigcup_{i=1}^{m} S_i) \leq f(\bigcup_{i=1}^{m} \mu(S_i)) \leq f(\bigcup_{i=1}^{m} S_i)$$

That is, one can construct an approximately optimal solution for a given data set by partitioning it into several (possibly overlapping) blocks, computing a coreset for each block, and then solving the problem for the union of the coresets. Composable coresets naturally lead to divide-and-conquer solutions to a collection of massive data processing problems. In particular, they have been used for the following tasks:
• Streaming computation: In the data stream model, a sequence of \( n \) data elements needs to be processed “on-the-fly” while using only limited storage. Such an algorithm can be easily obtained using composable coresets [GM16, AHV04]. Specifically, if a composable coreset for a given problem has size \( k \), we start by dividing the stream of data into \( \sqrt{n/k} \) blocks of size \( s = \sqrt{n/k} \). The algorithm then proceeds block by block. Each block is read and stored in the main memory, its coreset is computed and stored, and the block is deleted. At the end, the algorithm solves the problem for the union of the coresets. The whole algorithm takes only \( O(\sqrt{kn}) \) space. The storage can be reduced further by utilizing more than one level of compression, at the cost of increasing the approximation factor.

• Distributed data processing: composable coresets can be also used to process data in a distributed system, where each machine holds a block of the data. The algorithm is virtually identical to the one for streaming data: for each block, a composable coreset is computed and sent to the central server, where the computation is completed. As an example, this idea is directly applicable in the map-reduce framework [DG08] and gives an approximation algorithm in one round of map-reduce: using \( \sqrt{n/k} \) mappers, each mapper gets \( \sqrt{kn} \) points as input and computes a composable coreset of size \( k \) for this set. These sets will be passed to a single reducer. The input of this reducer is the union of the coresets, which is of size at most \( k\sqrt{n/k} = \sqrt{kn} \). It computes and outputs a solution on this union, which by the definition of coresets is a good approximation to the original problem. Recently, variants of this technique have been applied for optimization under map-reduce framework [KSV10, LMSV11, BEL13].

• Similarity search: recently, composable coresets have been used to construct efficient near neighbor search algorithms that maximize the diversity of the answers, both in theory [AAYT+1] and in practice [AAYIM13]. This is done by observing that several similarity search algorithms (notably those based on

\[1\]The paper [GM16] introduced this approach for the special case of \( k \)-median clustering. More general formulation of this method with other applications appeared in [AHV04].
the Locality-Sensitive Hashing technique) proceed by hashing each point into multiple buckets. Each query is then answered by retrieving the points stored in the buckets that the query is mapped into. Since the number of points stored in a bucket might be large (which is the case, e.g., when the data set contains one big cluster of close points), the query answering procedure might be slow. To improve the performance, the paper [AYIM13] proposed to replace the content of each bucket by its coreset. By collecting the coresets stored in all relevant buckets and performing the computation over the their union, the algorithm reports a diverse summary of the points close to the query in time that depends on the number of buckets, not the size of the whole data set. This is discussed in more details in Section 6.5.

The broad applicability of composable coresets motivates the study of efficient methods for constructing them. In particular in this chapter, we consider designing such composable coresets for the diversity maximization problem: we would like to design an algorithm that maps a set $S$ to one of its subsets such that for a collection of sets, the maximum diversity of the union of those sets is within an $\alpha$ factor of the maximum diversity of the union of the corresponding coresets.

6.1.1 Results

We present a thorough study of composable coresets for several well-studied diversity maximization problems. Suppose that the set $S$ of interest lives in some metric space $(U, \text{dist})$, and let $\text{div}(S)$ be any function that maps a set into a non-negative real number. The goal of the diversity maximization problem is to find a subset $S'$ of $S$ of size $k$ that maximizes the diversity objective.

The specific diversity functions $\text{div}$ considered in this work are described in Table 6.1, following the taxonomy of dispersion measures introduced in [CH01]. For each dispersion function we provide the approximation factor of the composable coreset that we obtain for that function. We note that for all coresets the approximation
factor matches that of the best “offline” algorithm for the corresponding diversity maximization problem [CH01] (up to a constant factor). All coresets are of size $k$.

The interpretation of the diversity measures is as follows. First, remote-edge [AAYIM13] and remote-clique [GS09] correspond to the well-studied diversity notions where the objective is to ensure that no two pairs of points are too “close” to each other, or that an average pair distance of points is not too “low”, respectively. Remote-pseudoforest falls in between the two notions, as its goal is to ensure that the average distance of a point to its nearest neighbor is not too “low”. Remote-pseudoforest can be viewed as a diversity analog of the well-studied Chamfer distance [JBS06]. Remote-tree and remote $k$-tree measure the diversity by the cost of clustering the data using the Single Link algorithm [Sib73]. Similarly, remote-star measures the diversity by the cost of connecting the points to the best center$^2$. Finally, remote-matching, remote-cycle and remote-bipartition are more “exotic” combinatorial variations of the aforementioned measures. We include them to complete the table of [CH01].

All aforementioned notions of diversity are “pairwise”, i.e., they are a function of the pair-wise distances between the selected items. We also consider a basic “higher order” notion of diversity which has been previously discussed in the context of diversity maximization [AMT13, BJLY17]. Intuitively, the idea is to model diversity by considering a set of topics that each item covers, and exploring the diversity or the union of topics covered by a set of items. More specifically, we consider the scenario where the items are binary vectors of topics, and the diversity of a set of items is equal to their coverage over another set of topics, i.e., the weight of the coordinate-wise OR of the item vectors. As before, the goal is to choose a set of size $k$ which maximizes the total coverage. This is directly related to the maximum $k$-coverage problem that admits a tight $1 - 1/e$-approximation algorithm [Fei98]. We show in Section 6.4 that this problem does not support composable coresets of size polynomial in $k$. In particular, for any $\alpha \leq \frac{\sqrt{k}}{\log k}$ and any constant $\beta > 0$, there exists a set of instances for which no $\alpha$-approximate coreset of size $k^\beta$ exists. As an illustrative

$^2$Note that the values of remote-clique and remote-star objectives are within a factor of $\Theta(k)$ from each other, and thus the coresets for the two objectives are equivalent up to constant factors.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Diversity of the point set $S$</th>
<th>Approx. factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote-edge</td>
<td>$\min_{p,q \in S} \operatorname{dist}(p, q)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote-clique</td>
<td>$\sum_{p,q \in S} \operatorname{dist}(p, q)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote-tree</td>
<td>$\operatorname{wt}(\operatorname{MST}(S))$, weight of the minimum spanning tree of $S$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote-cycle</td>
<td>$\min_C \operatorname{wt}(C)$ where $C$ is a TSP tour on $S$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote $t$-trees</td>
<td>$\min_{S=S_1\ldots S_t} \sum_{i=1}^t \operatorname{wt}(\operatorname{MST}(S_i))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote $t$-cycles</td>
<td>$\min_{S=S_1\ldots S_t} \sum_{i=1}^t \operatorname{wt}(\operatorname{TSP}(S_i))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote-star</td>
<td>$\min_{p \in S} \sum_{q \in S \setminus {p}} \operatorname{dist}(p, q)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote-bipartition</td>
<td>$\min_{B} \operatorname{wt}(B)$, where $B$ is a bipartition (i.e., bisection) of $S$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Remote-pseudoforest</td>
<td>$\sum_{p \in S} \min_{q \in S \setminus {p}} \operatorname{dist}(p, q)$</td>
<td>$O(\log k)$</td>
</tr>
<tr>
<td>Remote-matching</td>
<td>$\min_{M} \operatorname{wt}(M)$, where $M$ is a perfect matching of $S$</td>
<td>$O(\log k)$</td>
</tr>
<tr>
<td>Max $k$-Coverage</td>
<td>$\sum_{i \leq d} \max_{p \in S} p_i$, where $p_i$ denotes the $i$th coordinate of $p$</td>
<td>no $\frac{\sqrt{k}}{\log k}$ approx. coreset of size $k^3$</td>
</tr>
</tbody>
</table>

**Table 6.1:** Notions of diversity considered in this work. We use $S = S_1|\ldots|S_t$ to denote that $S_1\ldots S_t$ is a partition of $S$ into $t$ sets.
example of submodular maximization [NWF78, IB], the maximum coverage problem has been recently studied from distributed computation perspective [CKW10], e.g., in the map-reduce framework [CKT10, KMVV13]. Our negative result for existence of coresets for this problem implies that one cannot use the simple coreset approach to solve this problem in distributed or streaming settings.

6.1.2 Techniques

Our techniques for constructing composable coresets rely on offline algorithms that solve the corresponding diversity maximization problems. The three algorithms are given in Preliminaries 6.2. Our contribution is to show that the solutions produced by those algorithms satisfy the composable coreset property. The basic idea is to show that, for each algorithm, one can construct a mapping from each element in the optimum solution (to the whole data set) and an element of the coreset. This correspondence is then used to bound the error incurred by the coreset. Note that for the remote-edge diversity measure, this analysis is analogous to the analysis in [AAYI+] (although that analysis was focused on the k-center clustering as opposed to the diversity maximization).

6.1.3 Related Work

Composable coresets: The notion of coresets has been introduced in [AHV04]. Informally, a coreset for an optimization problem is a subset of the data with the property that solving the underlying problem on the subset gives an approximate solution for the original data. The notion is somewhat generic, and many variations of coresets exist. The notion of composable coresets used in this work has been implicit in earlier works that used coresets for streaming applications. For example, the paper [AHV04] (Section 5) specifies composability properties of ϵ-kernels (a variant of coresets) that are very similar to ours. To avoid confusion, in this chapter the
term “coreset” always means “composable coreset” according to the definition in the introduction.

The notion of composable coresets is related to the notion of mergeable summaries introduced in [ACH+12]. The main difference between the two notions is that aggregating mergeable summaries does not increase the approximation error, while in our case the error amplifies (similarly to [GM16]). In particular, every mergeable summary that is obtained by taking a sub-set of the data is also a composable coreset, but the opposite does not hold.

**Diversity Maximization.** The diversity maximization problem studied in this chapter generalizes the maximum dispersion problem [HRT97, GS09, BGM11]. This problem has been explored in the context of diversity maximization for recommender systems [GS09], and commerce search [BGM11]. A 2-approximation greedy algorithm has been developed for the unconstrained variant of this problem [HRT97], and the variant with knapsack constraints [BGM11]. More recently, local search algorithms have been developed to get a 2-approximation algorithm for the maximum dispersion problem under matroid constraints [AMT13, BJLY17].

**Diversity in Recommender Systems and Web Search.** Ranking and relevance maximization along with diversification have been extensively studied in recommender systems, web search, and database systems. In the context of web search, maximizing diversity has been explored as a post-processing step [CG98, XCYH06]. Other papers explore ranking while taking into account diversity by a query reformulation for re-ranking the top searches [RD06] or by sampling the search results by reducing homogeneity [ABC06]. Other methods are based on clustering results into groups of related topics [KLR+04], or expectation maximization for estimating the model parameters and reaching an equilibrium [RBS10]. Moreover, in the context of recommender systems, diversification has been explored in various recent papers [ZMKL05, YLAY09]. For example, topical diversity maximization is discussed in [ZMKL05], and explanation-based diversity maximization is explored in [YLAY09]. Finally, this topic has been also explored in database systems for example by presenting decision trees to users [CL07].
6.2 Preliminaries

We start by formalizing the notion of diversity used in this paper.

**Definition 6.2.1.** For a given set $S \subseteq U$, its $k$-diversity is defined as $\text{div}_k(S) = \max_{S' \subseteq S | |S'|=k} \text{div}(S')$. We also refer to the set $S'$ maximizing this $k$-diversity as the optimal $k$-subset of $S$. Note that $k$-diversity is not defined in the case where $|S| < k$.

**Definition 6.2.2.** Let $\text{div}$ be a diversity function defined for subsets of $U$. A function $c(S)$ that maps a set $S \subseteq U$ into one of its subsets is called an $\alpha$-composable coreset ($\alpha \geq 1$) for $\text{div}$, if for any collection of sets $S_1 \ldots S_L \subseteq U$ with $|S_i| \geq k$, we have

$$\text{div}_k(c(S_1) \cup \ldots \cup c(S_L)) \geq \frac{1}{\alpha} \cdot \text{div}_k(S_1 \cup \ldots \cup S_L)$$

The coreset is of size $k'$ if for every $S$, $|c(S)| \leq k'$. Note that in general $k'$ does not need to be the same as $k$. For example, in all applications mentioned in the previous section, a coreset of size $k^2$ would work as well when $k$ is a constant. However, as it turns out, all our positive results give coresets of size $k$.

Our algorithms for constructing coresets are based on existing offline approximation algorithms for the corresponding diversity maximization problems. In the rest of this section we review three such algorithms: GMM, Local Search and Prefix.

6.2.1 GMM Algorithm

We use the following slight variation of the “GMM” algorithm introduced in [Gon85, RRT91]. The algorithm receives a set of points $S$, and the parameter $k$ as the input. Initially, it chooses some arbitrary point $a \in S$. Then it repeatedly adds the next point to the output set until there are $k$ points. More precisely, in each step, it greedily adds the point whose minimum distance to the currently chosen points is maximized. This algorithm was also utilized in [CH01] to find approximation algorithms for several dispersion problems.

It is easy to see that the running time of the algorithm is $O(nk)$. Also, observe
Algorithm 13: GMM

| Input  | $S$: a set of points, $k$: size of the subset |
| Output | $S'$: a subset of $S$ of size $k$. |

1: $S' \leftarrow$ An arbitrary point $a$
2: for $i = 2, \ldots, k$ do
3: \hspace{1em} find $p \in S \setminus S'$ which maximizes $\min_{x \in S'} \text{dist}(p, x)$
4: \hspace{1em} $S' \leftarrow S' \cup \{p\}$
5: end for
6: return $S'$

that if we define the radius value $r = \min_{p,q \in S'} \text{dist}(p, q)$ as the minimum pairwise distance in the set $S'$, it is easy to see that the following two properties hold:

- $\forall p \in S': \text{dist}(p, S' \setminus \{p\}) \geq r$
- $\forall p \in S: \text{dist}(p, S') \leq r$

Such sets $S'$ are said to have the anticonvex property.

6.2.2 Local Search Algorithm

Algorithm 14 shows the local search algorithm. This was used in [AMT13] to find a subset with approximate maximum diversity under matroid constraints for the case of Remote Clique. The algorithm iteratively improves the current solution by a factor of $(1 + \varepsilon/n)$ and finds a more diverse set of $k$ points. Since the initial set contains the two farthest points, the total number of iterations needed is at most $\log_{1+\varepsilon/n}(k^2) = O\left(\frac{n}{\varepsilon} \log k\right)$.

6.2.3 Prefix Algorithm

The Prefix algorithm 15 was introduced in [CH01] which is used to solve the approximate maximum dispersion problem in the case of Remote Pseudo-forest and Remote Matching. Note that the algorithm works only in the case when $k \leq n/2$. 
Algorithm 14: Local Search Algorithm

**Input** $S$: a set of points, $k$: size of the subset  
**Output** $S'$: a subset of $S$ of size $k$.

1: $S' \leftarrow$ An arbitrary set of $k$ points which contains the two farthest points  
2: **while** there exists $p \in S \setminus S'$ and $p' \in S'$ such that $\text{div}(S' \setminus \{p'\} \cup \{p\}) \geq \text{div}(S')(1 + \frac{\varepsilon}{n})$ **do**  
3: $S' \leftarrow S' \setminus \{p'\} \cup \{p\}$  
4: **end while**  
5: **return** $S'$

Algorithm 15: PREFIX Algorithm

**Input** $S$: a set of points, $k$: size of the subset  
**Output** $S'$: a subset of $S$ of size $k$.

1: Run GMM obtaining a set $Y = \{y_1, \cdots, y_k\}$ with corresponding radii $r_1, \cdots, r_k$.  
2: $q \leftarrow$ the value from the set $\{1, \cdots, k - 1\}$ which maximizes $q \cdot r_q$.  
3: $Y_{q+1} \leftarrow$ the prefix subsequence of $Y$ of length $q + 1$  
4: $Q_i \leftarrow$ vertices of distance at most $r_q/2$ from $y_i$ for $i = 1, \cdots, q + 1$.  
5: $z \leftarrow \lfloor(q + 1)/2\rfloor$.  
6: $\{Q_i, \cdots, Q_i\} \leftarrow$ the $z$ sparsest spheres.  
7: $S' \leftarrow$ the centers of $\{Q_i, \cdots, Q_i\}$  
8: Add any set of $k - z$ vertices from $S \setminus \bigcup_{j=1}^{z} Q_i$ to $S'$  
9: **return** $S'$

### 6.3 Composable Coresets for Diversity Maximization

This section provides algorithms for finding composable coresets for different notions of diversity defined in Table 6.1. That is, we run one of the algorithms defined in Preliminaries section 6.2 to get $k$ points in each of the instances of the problem and prove their union is an approximate coreset for the union of the instances.

In all of the following cases, we let $S_1, \cdots, S_L \subseteq U$ be the subsets of $U$ that
correspond to the instances of the problem and let \( S = \bigcup_{i=1}^{L} S_i \) denote their union. For each such instance \( S_i \), we find a coreset \( T_i \) and we let \( T = \bigcup_{i=1}^{L} T_i \) denote the union of the coresets. Also we let \( O = \{o_1, \cdots, o_k\} \) be the optimal \( k \)-subset of \( S \), that is the subset of \( k \) points which maximizes the diversity. Moreover, we define \( O_i = \{o \in O \cap S_i | \forall \ j < i : o \notin S_j\} \) to be the set of points from the optimal set in each of the instances (we impose extra condition in order to make \( O_i \)'s a partition of \( O \)).

Next, for each notion of diversity, we describe how to choose \( T_i \) and compare \( k \)-diversity of \( T \) with that of \( S \), which is equal to the diversity of \( O \).

### 6.3.1 Remote Edge

**Lemma 6.3.1.** The GMM algorithm computes a 3-approximate composable coreset for the Remote Edge problem.

**Proof.** We run the GMM algorithm on each of the sets \( S_i \) and let \( T_i = GMM(S_i) \) be the point set returned by the GMM and we let \( r_i \) denote the radius of \( T_i \). Let \( T = \bigcup_{i=1}^{L} T_i \) denote the union of the coresets, and set \( r = \max_i r_i \) to be the maximum radius over the instances. The goal is to prove that \( \text{div}_k(T) \geq \frac{1}{3} \text{div}_k(S) \).

Define a mapping \( f : O_i \to T_i \) which maps each point \( o \in O_i \) to one of its closest points in the set \( T_i \), i.e., \( \text{dist}(o, f(o)) = \text{dist}(o, T_i) \). By the anticover property of GMM we have \( \text{dist}(o, f(o)) \leq r_i \leq r \). Note that since \( O_i \)'s form a partition of \( O \), for any \( o \in O \), we can define \( f(o) = f_i(o) \) if \( o \in O_i \).

It is easy to see that for any \( i \), since \( T \) is a superset of \( T_i \), then \( \text{div}_k(T) \geq \text{div}(T_i) = r_i \) and thus \( \text{div}_k(T) \geq r \). Next, note that if for two points \( o_1, o_2 \in O \), we have \( f(o_1) = f(o_2) \), then

\[
\text{div}(O) \leq \text{dist}(o_1, o_2) \leq \text{dist}(o_1, f(o_1)) + \text{dist}(o_2, f(o_2)) \\
\leq 2r \leq 2\text{div}_k(T)
\]

and the lemma is proved. Otherwise \( f \) is a 1-to-1 mapping. Now if \( \text{div}(O) \leq 3r \leq 3\text{div}_k(T) \) then in this case the lemma is proved as well. Otherwise, we can assume that for any pair of points \( o_1, o_2 \in O \), \( \text{dist}(o_1, o_2) \geq 3r \) and thus \( \text{div}(O) \geq 3r \). Hence,
by triangle inequality
\[
\text{dist}(f(o_1), f(o_2)) \\
\geq \text{dist}(o_1, o_2) - \text{dist}(o_1, f(o_1)) - \text{dist}(o_2, f(o_2)) \\
\geq \text{div}(O) - 2r \\
\geq \text{div}(O) - 2\text{div}(O)/3 \\
\geq \text{div}(O)/3
\]
since this holds for any pair \(o_1, o_2\), the set \(\{f(o_1), \cdots, f(o_k)\}\) has diversity at least \(\text{div}(O)/3\) and thus \(\text{div}_k(T) \geq \text{div}(O)/3\) and the lemma is proved. \(\square\)

6.3.2 Remote Clique, Remote Star and Remote Bipartition

In this section, we show that the local search algorithm gives a constant-factor approximation for the following diversity notions: Remote Clique, Remote Star and Remote Bipartition.

**Lemma 6.3.2.** The local search algorithm computes a constant-factor approximate composable coreset for the remote-clique problem.

**Proof.** We run the Local Search algorithm on each of the sets \(S_i\) and let \(T_i = \text{LS}(S_i)\) be the point set returned by the Local Search and let \(r_i\) represent the normalized diversity of the corresponding sets \(T_i\), i.e., \(r_i = \frac{1}{(k^2)} \text{div}(T_i)\) and set \(r = \max_i r_i\).

**Claim 6.3.3.** There exists a 1-to-1 mapping \(f : O \rightarrow T\) such that \(\text{dist}(o, f(o)) \leq 25r\) for any \(o \in O\)

**Proof.** Build an unweighted bipartite graph \(G_x = (V_O, U_T, E_x)\) with vertices of one side corresponding to \(O\) and vertices of the other side corresponding to \(T\) as follows. For any \(o \in O\) and \(s \in T\), we connect \(v_o \in V_O\) to \(u_s \in U_T\) iff \(\text{dist}(o, s) \leq x \times r\). Now, take any \(o \in O\) and suppose that \(o \in O_i \setminus T_i\), that is, \(o\) is in the \(i\)th instance but has not been selected by LocalSearch algorithm. However, since no more improvement
on the set $T_i$ could be made, we have

$$\sum_{s \in T_i} \text{dist}(o, s) \leq (k - 1)(1 + \frac{\varepsilon}{n})r_i \leq kr$$

Note that since $|T_i| = k$, thus for at least $(1 - 1/x)$ fraction of the values $s$ in the above equation, we have $\text{dist}(o, s) \leq xr$ and therefore the corresponding edges in the graph $G_x$ exist. Thus the degree of each vertex $v_o$ corresponding to $o \in O \setminus T$ is at least $k(1 - 1/x)$.

First, take the graph $G_3$. If $G_3$ has a matching which saturates the vertices of $V_O$, then the claim is proved. Otherwise, let $M$ be a maximal matching in $G_3$ such that for any point $o \in O \cap T$, the corresponding vertices $v_o$ and $u_o$ are matched together. This means that the points corresponding to the set of unmatched vertices in $U_T$ (which we denote by $T \setminus M$) is disjoint from $O$, and also $O \setminus M$ is disjoint from $T$. Let $A = O \setminus M$ be the set of points which corresponds to the unmatched vertices. Then for any point $a \in A$, since $a \notin T$, the degree of $v_a$ is at least $2k/3$, and since $M$ is a maximal matching, all the neighbors of $v_a$ should be matched in $M$. Therefore there are at least $2k/3$ points $o \in O \setminus \{a\}$ such that $\text{dist}(o, a) \leq 6r$.

Now take the graph $G_{25}$. If all vertices in $V_A = V_O \setminus M$ are neighbors to all vertices in $U_T \setminus M$, then clearly $G_{25}$ has a saturating matching for $O$ and thus the claim is proved. Otherwise there exists a point $a \in A$ and $s \in T \setminus M$ such that $\text{dist}(a, s) > 25r$.

Let $B \subset O$ be the set of points whose distance is at most $6r$ from $a$. Then as we proved earlier $|B| > 2k/3$. Hence, if we replace the point $a$ in the set $O$ with the point $s$ to get the set $O'$ (note that since $T \setminus M$ is disjoint from $O$, we have $s \notin O$), the diversity will increase as follows.
\[
div(O') - \div(O) = \sum_{o \in O \setminus \{a\}} \dist(s, o) - \dist(a, o)
\]
\[
= \sum_{o \in B \setminus \{a\}} \dist(s, o) - \dist(a, o)
+ \sum_{o \in O \setminus B} \dist(s, o) - \dist(a, o)
\geq \sum_{o \in B \setminus \{a\}} \dist(a, s) - 2\dist(a, o)
- \sum_{o \in O \setminus B} \dist(a, s)
\geq 2\frac{k}{3} \times (\dist(a, s) - 12r) - \frac{k}{3} \times \dist(a, s)
= \frac{k}{3} (\dist(a, s) - 24r) \geq kr/3
\]

which contradicts the fact that \(O\) has the optimal diversity. Therefore the claim holds. \(\square\)

As claim 6.3.3 suggests, there is a 1-to-1 mapping between the vertices of \(O\) and the vertices of \(T\) such that for each \(o \in O\) we have \(\dist(o, f(o)) \leq 25r\). First of all note that if \(\binom{k}{2} \times r \geq \div(O)/51\) the theorem is proved since for one of the \(T_i\) we have \(\div(T_i) = \binom{k}{2} \times r\) and thus

\[
\div_k(T) \geq \div(T_i) = \binom{k}{2} \times r \geq \div(O)/51
\]

Otherwise, we have that

\[
\div_k(T) \geq \sum_{o_1, o_2 \in O} \dist(f(o_1), f(o_2))
\geq \sum_{o_1, o_2 \in O} \dist(o_1, o_2) - \dist(o_1, f(o_1)) - \dist(o_2, f(o_2))
\geq \div(O) - \binom{k}{2} \times 50r
\geq \div(O)(1 - 50/51) = \div(O)/51
\]
So the lemma is proved and the algorithm computes a 51-approximate coreset of size $k$. □

**Corollary 6.3.4.** *Local Search algorithm computes a constant factor coreset for the minimum star and minimum bipartition problems as well.*

*Proof.* First note that for a set of $k$ points $Q$, a star is the tree achieved by connecting one point to all the others, and its weight is sum of the weights of its edges. Also a bipartition of $Q$ is a bipartite graph which divides the vertices of $Q$ into two parts of cardinality $k/2$ and its weight is the sum of all the edges between the two parts. It can easily be seen that

- by symmetry $wt(\text{minimum star}(Q)) \leq 2wt(\text{clique}(Q))/k$
- by triangle inequality

$$wt(\text{clique}(Q)) \leq k \times wt(\text{minimum star}(Q))$$

and that

- $wt(\text{minimum bipartition}(Q)) \leq wt(\text{clique}(Q))$
- by triangle inequality

$$wt(\text{clique}(Q)) \leq 5 \times wt(\text{minimum bipartition}(Q))$$

Therefore the same algorithm computes a constant factor coreset for these two problems as well. □

### 6.3.3 Remote Tree, Remote Cycle, Remote $t$-trees and Remote $t$-cycles

**Lemma 6.3.5.** *The GMM algorithm computes a 6-approximate coreset for the remote-tree problem.*
Proof. We run the GMM algorithm on each of the sets $S_i$ and let $T_i = GMM(S_i)$ be the point set returned by the GMM and we let $r_i$ denote the radius of $T_i$. Let $T = \bigcup_{i=1}^L T_i$ denote the union of the coresets, and set $r = \max_i r_i$ to be the maximum radius over the instances. Now define a mapping (this time not a 1-to-1) $f : O_i \rightarrow T_i$ which maps each point $o \in O_i$ to one of its closest points in the set $T_i$, i.e., $\text{dist}(o, f(o)) = \text{dist}(o, T_i)$. By anticover property of GMM we have $\text{dist}(o, f(o)) \leq r_i \leq r$.

It is easy to see that for any $i$, $\text{div}_k(T) \geq \text{div}(T_i) \geq (k-1)r_i$ (since the minimum pairwise distance in $T_i$ is $r_i$), and thus $\text{div}_k(T) \geq (k-1)r$. Now if $\text{div}(O) \leq 3(k-1)r \leq 3\text{div}_k(T)$, then the lemma is proved. Otherwise let $F = \text{range}(f) = \{f(o) | o \in O\}$ (note that $F$ is a subset of $T$), and let $F^+ \subset T$ be an arbitrary superset of $F$ of size $k$. Then by triangle inequality and shortcutting

$$\text{div}(O) = \text{wt}(MST(O)) \leq \text{wt}(MST(F)) + kr$$

$$\leq \text{wt}(MST(F)) + 2(k-1)r$$

which uses the fact that $k > 1$, otherwise any one point is a solution. Next, note that given the $MST(F^+)$, we can double the edges and traverse them using $DFS$ and remove the vertices not in $F$ by shortcutting. Hence, by triangle inequality, we find a Hamiltonian cycle of length at most $2\text{wt}(MST(F^+))$ on the set $F$, therefore we have $\text{wt}(MST(F)) \leq 2\text{wt}(MST(F^+))$ and thus

$$\text{div}_k(T) \geq \text{wt}(MST(F^+))$$

$$\geq \text{wt}(MST(F))/2$$

$$\geq \frac{1}{2}[\text{div}(O) - 2(k-1)r]$$

$$\geq \frac{\text{div}(O)}{3} - \frac{\text{div}(O)}{3}$$

$$\geq \frac{\text{div}(O)}{6}$$

Lemma 6.3.6. The same algorithm computes a 6 coreset for the Remote-t-tree.
Proof. Let \( wt(MST_t(A)) \) of a set of points \( A \), denote the minimum sum of the weights of spanning trees achieved by dividing \( A \) into \( t \) sets, i.e.,
\[
\min_{A = A_1 | \ldots | A_t} \sum_{i=1}^t wt(MST(A_i)),
\]
where \( A = A_1 | \ldots | A_t \) is a partition of \( A \) into \( t \) sets.

We run the GMM algorithm on each of the sets \( S_i \) and let \( T_i = GMM(S_i) \) be the point set returned by the GMM and we let \( r_i \) denote the radius of \( T_i \). Let \( T = \bigcup_{i=1}^L T_i \) denote the union of the coresets, and set \( r = \max_i r_i \) to be the maximum radius over the instances. Now define a mapping (not necessarily a 1-to-1) \( f : O_i \rightarrow T_i \) which maps each point \( o \in O_i \) to one of its closest points in the set \( T_i \), i.e., \( \text{dist}(o,f(o)) = \text{dist}(o,T_i) \). By properties of GMM we have \( \text{dist}(o,f(o)) \leq r_i \leq r \).

First of all, note that it only makes sense if \( t \leq k/2 \) otherwise in any optimum solution, at least \( 2t - k \) of the partitions include exactly one of the \( k \) points and therefore incur no cost. So instead we could consider the problem of choosing \( k' = k - t \) points and having \( t' = 2(k - t) \) partitions in which \( t' \leq k'/2 \).

It is easy to see that for any \( i \), \( \text{div}_k(T) \geq \text{div}(T_i) \geq (k - t)r_i \) (since the minimum pairwise distance in \( T_i \) is \( r_i \)), and thus \( \text{div}_k(T) \geq (k - t)r \). Now if \( \text{div}(O) \leq 3(k-t)r \leq 3\text{div}_k(T) \), then the lemma is proved. Otherwise let \( F = \text{range}(f) = \{ f(o) | o \in O \} \) (note that \( F \) is a subset of \( T \)), and let \( F^+ \subset T \) be an arbitrary superset of \( F \) of size \( k \). Then by triangle inequality and shortcutting
\[
\text{div}(O) = wt(MST_t(O)) \leq wt(MST_t(F)) + kr \\
\leq wt(MST_t(F)) + 2(k - t)r
\]
which uses the fact that \( t \leq k/2 \). Next, note that given the \( MST_t(F^+) \), we can double the edges and traverse them using \( DFS \) and remove the vertices not in \( F \) by shortcutting. Hence, by triangle inequality, we find a Hamiltonian cycle in each part of the partition with total length at most \( 2wt(MST_t(F^+)) \) on the set \( F \), therefore we
have \(\text{wt}(\text{MST}_i(F)) \leq 2\text{wt}(\text{MST}_i(F^+))\) and thus

\[
\text{div}_k(T) \geq \text{wt}(\text{MST}_i(F^+)) \geq \text{wt}(\text{MST}_i(F))/2 \\
\geq \frac{1}{2}[\text{div}(O) - 2(k - t)r] \\
\geq \text{div}(O)/2 - \text{div}(O)/3 \\
\geq \text{div}(O)/6
\]

\[\square\]

**Corollary 6.3.7.** Note that since the minimum TSP tour is within a factor 2 of the MST, the above algorithm also computes a constant factor coreset for the remote-cycle problem and remote t-cycle problem.

### 6.3.4 Remote Pseudoforest and Remote Matching

**Lemma 6.3.8.** The GMM algorithm computes a \(O(\log k)\) coreset for the remote-pseudoforest problem.

**Proof.** We run the GMM algorithm on each of the sets \(S_i\) and let \(T_i = \text{GMM}(S_i)\) be the point set returned by the GMM. Let \(T = \bigcup_{i=1}^{L} T_i\) denote the union of the coresets.

It is shown in page 11 of the paper [CH01] that when we run the Prefix algorithm on an input set \(A\), the diversity achieved by this algorithm is at least \(q \cdot r_A/4\) and that \(q \cdot r_A/4 \geq \text{div}_k(A)/O(\log k)\). Next, we compare running the PREFIX algorithm on the set \(S\) and on the set \(T\). Let \(r_S^1, \cdots, r_S^k\) be the radii defined in line 1 of Algorithm 6.2.3, and let \(q_S\) be the index chosen in line 2, when we run it on the set \(S\). Similarly, let us define \(r_T^1, \cdots, r_T^k\) and \(q_T\), when we run the algorithm on the set \(T\).

However by Lemma 6.3.1, GMM algorithm computes a coreset for minimum pairwise distances. Together with the fact that running GMM in the Prefix algorithm on the sets \(S\) and \(T\) preserves the radii up to a constant factor, we get that \(r_T^i \geq r_S^i/c\), for any value of \(i \leq k\) and some constant \(c\). The diversity achieved by the prefix algorithm is therefore \(\text{div}_k(T) \geq q_T^T \cdot r_T^T/4 \geq q_S^S \cdot r_S^T/4 \geq q_S^S \cdot r_S^S/(4c) \geq \frac{\text{div}_k(S)}{O(\log k)}\)

For the same reason the GMM algorithm computes a \(O(\log k)\) coreset for the
remote-matching problem as well with the only difference that the value of the matching achieved by the prefix algorithm when we run in on the input set $A$ is at least $qr_q/8$ instead of $qr_q/4$.

# 6.4 Non-existence of Coreset for the Max $k$-Coverage

An instance of the max $k$-coverage problem is a collection of sets. The objective is to find $k$ sets in this collection whose union has the maximum size.

**Theorem 6.4.1.** For any $\alpha < \frac{\sqrt{k}}{\log k}$ and any constant $\beta > 1$, there is no $\alpha$-approximate coreset of size $k^\beta$ for the max $k$-coverage problem.

**Proof.** Let $\mathcal{U} = \{1, \ldots, N\}$ for a large $N$ and $k = m^2$. We construct a number of instances of the problem as follows: For every subset $S \subset \mathcal{U}$ of size $m^2$, we have an instance $I_S$ consisting of all $m$-subsets of $S$. Assume, for contradiction, that there is an $\alpha$-approximate coreset, and let $C_S$ denote the coreset on the instance $I_S$.

Now, fix a $m^2$-set $S$, and let $R$ be a random $m$-subset of $S$. For each fixed $A \in C_S$, the random variable $|A \cap R|$ is distributed according to the binomial distribution $Bin(m, \frac{1}{m})$. The probability that the value of this variable is at least $t$ is at most $\binom{m}{t} \frac{1}{m^t} < \frac{1}{t!}$. With $t = \log m$, this probability is at most $O(m^{-c})$ for every constant $c$. Using the union bound and the fact that $|C_S| \leq m^{2\beta}$, we get: $\Pr[\exists A \in C_S : |A \cap R| > \log m] < O(m^{2\beta - c})$ for every constant $c$. We say that $R$ is an easy subset of $S$ if $\exists A \in C_S : |A \cap R| > \log m$. Therefore for every $S$, at most a $O(m^{-\gamma})$ fraction of the $m$-subsets of $S$ are easy, for every $\gamma$.

We construct a graph whose vertex set is the set of all $m^2$ subsets of $\mathcal{U}$. Two $m^2$-sets $S_1$ and $S_2$ in this graph are adjacent if $|S_1 \setminus S_2| = m$. We say that $S_1$ marks a neighbor $S_2$ as bad if $S_1 \setminus S_2$ is an easy subset of $S_1$. By the above argument, each vertex $S_1$ marks at most an $O(m^{-\gamma})$ fraction of its neighbors as bad. Since the total indegree of nodes in a graph is equal to the total outdegree, there must be a vertex $S_1$ in this graph such that at most an $O(m^{-\gamma})$ fraction of its neighbors have marked
$S_1$ as bad. Therefore, at most an $O(m^{-\gamma})$ fraction of the neighbors of $S_1$ have either marked $S_1$ as bad or $S_1$ has marked them as bad. We call these neighbors the bad neighbors of $S_1$, and the remaining neighbors the good ones.

We now pick a collection of $m^2$-sets $S_1, S_2, \ldots$ as follows: $S_1$ is the vertex defined above. $S_2$ is an arbitrary good neighbor of $S_1$. $S_{i+1}$ is a good neighbor of $S_1$ such that $S_{i+1} \setminus S_1$ does not intersect any of the sets $S_j \setminus S_1$ for $j \leq i$. We argue that for any $i < m^2$, there is a set $S_{i+1}$ with the above properties that we can pick. This is because for any $x \in \mathcal{U} \setminus S_1$, the fraction of neighbors of $S_1$ that contain $x$ is precisely $\frac{m^2}{N-m^2}$. Therefore, by the union bound, the fraction of neighbors of $S_1$ that contain any of the elements of $S_j \setminus S_1$ for $j \leq i$ is at most $\frac{m^2i}{N-m^2}$, which is less than $1/2$ for $N > 3m^4$. This means that at most a $\frac{1}{2} + O(m^{-\gamma}) < 1$ fraction of neighbors of $S_1$ either have intersection with some $S_j \setminus S_1$ for $j \leq i$ or are bad. Thus, we can find $S_{i+1}$ with the desired properties, for $i < m^2$.

Now, consider the union of the instances $I_{S_1}, I_{S_2}, \ldots, I_{S_{m^2-m}}$. This instance has a perfect $k$-coverage solution: pick $m$ non-overlapping $m$-subsets of $S_1$ to cover $S_1$, and for every $i > 1$, pick the $m$-subset $S_i \setminus S_1$. The total number of subsets picked is $m + m^2 - m = k$, and all of the $O(m^3)$ elements in $S_1 \cup \cdots \cup S_{m^2-m}$ are covered. On the other hand, by our construction, we know that for every $i > 1$, $S_i \setminus S_1$ is not an easy subset of $S_i$. Therefore, any set in $C_{S_i}$ covers at most $\log m$ elements of $S_i \setminus S_1$, and for $j \neq i$, $C_{S_j}$ does not cover any element of $S_i \setminus S_1$. Thus, a collection of $k$ sets in $\bigcup_i C_{S_i}$ can cover at most $k \log m$ elements in $\bigcup_i (S_i \setminus S_1)$ plus $m^2$ elements of $S_1$. This means that the ratio of the best solution on the union of these instances and the solution that is limited to the union of the coresets is at most $\frac{m^2 \log m + m^2}{m^3} < \frac{\log k}{\sqrt{k}}$. \(\square\)

### 6.5 Application to Approximate Nearest Neighbor

In this section, we briefly describe how we can apply the aforementioned coresets to solve $k$-diverse near neighbor problem. The problem is defined as follows. Given a query point $q \in \mathcal{U}$, the goal is to report the maximum diversity set $S$ of $k$ points in the ball of radius $r$ around $q$. The points in the set $S$ are chosen from a dataset of points
$P \subseteq U$ of size $n$ which is given to the algorithm at the preprocessing time. We would like to answer queries in sublinear time which necessitates solving the approximate problem. The approximate $k$-diverse Near Neighbor is defined as follows. For some approximation factors $c > 1$ and $\alpha > 1$, we allow the points of the reported set $S$ to be within distance $cr$ of the query point, i.e., $S \subseteq P \cap B(q, cr)$. Moreover, we require that the diversity of the set $S$ is within an $\alpha$ factor of the $k$-diversity of the optimal set, i.e., $\text{div}(S) \geq \frac{1}{\alpha} \text{div}_k(P \cap B(q, r))$.

The definitions and algorithm mentioned here are from [AAYIM13, AAYI+] and are only included for completeness. Please see the original papers for the detailed theoretical [AAYI+] or experimental [AAYIM13] analysis of its performance. The algorithm uses the techniques of locality-sensitive hashing. Its basic idea is to hash the data and query points in a way that the probability of collision is much higher for points that are close to each other, than for those which are far apart. Formally, we require the following.

**Definition 6.5.1.** A family $H = h : U \rightarrow U$ is $(r_1, r_2, p_1, p_2)$-sensitive for $(U, \text{dist})$, if for any $p, q \in U$, we have

- if $\text{dist}(p, q) \leq r_1$, then $\Pr_{H}[h(q) = h(p)] \geq p_1$
- if $\text{dist}(p, q) \leq r_2$, then $\Pr_{H}[h(q) = h(p)] \leq p_2$

In order for a locality sensitive family to be useful, it has to satisfy inequalities $p_1 > p_2$ and $r_1 < r_2$.

Given an LSH family, the algorithm creates $L$ hash functions $g_1, g_2, \ldots, g_L$, as well as the corresponding hash arrays $A_1, A_2, \ldots, A_L$. Each hash function is of the form $g_i = < h_{i,1}, \ldots, h_{i,K} >$, where $h_{i,j}$ is chosen uniformly at random from $H$. Then each point $p$ is stored in bucket $g_i(p)$ of $A_i$ for all $1 \leq i \leq L$. In order to answer a query $q$, we then search points in $A_1[g_1(q)] \cup \cdots \cup A_L[g_L(q)]$. That is, in each array, we only retrieve points from the single bucket which corresponds to the query point $q$.

The aforementioned algorithm does not limit the number of points stored in a bucket, and hence its running time is unbounded. To avoid this problem we proceed
as follows. During the preprocessing stage, for each of the buckets in all arrays $A_i$, we replace the bucket content by its coreset, using the algorithms presented in this paper. Then, given a query point $q$, we collect the coreset points from the corresponding buckets of $q$, i.e., $T = \bigcup_i c(A_i[g_i(q)])$. Since the coresets has polynomial size in $k$, and the total number of hash functions $L$ is sublinear in $n$, then the total number of points we collect in $T$ is sublinear in $n$. By properties of coresets, the $k$-diversity of the set $T$ is comparable to $k$-diversity of the set $S = \bigcup_i A_i[g_i(q)]$. Moreover, one can set the parameters of $LSH$ (i.e., $L$ and $K$) such that with high probability the two following conditions hold:

- $P \cap B(q, r) \subset S$, every point in the $r$-neighborhood of $q$ is included in the set $S$.
- $S \subset B(q, cr)$, any retrieved point is in the $cr$-neighborhood of $q$, i.e., there are no outliers.

Thus, if $\beta$ shows the approximation factor of the coreset, then the value of $\text{div}_k(T)$ is within $\beta$ factor of the value $\text{div}_k(S)$. Since $S$ is a superset of $(P \cap B(q, r))$, we get that $\text{div}_k(T)$ is within $\beta$-factor of $\text{div}_k(P \cap B(q, r))$. Therefore we can run the “offline” algorithm on the set $T$ to get an approximate $k$-diverse subset of $T$ whose diversity approximates the diversity of the optimal set.

More specifically, if $\beta'$ shows the best approximation factor for the “offline” version of diversity approximation, with running time of $T(m)$ on $m$ points, we can get final bounds as follows. We can achieve approximation factor $\alpha = \beta\beta'$, with query time of

$$O(T(k(\log k)^{1+\frac{1}{c+1}} n^{\frac{1}{c+1}}) + \frac{d}{r} (\log k)^{1+\frac{1}{c+1}} n^{\frac{1}{c+1}} \log n)$$

and data structure space equal to $O((n \log k)^{1+\frac{1}{c+1}} + nd)$. 

156
Part III

The Set Cover Problem
Chapter 7

The Streaming Set Cover Problem

7.1 Introduction and Background

In this chapter, given a universe of elements $U$ and a collection of sets $S_1, \cdots, S_m$ over those elements, we aim to design a streaming algorithm for the Set Cover problem. The objective is to design an efficient approximation algorithm for the Set Cover problem that performs few passes over the data, and uses as little memory as possible.

The last few years have witnessed a rapid development of new streaming algorithms for the Set Cover problem, in both theory and applied communities, see [SG09, CKW10, KVV13, ER14, DIMV14, CW16]. Table 7.1 presents the approximation and space bounds achieved by those algorithms, as well as the lower bounds$^1$.

Related work. The semi-streaming Set Cover problem was first studied by Saha and Getoor [SG09]. Their result for Max $k$-Cover problem implies a $O(\log n)$-pass $O(\log n)$-approximation algorithm for the Set Cover problem that uses $\tilde{O}(n^2)$ space. Adopting the standard greedy algorithm of Set Cover with a thresholding technique leads to $O(\log n)$-pass $O(\log n)$-approximation using $\tilde{O}(n)$ space. In $\tilde{O}(n)$ space regime, Emek and Rosen studied designing one-pass streaming algorithms for the Set Cover problem [ER14] and gave a deterministic greedy based $O(\sqrt{n})$-

$^1$Note that the simple greedy algorithm can be implemented by either storing the whole input (in one pass), or by iteratively updating the set of yet-uncovered elements (in at most $n$ passes).
<table>
<thead>
<tr>
<th>Result</th>
<th>Approximation</th>
<th>Passes</th>
<th>Space</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy algorithm</td>
<td>(\ln n)</td>
<td>1</td>
<td>(O(n))</td>
<td></td>
</tr>
<tr>
<td>[SG09]</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O(n^2 \ln n))</td>
<td>R</td>
</tr>
<tr>
<td>[ER14]</td>
<td>(O(\sqrt{n}))</td>
<td>1</td>
<td>(\Theta(n))</td>
<td>R</td>
</tr>
<tr>
<td>[CW16]</td>
<td>(O(n^\delta))</td>
<td>(1/\delta - 1)</td>
<td>(\Theta(n))</td>
<td>R</td>
</tr>
<tr>
<td>[Nis02]</td>
<td>(1/2 \log n)</td>
<td>(O(\log n))</td>
<td>(\Omega(m))</td>
<td>R</td>
</tr>
<tr>
<td>[DIMV14]</td>
<td>(O(4^{1/\delta} \rho))</td>
<td>(O(4^{1/\delta}))</td>
<td>(O(mn^\delta))</td>
<td>R</td>
</tr>
<tr>
<td>[DIMV14]</td>
<td>(O(1))</td>
<td>(O(\log n))</td>
<td>(\Omega(mn))</td>
<td></td>
</tr>
<tr>
<td>Geometric Set Cover</td>
<td>(O(\rho/\delta))</td>
<td>(2/\delta)</td>
<td>(O(mn^\delta))</td>
<td>R</td>
</tr>
<tr>
<td>(s)-Sparse Set Cover</td>
<td>(O(1))</td>
<td>(1/2\delta - 1)</td>
<td>(\Omega(mn^\delta))</td>
<td>R</td>
</tr>
<tr>
<td>Table 7.1: Summary of past work and our results. The last column indicates if the scheme is randomized, (\rho) denotes the approximation factor of an off-line algorithm solving Set Cover, which is (\ln n) for the greedy, and 1 for exponential algorithm. Similarly, (\rho_g) denotes the approximation factor of an off-line algorithm solving geometric Set Cover. Finally, in the (s)-Sparse Set Cover problem, (s \leq n^\delta) denotes an upper bound on the sizes of the input sets. Our lower bounds for Set Cover and (s)-Sparse Set Cover hold for (m = O(n)). Moreover, [ER14] and [CW16] proved that their algorithms are tight. Here, and in the rest of the chapter, all log are in base two.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In a different regime which was first studied by Demaine et al., the goal is to design a "low" approximation algorithms (depending on the computational model, it could be $O(\log n)$ or $O(1)$) in the smallest possible space [DIMV14]. They proved that any constant pass deterministic $(\log n/2)$-approximation algorithm for the Set Cover problem requires $\tilde{O}(mn)$ space. It shows that unlike the results in $O(n)$-space regime, to obtain a sublinear "low" approximation streaming algorithm for the Set Cover problem in a constant number of passes, using randomness is necessary. Moreover, [DIMV14] presented a $O(4^{1/\delta})$-approximation algorithm that makes $O(4^{1/\delta})$ passes and uses $\tilde{O}(mn^\delta)$ memory space.

The Set Cover problem is not polynomially solvable even in the restricted instances with points in $\mathbb{R}^2$ as elements, and geometric objects (either all disks or axis parallel rectangles or fat triangles) in plane as sets [FG88, FPT81, HQ15]. As a result, there has been a large body of work on designing approximation algorithms for the geometric Set Cover problems. See for example [MRR14, AP14, AES10, CV07] and references therein.

### 7.1.1 Results

Despite the progress outlined above, however, some basic questions still remained open. In particular:

(A) Is it possible to design a single pass streaming algorithm with a "low" approximation factor\(^2\) that uses sublinear (i.e., $o(mn)$) space?

(B) If such single pass algorithms are not possible, what are the achievable trade-offs between the number of passes and space usage?

(C) Are there special instances of the problem for which more efficient algorithms can be designed?

In this work, we make a significant progress on each of these questions. Our upper and lower bounds are depicted in 7.1.

\(^2\)Note that the lower bound in [DIMV14] excluded this possibility only for deterministic algorithms, while the upper bound in [ER14, CW16] suffered from a polynomial approximation factor.
On the algorithmic side, we give a $O(1/\delta)$-pass algorithm with a strongly sub-linear $\tilde{O}(mn^\delta)$ space and logarithmic approximation factor. This yields a significant improvement over the earlier algorithm of Demaine et al. [DIMV14] which used exponentially larger number of passes. The trade-off offered by our algorithm matches the lower bound of Nisan [Nis02] that holds at the endpoint of the trade-off curve, i.e., for $\delta = \Theta(1/\log n)$, up to poly-logarithmic factors in space\(^3\). Furthermore, our algorithm is very simple and succinct, and therefore easy to implement and deploy.

Our algorithm exhibits a natural tradeoff between the number of passes and space, which resembles tradeoffs achieved for other problems [GM07, GM08, GO13]. We show a matching lower bound for the case when the approximation factor is equal to 1, i.e., the goal is to compute the optimal set cover. In particular, by an information theoretic lower bound, we show that any streaming algorithm that computes set cover using $(\frac{1}{23} - 1)$ passes must use $\tilde{\Omega}(mn^\delta)$ space (even assuming exponential computational power) in the regime of $m = O(n)$. Furthermore, we show that a stronger lower bound holds if all the input sets are sparse, that is if their cardinality is at most $s$. We prove a lower bound of $\tilde{\Omega}(ms)$ for $s = O(n^\delta)$ and $m = O(n)$.

We also consider the problem in the geometric setting in which the elements are points in $\mathbb{R}^2$ and sets are either discs, axis-parallel rectangles, or fat triangles in the plane. We show that a slightly modified version of our algorithm achieves the optimal $\tilde{O}(n)$ space to find an $O(\rho)$-approximation in $O(1)$ passes.

Finally, we show that any randomized one-pass algorithm that distinguishes between covers of size 2 and 3 must use a linear (i.e., $\Omega(mn)$) amount of space. This is the first result showing that a randomized, approximate algorithm cannot achieve a sub-linear space bound.

**Further research.** Recently Assadi et al. [AKL16] generalized the lower bound for the single pass algorithms to any approximation ratio $\alpha = O(\sqrt{n})$. More precisely they showed that approximating Set Cover within any factor $\alpha = O(\sqrt{n})$ in a single pass requires $\Omega(mn^\alpha)$ space. Very recently, Assadi [Ass17] proved a lower bound for

---

\(^3\)Note that to achieve a logarithmic approximation ratio we can use an off-line algorithm with the approximation ratio $\rho = 1$, i.e., one that runs in exponential time (see 7.2.8).
streaming algorithms with multiple passes which is tight up to polylog factors: any $\alpha$-approximation algorithm for Set Cover requires $\Omega(mn^{1/\alpha})$ space, even if it is allowed polylog($n$) passes over the stream, and even if the sets are arriving in a random order in the stream. This shows that the bounds trade off provided by our algorithm is essentially tight.

**Fractional streaming set cover.** In [IMR+17], we studied the Fractional Set Cover problem in the streaming model. This problem considers the relaxation of the set cover problem over a universe of $n$ elements and a collection of $m$ sets, where each set can be picked fractionally, with a value in $[0,1]$. A randomized $(1+\varepsilon)$-approximation algorithm that makes $p$ passes over the data, and uses $\tilde{O}(mn^{O(1/p\varepsilon)}+n)$ memory space is presented in [IMR+17]. The result is obtained by employing the multiplicative weights update framework in the streaming settings.

### 7.1.2 Techniques

**Basic idea.** Our algorithm is based on the idea that whenever a large enough set is encountered, we can immediately add it to the cover. Specifically, we guess (up to factor two) the size of the optimal cover $k$. Thus, a set is “large” if it covers at least $1/k$ fraction of the remaining elements. A small set, on the other hand, can cover only a “few” elements, and we can store (approximately) what elements it covers by storing (in memory) an appropriate random sample. At the end of the pass, we have (in memory) the projections of “small” sets onto the random sample, and we compute the optimal set cover for this projected instance using an offline solver. By carefully choosing the size of the random sample, this guarantees that only a small fraction of the set system remains uncovered. The algorithm then makes an additional pass to find the residual set system (i.e., the yet uncovered elements), making two passes in each iteration, and continuing to the next iteration.

Thus, one can think about the algorithm as being based on a simple iterative “dimensionality reduction” approach. Specifically, in two passes over the data, the algorithm selects a “small” number of sets that cover all but $n^{-\delta}$ fraction of the
uncovered elements, while using only $\tilde{O}(mn^\delta)$ space. By performing the reduction step $1/\delta$ times we obtain a complete cover. The dimensionality reduction step is implemented by computing a small cover for a random subset of the elements, which also covers the vast majority of the elements in the ground set. This ensures that the remaining sets, when restricted to the random subset of the elements, occupy only $\tilde{O}(mn^\delta)$ space. As a result the procedure avoids a complex set of recursive calls as presented in Demaine et al. [DIMV14], which leads to a simpler and more efficient algorithm.

**Geometric results.** Further using techniques and results from computational geometry we show how to modify our algorithm so that it achieves almost optimal bounds for the Set Cover problem on geometric instances. In particular, we show that it gives a $O(1)$-pass $O(\rho)$-approximation algorithm using $\tilde{O}(n)$ space when the elements are points in $\mathbb{R}^2$ and the sets are either discs, axis parallel rectangles, or fat triangles in the plane. In particular, we use the following surprising property of the set systems that arise out of points and disks: the the number of sets is nearly linear as long as one considers only sets that contain “a few” points.

More surprisingly, this property extends, with a twist, to certain geometric range spaces that might have quadratic number of shallow ranges. Indeed, it is easy to show an example of $n$ points in the plane, where there are $\Omega(n^2)$ distinct rectangles, each one containing exactly two points, see 7-1. However, one can “split” such ranges into a small number of canonical sets, such that the number of shallow sets in the canonical set system is near linear. This enables us to store the small canonical sets encountered during the scan explicitly in memory, and still use only near linear space.

We note that the idea of splitting ranges into small canonical ranges is an old idea in orthogonal range searching. It was used by Aronov et al. [AES10] for computing small $\varepsilon$-nets for these range spaces. The idea in the form we use, was further formalized by Ene et al. [EHR12].
Consider two parallel lines in the plane with positive slope. Place \(n/2\) points on each line such that all the points on the top line lie above and to the left of all the points on the bottom line. Let the set of rectangles for this instance be all the rectangles which have a point on the top line as their upper left corner and a point on the bottom line as their lower right corner. Clearly, we have \(n^2/4\) distinct rectangles (i.e., sets), each containing two points. As such, we cannot afford to store explicitly in memory the set system, since it requires too much space.

**Lower bounds.** The lower bounds for multi-pass algorithms for the Set Cover problem are obtained via a careful reduction from Intersection Set Chasing. The latter problem is a communication complexity problem where \(n\) players need to solve a certain “set-disjointness-like” problem in \(p\) rounds. A recent paper [GO13] showed that this problem requires \(\frac{n^{1+\Omega(1/p)}}{p^\Theta(1)}\) bits of communication complexity for \(p\) rounds. This yields our desired trade-off of \(\tilde{\Omega}(mn^\delta)\) space in \(1/2\delta\) passes for exact protocols for Set Cover in the communication model and hence in the streaming model for \(m = O(n)\). Furthermore, we show a stronger lower bound on memory space of sparse instances of Set Cover in which all input sets have cardinality at most \(s\). By a reduction from a variant of Equal Pointer Chasing which maps the problem to a sparse instance of Set Cover, we show that in order to have an exact streaming algorithm for \(s\)-Sparse Set Cover with \(o(ms)\) space, \(\Omega(\log n)\) passes is necessary. More precisely, any \((\frac{1}{2^s} - 1)\)-pass exact randomized algorithm for \(s\)-Sparse Set Cover requires \(\tilde{\Omega}(ms)\) memory space, if \(s \leq n^\delta\) and \(m = O(n)\).

Our single pass lower bound proceeds by showing a lower bound for a one-way communication complexity problem in which one party (Alice) has a collection of sets, and the other party (Bob) needs to determine whether the complement of his set is covered by one of the Alice’s sets. We show that if Alice’s sets are chosen at random,
Algorithm 16: A tight streaming algorithm for the (unweighted) Set Cover problem. Here, algOfflineSC is an offline solver for Set Cover that provides \( \rho \)-approximation, and \( c \) is some appropriate constant.

then Bob can decode Alice’s input by employing a small collection of “query” sets. This implies that the amount of communication needed to solve the problem is linear in the description size of Alice’s sets, which is \( \Omega(mn) \).

7.2 Streaming Algorithm for Set Cover

7.2.1 Algorithm

In this section, we design an efficient streaming algorithm for the Set Cover problem that matches the lower bound results we already know about the problem. In the Set Cover problem, for a given set system \((U, F)\), the goal is to find a subset \( I \subseteq F \), such that \( I \) covers \( U \) and its cardinality is minimum. In the following, we sketch the iterSetCover algorithm (see also 16).

In the iterSetCover algorithm, we have access to the algOfflineSC subroutine that solves the given Set Cover instance offline (using linear space) and returns a \( \rho \)-approximation.
approximate solution where \( \rho \) could be anywhere between 1 and \( \Theta(\log n) \) depending on the computational model one assumes. Under exponential computational power, we can achieve the optimal cover of the given instance of the Set Cover (\( \rho = 1 \)); however, under P \( \neq \) NP assumption, \( \rho \) cannot be better than \( c \cdot \ln n \) where \( c \) is a constant [Fei98, RS97, AMS06, Mos12, DS14] given polynomial computational power.

Let \( n = |U| \) be the initial number of elements in the given ground set. The \textit{iterSetCover} algorithm, needs to guess (up to a factor of two) the size of the optimal cover of \((U, \mathcal{F})\). To this end, the algorithm tries, in parallel, all values \( k \) in \( \{2^i \mid 0 \leq i \leq \log n\} \). This step will only increase the memory space requirement by a factor of \( \log n \).

Consider the run of the \textit{iterSetCover} algorithm, in which the guess \( k \) is correct (i.e., \( |\text{opt}| \leq k < 2|\text{opt}| \), where \( \text{opt} \) is an optimal solution). The idea is to go through \( O(1/\delta) \) iterations such that each iteration only makes two passes and at the end of each iteration the number of uncovered elements reduces by a factor of \( n^\delta \). Moreover, the algorithm is allowed to use \( \tilde{O}(mn^\delta) \) space.

In each iteration, the algorithm starts with the current ground set of uncovered elements \( U \), and copies it to a \textit{leftover} set \( L \). Let \( Z \) be a large enough uniform sample of elements \( U \). In a single pass, using \( Z \), we estimate the size of all large sets in \( \mathcal{F} \) and add \( S \in \mathcal{F} \) to the solution \( \text{sol} \) immediately (thus avoiding the need to store it in memory). Formally, if \( S \) covers at least \( \Omega(|U|/k) \) yet-uncovered elements of \( L \) then it is a heavy set, and the algorithm immediately adds it to the output cover. Otherwise, if a set is small, i.e., its covers less than \( |U|/k \) uncovered elements of \( L \), the algorithm stores the set \( S \) in memory. Fortunately, it is enough to store its projection over the sampled elements explicitly (i.e., \( S \cap L \)) – this requires remembering only the \( O(|Z|/k) \) indices of the elements of \( S \cap L \).

In order to show that a solution of the Set Cover problem over the sampled elements is a good cover of the initial Set Cover instance, we apply the \textit{relative \((p, \varepsilon)\)-approximation} sampling result of [HS11] (see 7.2.4) and it is enough for \( Z \) to be of size \( \tilde{O}(\rho kn^\delta) \). Using \textit{relative \((p, \varepsilon)\)-approximation} sampling, we show that after two passes the number of uncovered elements is reduced by a factor of \( n^\delta \). Note that the relative
$(p, ε)$-approximation sampling improves over the Element Sampling technique used in [DIMV14] with respect to the number of passes.

Since in each iteration we pick $O(ρk)$ sets and the number of uncovered elements decreases by a factor of $n^δ$, after $1/δ$ iterations the algorithm picks $O(ρk/δ)$ sets and covers all elements. Moreover, the memory space of the whole algorithm is $\tilde{O}(ρmn^δ)$ (see 7.2.2).

### 7.2.2 Analysis

In the rest of this section we prove that the iterSetCover algorithm with high probability returns a $O(ρ/δ)$-approximate solution of Set Cover($U, F$) in $2/δ$ passes using $\tilde{O}(mn^δ)$ memory space.

**Lemma 7.2.1.** The number of passes the iterSetCover algorithm makes is $2/δ$.

*Proof.* In each of the $1/δ$ iterations of the iterSetCover algorithm, the algorithm makes two passes. In the first pass, based on the set of sampled elements $Z$, it decides whether to pick a set or keep its projection over $Z$ (i.e., $S \cap L$) in the memory. Then the algorithm calls algOfflineSC which does not require any passes over $F$. The second pass is for computing the set of uncovered elements at the end of the iteration. We need this pass because we only know the projection of the sets we picked in the current iteration over $Z$ and not over the original set of uncovered elements. Thus, in total we make $2/δ$ passes. Also note that for different guesses for the value of $k$, we run the algorithm in parallel and hence the total number of passes remains $2/δ$.

**Lemma 7.2.2.** The memory space used by the iterSetCover algorithm is $\tilde{O}(mn^δ)$.

*Proof.* In each iteration of the algorithm, it picks during the first pass at most $m$ sets (more precisely at most $k$ sets) which requires $O(m \log m)$ memory. Moreover, in the first pass we keep the projection of the sets whose projection over the uncovered sampled elements has size at most $|Z|/k$. Since there are at most $m$ such sets, the total required space for storing the projections is bounded by $O(ρmn^δ \log m \log n)$.

Since in the second pass the algorithm only updates the set of uncovered elements, the amount of space required in the second pass is $O(n)$. Thus, the total required
space to perform each iteration of the iterSetCover algorithm is \( \tilde{O}(mn^\delta) \). Moreover, note that the algorithm does not need to keep the memory space used by the earlier iterations; thus, the total space consumed by the algorithm is \( \tilde{O}(mn^\delta) \).

Next we show the sets we picked before calling algOfflineSC has large size on \( \mathcal{U} \).

**Lemma 7.2.3.** With probability at least \( 1 - m^{-c} \) all sets that pass the “Size Test” in the iterSetCover algorithm have size at least \( |\mathcal{U}|/ck \).

**Proof.** Let \( S \) be a set of size less than \( |\mathcal{U}|/ck \). In expectation, \( |S \cap Z| \) is less than \( (|\mathcal{U}|/ck) \cdot (|Z|/|\mathcal{U}|) = \rho n^\delta m \log n \). By Chernoff bound for large enough \( c \),

\[
\Pr(|S \cap Z| \geq cpn^\delta \log m \log n) \leq m^{-(c+1)}.
\]

Applying the union bound, with probability at least \( 1 - m^{-c} \), all sets passing “Size Test” have size at least \( |\mathcal{U}|/(ck) \).

In what follows we define the relative \((p, \varepsilon)\)-approximation sample of a set system and mention the result of Har-Peled and Sharir [HS11] on the minimum required number of sampled elements to get a relative \((p, \varepsilon)\)-approximation of the given set system.

**Definition 7.2.4.** Let \((\mathcal{V}, \mathcal{H})\) be a set system, i.e., \( \mathcal{V} \) is a set of elements and \( \mathcal{H} \subseteq 2^\mathcal{V} \) is a family of subsets of the ground set \( \mathcal{V} \). For given parameters \( 0 < \varepsilon, p < 1 \), a subset \( Z \subseteq \mathcal{V} \) is a relative \((p, \varepsilon)\)-approximation for \((\mathcal{V}, \mathcal{H})\), if for each \( S \in \mathcal{H} \), we have that

\[
(1 - \varepsilon) \frac{|S|}{|\mathcal{V}|} \leq \frac{|S \cap Z|}{|Z|} \leq (1 + \varepsilon) \frac{|S|}{|\mathcal{V}|}.
\]

If the range is light (i.e., \( |S| < p|\mathcal{V}| \)) then it is required that

\[
\frac{|S|}{|\mathcal{V}|} - \varepsilon p \leq \frac{|S \cap Z|}{|Z|} \leq \frac{|S|}{|\mathcal{V}|} + \varepsilon p.
\]

Namely, \( Z \) is \((1 \pm \varepsilon)\)-multiplicative good estimator for the size of ranges that are at least \( p \)-fraction of the ground set.
The following lemma is a simplified variant of a result in Har-Peled and Sharir [HS11] – indeed, a set system with $M$ sets, can have VC dimension at most $\log M$. This simplified form also follows by a somewhat careful but straightforward application of Chernoff’s inequality.

**Lemma 7.2.5.** Let $(U, F)$ be a finite set system, and $p, \varepsilon, q$ be parameters. Then, a random sample of $U$ such that $|U| = \frac{c'}{\varepsilon^2 p} \left( \log |F| \log \frac{1}{p} + \log \frac{1}{q} \right)$, for an absolute constant $c'$ is a relative $(p, \varepsilon)$-approximation, for all ranges in $F$, with probability at least $(1 - q)$.

**Lemma 7.2.6.** Assuming $|\text{opt}| \leq k \leq 2|\text{opt}|$, after any iteration, with probability at least $1 - m^{1-c/4}$ the number of uncovered elements decreases by a factor of $n^\delta$, and this iteration adds $O(\rho|\text{opt}|)$ sets to the output cover.

**Proof.** Let $V \subseteq U$ be the set of uncovered elements at the beginning of the iteration and note that the total number of sets that is picked during the iteration is at most $(1 + \rho)k$ (see 7.2.3). Consider all possible such covers, that is $G = \{ F' \subseteq F | |F'| \leq (1 + \rho)k \}$, and observe that $|G| \leq m^{(1+\rho)k}$. Let $H$ be the collection that contains all possible sets of uncovered elements at the end of the iteration, defined as $H = \{ V \setminus \bigcup_{S \subseteq C} S | C \in G \}$. Moreover, set $p = 2/n^\delta$, $\varepsilon = 1/2$ and $q = m^{-c}$ and note that $|H| \leq |G| \leq m^{(1+\rho)k}$. Since $\frac{c'}{\varepsilon^2 p} (\log |H| \log \frac{1}{p} + \log \frac{1}{q}) \leq c' p n^\delta \log m \log n = |Z|$ for large enough $c$, by 7.2.5, $Z$ is a relative $(p, \varepsilon)$-approximation of $(V, H)$ with $(1 - q)$ probability. Let $D \subseteq F$ be the collection of sets picked during the iteration which covers all elements in $Z$. Since $Z$ is a relative $(p, \varepsilon)$-approximation sample of $(V, H)$ with probability at least $1 - m^{-c}$, the number of uncovered elements of $V$ (or $U$) by $D$ is at most $\varepsilon p |V| = |U|/n^\delta$.

Hence, in each iteration we pick $O(\rho k)$ sets and at the end of iteration the number of uncovered elements reduces by $n^\delta$. \hfill \square

**Lemma 7.2.7.** The iterSetCover algorithm computes a set cover of $(U, F)$, whose size is within a $O(\rho/\delta)$ factor of the size of an optimal cover with probability at least $1 - m^{1-c/4}$. 

170
Proof. Consider the run of iterSetCover for which the value of $k$ is between $|{\text{opt}}|$ and $2|{\text{opt}}|$. In each of the $(1/\delta)$ iterations made by the algorithm, by 7.2.6, the number of uncovered elements decreases by a factor of $n^\delta$ where $n$ is the number of initial elements to be covered by the sets. Moreover, the number of sets picked in each iteration is $O(\rho k)$. Thus after $(1/\delta)$ iterations, all elements would be covered and the total number of sets in the solution is $O(\rho|{\text{opt}}|/\delta)$. Moreover by 7.2.6, the success probability of all the iterations, is at least $1 - \frac{1}{\delta m^{c/4}} \geq 1 - (1/m)^{\frac{c}{4}}$.

Theorem 7.2.8. The iterSetCover($U, F, \delta$) algorithm makes $2/\delta$ passes, uses $\tilde{O}(mn^\delta)$ memory space, and finds a $O(\rho/\delta)$-approximate solution of the SetCover problem with high probability.

Furthermore, given enough number of passes the iterSetCover algorithm matches the known lower bound on the memory space of the streaming SetCover problem up to a polylog($m$) factor where $m$ is the number of sets in the input.

Proof. The first part of the proof implied by 7.2.1, 7.2.2, and 7.2.7.

As for the lower bound, note that by a result of Nisan [Nis02], any randomized $(\log n/2)$-approximation protocol for Set Cover($U, F$) in the one-way communication model requires $\Omega(m)$ bits of communication, no matter how many number of rounds it makes. This implies that any randomized $O(\log n)$-pass, $(\log n/2)$-approximation algorithm for Set Cover($U, F$) requires $\tilde{O}(m)$ space, even under the exponential computational power assumption.

By the above, the iterSetCover algorithm makes $O(1/\delta)$ passes and uses $\tilde{O}(mn^\delta)$ space to return a $O(1/\delta)$-approximate solution under the exponential computational power assumption ($\rho = 1$). Thus by letting $\delta = c/\log n$, we will have a $(\log n/2)$-approximation streaming algorithm using $\tilde{O}(m)$ space which is optimal up to a factor of polylog($m$).

7.2.8 provides a strong indication that our trade-off algorithm is optimal.
7.3 Lower Bound for Single Pass Algorithms

In this section, we study the Set Cover problem in the two-party communication model and give a tight lower bound on the communication complexity of the randomized protocols solving the problem in a single round. In the two-party Set Cover, we are given a set of elements $U$ and there are two players Alice and Bob where each of them has a collection of subsets of $U$, $F_A$ and $F_B$. The goal for them is to find a minimum size cover $C \subseteq F_A \cup F_B$ covering $U$ while communicating the fewest number of bits from Alice to Bob (In this model Alice communicates to Bob and then Bob should report a solution). Our main lower bound result for the single pass protocols for Set Cover is the following theorem which implies that the naive approach in which one party sends all of its sets to the the other one is optimal.

**Theorem 7.3.1.** Any single round randomized protocol that approximates Set Cover($U, F$) within a factor better than $3/2$ and error probability $O(m^{-c})$ requires $\Omega(mn)$ bits of communication where $n = |U|$ and $m = |F|$ and $c$ is a sufficiently large constant.

We consider the case in which the parties want to decide whether there exists a cover of size 2 for $U$ in $F_A \cup F_B$ or not. If any of the parties has a cover of size at most 2 for $U$, then it becomes trivial. Thus the question is whether there exist $S_a \in F_A$ and $S_b \in F_B$ such that $U \subseteq S_a \cup S_b$.

A key observation is that to decide whether there exist $S_a \in F_A$ and $S_b \in F_B$ such that $U \subseteq S_a \cup S_a$, one can instead check whether there exists $S_a \in F_A$ and $S_b \in F_B$ such that $S_a \cap S_b = \emptyset$. In other words we need to solve OR of a series of two-party Set Disjointness problems. In two-party Set Disjointness problem, Alice and Bob are given subsets of $U$, $S_a$ and $S_b$ and the goal is to decide whether $S_a \cap S_b$ is empty or not with the fewest possible bits of communication. Set Disjointness is a well-studied problem in the communication complexity and it has been shown that any randomized protocol for Set Disjointness with $O(1)$ error probability requires $\Omega(n)$ bits of communication where $n = |U|$ [BJKS04, KS92, Raz92].
We can think of the following extensions of the Set Disjointness problem.

- **Many vs One**: In this variant, Alice has $m$ subsets of $U$, $\mathcal{F}_A$ and Bob is given a single set $S_b$. The goal is to determine whether there exists a set $S_a \in \mathcal{F}_A$ such that $S_a \cap S_b = \emptyset$.

- **Many vs Many**: In this variant, each of Alice and Bob are given a collection of subsets of $U$ and the goal for them is to determine whether there exist $S_a \in \mathcal{F}_A$ and $S_b \in \mathcal{F}_B$ such that $S_a \cap S_b = \emptyset$.

Note that deciding whether two-party Set Cover has a cover of size 2 is equivalent to solving the (Many vs Many)-Set Disjointness problem. Moreover, any lower bound for (Many vs One)-Set Disjointness clearly implies the same lower bound for the (Many vs Many)-Set Disjointness problem. In the following theorem we show that any single-round randomized protocol that solves (Many vs One)-Set Disjointness($m,n$) with $O(m^{-c})$ error probability requires $\Omega(mn)$ bits of communication.

**Theorem 7.3.2.** Any randomized protocol for (Many vs One)-Set Disjointness($m,n$) with error probability that is $O(m^{-c})$ requires $\Omega(mn)$ bits of communication if $n \geq c_1 \log m$ where $c$ and $c_1$ are large enough constants.

The idea is to show that if there exists a single-round randomized protocol for the problem with $o(mn)$ bits of communication and error probability $O(m^{-c})$, then with constant probability one can distinguish $\Omega(2^{mn})$ distinct inputs using $o(mn)$ bits which is a contradiction.

Suppose that Alice has a collection of $m$ uniformly and independently random subsets of $U$ (in each of her subsets the probability that $e \in U$ is in the subset is 1/2). Lets assume that there exists a single round protocol $I$ for (Many vs One)-Set Disjointness($n,m$) with error probability $O(m^{-c})$ using $o(mn)$ bits of communication. Let $alg\text{ExistsDisj}$ be Bob's algorithm in protocol $I$. Then we show that one can recover $mn$ random bits with constant probability using $alg\text{ExistsDisj}$ subroutine and the message $s$ sent by the first party in protocol $I$. The $alg\text{RecoverBit}$
Algorithm 17: \texttt{algRecoverBit} uses a protocol for (Many vs One)-Set Disjointness\((m, n)\) to recover Alice’s sets, \(\mathcal{F}_A\) in Bob’s side.

which is shown in 17, is the algorithm to recover random bits using protocol I and \texttt{algExistsDisj}.

To this end, Bob gets the message \(s\) communicated by protocol I from Alice and considers all subsets of size \(c_1 \log m\) and \(c_1 \log m + 1\) of \(U\). Note that \(s\) is communicated only once and thus the same \(s\) is used for all queries that Bob makes. Then at each step Bob picks a random subset \(S_b\) of size \(c_1 \log m\) of \(U\) and solve the (Many vs One)-Set Disjointness problem with input \((\mathcal{F}_A, S_b)\) by running \texttt{algExistsDisj}\((s, S_b)\). Next we show that if \(S_b\) is disjoint from a set in \(\mathcal{F}_A\), then with high probability there is exactly one set in \(\mathcal{F}_A\) which is disjoint from \(S_b\) (see 7.3.3). Thus once Bob finds out that his query, \(S_b\), is disjoint from a set in \(\mathcal{F}_A\), he can query all sets \(S_b^+ \in \{S_b \cup e | e \in U \setminus S_b\}\) and recover the set (or union of sets) in \(\mathcal{F}_A\) that is disjoint from \(S_b\). By a simple pruning step we can detect the ones that are union of more than one set in \(\mathcal{F}_A\) and only keep the sets in \(\mathcal{F}_A\).

In 7.3.6, we show that the number of queries that Bob is required to make to recover \(\mathcal{F}_A\) is \(O(m^c)\) where \(c\) is a constant.

**Lemma 7.3.3.** Let \(S_b\) be a random subset of \(U\) of size \(c \log m\) and let \(\mathcal{F}_A\) be a collection
of \( m \) random subsets of \( U \). The probability that there exists exactly one set in \( F_A \) that is disjoint from \( S_b \) is at least \( \frac{1}{m^{c+1}} \).

**Proof.** The probability that \( S_b \) is disjoint from exactly one set in \( F_A \) is

\[
\Pr(S_b \text{ is disjoint from } \geq 1 \text{ set in } F_A) - \Pr(S_b \text{ is disjoint from } \geq 2 \text{ sets in } F_A) \\
\geq \left( \frac{1}{2} \right)^{c \log m} - \left( \frac{m}{2} \right) \left( \frac{1}{2} \right)^{2c \log m} \geq \frac{1}{m^{c+1}}.
\]

First we prove the first term in the above inequality. For an arbitrary set \( S \in F_A \), since any element is contained in \( S \) with probability \( \frac{1}{2} \), the probability that \( S \) is disjoint from \( S_b \) is \( (1/2)^{c \log m} \).

\[
\Pr(S_b \text{ is disjoint from at least one set in } F_A) \geq 2^{-c \log m}.
\]

Moreover since there exist \( \binom{m}{2} \) pairs of sets in \( F_A \), and for each \( S_1, S_2 \in F_A \), the probability that \( S_1 \) and \( S_2 \) are disjoint from \( S_b \) is \( m^{-2c} \),

\[
\Pr(S_b \text{ is disjoint from at least two sets in } F_A) \leq m^{-(2c-2)}.
\]

\[ \square \]

A family of sets \( \mathcal{M} \) is called *intersecting* if and only if for any sets \( A, B \in \mathcal{M} \) either both \( A \setminus B \) and \( B \setminus A \) are non-empty or both \( A \setminus B \) and \( B \setminus A \) are empty; in other words, there exists no \( A, B \in \mathcal{M} \) such that \( A \subseteq B \). Let \( F_A \) be a collection of subsets of \( U \). We show that with high probability after testing \( O(m^c) \) queries for sufficiently large constant \( c \), the \texttt{algRecoverBit} algorithm recovers \( F_A \) completely if \( F_A \) is intersecting. First we show that with high probability the collection \( F_A \) is intersecting.

**Observation 7.3.4.** Let \( F_A \) be a collection of \( m \) uniformly random subsets of \( U \) where \( |U| \geq c \log m \). With probability at least \( 1 - m^{-c/4+2} \), \( F_A \) is an intersecting family.

**Proof.** The probability that \( S_1 \subseteq S_2 \) is \( \left( \frac{3}{4} \right)^n \) and there are at most \( m(m-1) \) pairs
of sets in $\mathcal{F}_A$. Thus with probability at least $1 - m^2 (\frac{3}{4})^n \geq 1 - 1/m^{c_1-2}$, $\mathcal{F}_A$ is intersecting.

\begin{observation}
The number of distinct inputs of Alice (collections of random subsets of U), that is distinguishable by $\text{algRecoverBit}$ is $\Omega(2^{mn})$.
\end{observation}

\begin{proof}
There are $2^{mn}$ collections of $m$ random subsets of $U$. By 7.3.4, $\Omega(2^{mn})$ of them are intersecting. Since we can only recover the sets in the input collection and not their order, the distinct number of input collection that are distinguished by $\text{algRecoverBit}$ is $\Omega(2^{mn}/m!)$ which is $\Omega(2^{mn})$ for $n \geq c \log m$.
\end{proof}

By 7.3.4 and only considering the case such that $\mathcal{F}_A$ is intersecting, we have the following lemma.

\begin{lemma}
Let $\mathcal{F}_A$ be a collection of $m$ uniformly random subsets of $U$ and suppose that $|U| \geq c \log m$. After testing at most $mc^1 \log m$ queries, with probability at least $1 - \frac{1}{m^{c_1+2}}$, $\mathcal{F}_A$ is fully recovered, where $p$ is the success rate of protocol I for the (Many vs One)-Set Disjointness problem.
\end{lemma}

\begin{proof}
By 7.3.3, for each $S_b \subset U$ of size $c_1 \log m$ the probability that $S_b$ is disjoint from exactly one set in a random collection of sets $\mathcal{F}_A$ is at least $1/m^{c_1+1}$. Given $S_b$ is disjoint from exactly one set in $\mathcal{F}_A$, due to symmetry of the problem, the chance that $S_b$ is disjoint from a specific set $S \in \mathcal{F}_A$ is at least $\frac{1}{m^{c_1+2}}$. After $\alpha m^{c_1+2} \log m$ queries where $\alpha$ is a large enough constant, for any $S \in \mathcal{F}_A$, the probability that there is not a query $S_b$ that is only disjoint from $S$ is at most $(1 - \frac{1}{m^{c_1+2}})^{\alpha m^{c_1+2} \log m} \leq e^{-\alpha \log m} = \frac{1}{m^2}$.

Thus after trying $\alpha m^{c_1+2} \log m$ queries, with probability at least $(1 - \frac{1}{2m^{c_1+2}}) \geq (1 - \frac{1}{m})$, for each $S \in \mathcal{F}_A$ we have at least one query that is only disjoint from $S$ (and not any other sets in $\mathcal{F}_A \setminus S$).

Once we have a query subset $S_b$ which is only disjoint from a single set $S \in \mathcal{F}_A$, we can ask $n - c \log m$ queries of size $c_1 \log m + 1$ and recover $S$. Note that if $S_b$ is disjoint from more than one sets in $\mathcal{F}_A$ simultaneously, the process (asking $n - c \log m$ queries of size $c_1 \log m + 1$) will end up in recovering the union of those sets. Since $\mathcal{F}_A$ is an intersecting family with high probability (7.3.4), by pruning step in the
**algRecoverBit** algorithm we are guaranteed that at the end of the algorithm, what we returned is exactly $F_A$. Moreover the total number of queries the algorithm makes is at most

$$n \times (\alpha m^{c_1 + 2} \log m) \leq \alpha m^{c_1 + 3} \log m \leq m^c$$

for $c \geq c_1 + 4$.

Thus after testing $m^c$ queries, $F_A$ will be recovered with probability at least $(1 - \frac{1}{m})p^{m^c}$ where $p$ is the success probability of the protocol $I$ for (Many vs One)-Set Disjointness($m, n$).

**Corollary 7.3.7.** Let $I$ be a protocol for (Many vs One)-Set Disjointness($m, n$) with error probability $O(m^{-c})$ and $s$ bits of communication such that $n \geq c \log m$ for large enough $c$. Then **algRecoverBit** recovers $F_A$ with constant success probability using $s$ bits of communication.

By 7.3.5, since **algRecoverBit** distinguishes $\Omega(2^{mn})$ distinct inputs with constant probability of success (by 7.3.7), the size of message sent by Alice, should be $\Omega(mn)$. This proves 7.3.2.

**Proof of 7.3.1:** As we showed earlier, the communication complexity of (Many vs One)-Set Disjointness is a lower bound for the communication complexity of Set Cover. 7.3.2 showed that any protocol for (Many vs One)-Set Disjointness($n, |F_A|$) with error probability less than $O(m^{-c})$ requires $\Omega(mn)$ bits of communication. Thus any single-round randomized protocol for Set Cover with error probability $O(m^{-c})$ requires $\Omega(mn)$ bits of communication.

Since any $p$-pass streaming $\alpha$-approximation algorithm for problem $P$ that uses $O(s)$ memory space, is a $p$-round two-party $\alpha$-approximation protocol for problem $P$ using $O(sp)$ bits of communication [GM08], and by 7.3.1, we have the following lower bound for Set Cover problem in the streaming model.
Theorem 7.3.8. Any single-pass randomized streaming algorithm for Set Cover\((U, \mathcal{F})\) that computes a \((3/2)\)-approximate solution with probability \(\Omega(1 - m^{-c})\) requires \(\Omega(mn)\) memory space (assuming \(n \geq c_1 \log m\)).

7.4 Geometric Set Cover

In this section, we consider the streaming Set Cover problem in the geometric settings. We present an algorithm for the case where the elements are a set of \(n\) points in the plane \(\mathbb{R}^2\) and the \(m\) sets are either all disks, all axis-parallel rectangles, or all \(\alpha\)-fat triangles (which for simplicity we call shapes) given in a data stream. As before, the goal is to find the minimum size cover of points from the given sets. We call this problem the Points-Shapes Set Cover problem.

Note that, the description of each shape requires \(O(1)\) space and thus the Points-Shapes Set Cover problem is trivial to be solved in \(O(m + n)\) space. In this setting the goal is to design an algorithm whose space is sub-linear in \(O(m + n)\). Here we show that almost the same algorithm as iterSetCover (with slight modifications) uses \(O(n)\) space to find an \(O(\rho)\)-approximate solution of the Points-Shapes Set Cover problem in constant passes.

7.4.1 Preliminaries

A triangle \(\triangle\) is called \(\alpha\)-fat (or simply fat) if the ratio between its longest edge and its height on this edge is bounded by a constant \(\alpha > 1\) (there are several equivalent definitions of \(\alpha\)-fat triangles).

Definition 7.4.1. Let \((U, \mathcal{F})\) be a set system such that \(U\) is a set of points and \(\mathcal{F}\) is a collection of shapes, in the plane \(\mathbb{R}^2\). The canonical representation of \((U, \mathcal{F})\) is a collection \(\mathcal{F}'\) of regions such that the following conditions hold. First, each \(S' \in \mathcal{F}'\) has \(O(1)\) description. Second, for each \(S' \in \mathcal{F}'\), there exists \(S \in \mathcal{F}\) such that \(S' \cap U \subseteq S \cap U\). Finally, for each \(S \in \mathcal{F}\), there exists \(c_1\) sets \(S_1', \ldots, S_{c_1}' \in \mathcal{F}'\) such that \(S \cap U = (S_1' \cup \cdots \cup S_{c_1}') \cap U\) for some constant \(c_1\).
The following two results are from [EHR12] which are the formalization of the ideas in [AES10].

**Lemma 7.4.2.** (Lemma 4.18 in [EHR12]) Given a set of points \( U \) in the plane \( \mathbb{R}^2 \) and a parameter \( w \), one can compute a set \( F'_{\text{total}} \) of \( O(|U|w^2 \log |U|) \) axis-parallel rectangles with the following property. For an arbitrary axis-parallel rectangle \( S \) that contains at most \( w \) points of \( U \), there exist two axis-parallel rectangles \( S_1', S_2' \in F'_{\text{total}} \) whose union has the same intersection with \( U \) as \( S \), i.e., \( S \cap U = (S_1' \cup S_2') \cap U \).

**Lemma 7.4.3.** (Theorem 5.6 in [EHR12]) Given a set of points \( U \) in \( \mathbb{R}^2 \), a parameter \( w \) and a constant \( \alpha \), one can compute a set \( F'_{\text{total}} \) of \( O(|U|w^3 \log^2 |U|) \) regions each having \( O(1) \) description with the following property. For an arbitrary \( \alpha \)-fat triangle \( S \) that contains at most \( w \) points of \( U \), there exist nine regions from \( F'_{\text{total}} \) whose union has the same intersection with \( U \) as \( S \).

Using the above lemmas we get the following lemma.

**Lemma 7.4.4.** Let \( U \) be a set of points in \( \mathbb{R}^2 \) and let \( F \) be a set of shapes (discs, axis-parallel rectangles or fat triangles), such that each set in \( F \) contains at most \( w \) points of \( U \). Then, in a single pass over the stream of sets \( F \), one can compute the canonical representation \( F' \) of \((U, F)\). Moreover, the size of the canonical representation is at most \( O(|U|w^2 \log |U|) \) and the space requirement of the algorithm is \( \tilde{O}(|F'|) = \tilde{O}(|U|w^3) \).

**Proof.** For the case of axis-parallel rectangles and fat triangles, first we use 7.4.2 and 7.4.3 to get the set \( F'_{\text{total}} \) offline which require \( \tilde{O}(F'_{\text{total}}) = \tilde{O}(|U|w^3 \log^2 |U|) \) memory space. Then by making one pass over the stream of sets \( F \), we can find the canonical representation \( F' \) by picking all the sets \( S' \in F'_{\text{total}} \) such that \( S' \cap U \subseteq S \cap U \) for some \( S \in F \). For discs however, we just make one pass over the sets \( F \) and keep a maximal subset \( F' \subseteq F \) such that for each pair of sets \( S'_1, S'_2 \in F' \) their projection on \( U \) are different, i.e., \( S'_1 \cap U \neq S'_2 \cap U \). By a standard technique of Clarkson and Shor [CS89], it can be proved that the size of the canonical representation, i.e., \( |S'| \), is bounded by \( O(|U|w^2) \). Note that this is just counting the number of discs that contain at most \( w \) points, namely the at most \( w \)-level discs. \( \square \)
7.4.2 Algorithm

The outline of the Points-Shapes-Set-Cover algorithm (shown in 18) is very similar to the iterSetCover algorithm presented earlier in 7.2.

In the first pass, the algorithm picks all the sets that cover a large number of yet-uncovered elements. Next, we sample \( Z \). Since we have removed all the ranges that have large size, in the first pass, the size of the remaining ranges restricted to the sample \( Z \) is small. Therefore by 7.4.4, the canonical representation of \((Z, F_Z)\) has small size and we can afford to store it in the memory. We use 7.4.4 to compute the canonical representation \( F_Z \) in one pass. The algorithm then uses the sets in \( F_Z \) to find a cover \( \text{sol}_Z \) for the points of \( Z \). Next, in one additional pass, the algorithm replaces each set in \( \text{sol}_Z \) by one of its supersets in \( F \).

Finally, note that in the algorithm of 7.2, we are assuming that the size of the optimal solution is \( O(k) \). Thus it is enough to stop the iterations once the number of uncovered elements is less than \( k \). Then we can pick an arbitrary set for each of the uncovered elements. This would add only \( k \) more sets to the solution. Using this idea, we can reduce the size of the sampled elements down to \( c pk(A/k)^\delta \log m \log n \) which would help us in getting near-linear space in the geometric setting. Note that the final pass of the algorithm can be embedded into the previous passes but for the sake of clarity we write it separately.

7.4.3 Analysis

By a similar approach to what we used in 7.2 to analyze the pass count and approximation guarantee of iterSetCover algorithm, we can show that the number of passes of the algGeomSC algorithm is \( 3/\delta + 1 \) (which can be reduced to \( 3/\delta \) with minor changes), and the algorithm returns an \( O(p/\delta) \)-approximate solution. Next, we analyze the space usage and the correctness of the algorithm. Note that our analysis in this section only works for \( \delta \leq 1/4 \).

**Lemma 7.4.5.** The algorithm uses \( \tilde{O}(n) \) space.
algGeomSC(U, F, δ):

for k ∈ {2^i | 0 ≤ i ≤ log n} do in parallel: ▷ // n = |U|

Let L ← U and sol ← ∅

Repeat 1/δ times:

for S ∈ F do ▷ // Pass

if |S ∩ L| ≥ |U|/k then

sol ← sol ∪ {S}

L ← L \ S

Z ← sample of L of size \(cρk(\frac{n}{k})^δ \log m \log n\)

\(F_Z ← \text{compCanonicalRep}(Z, F, \frac{|Z|}{k})\) ▷ // Pass

sol_Z ← algOfflineSC(Z, F_Z)

for S ∈ F do ▷ // Final Pass

if S ∩ L = ∅ then

sol ← sol ∪ {S}

L ← L \ S

return smallest sol computed in parallel

Algorithm 18: A streaming algorithm for Points-Shapes Set Cover problem.

Proof. Consider an iteration of the algorithm. The memory space used in the first pass of each iteration is \(\tilde{O}(n)\). The size of Z is \(cρk(\frac{n}{k})^δ \log m \log n\) and after the first pass the size of each set is at most \(|U|/k\). Thus using Chernoff bound for each set \(S ∈ F \setminus \text{sol},\)

\[
\Pr \left[ |S ∩ Z| > (1 + 2) \frac{|U|}{k} \times \frac{|Z|}{|U|} \right] \leq \exp \left( -\frac{4|Z|}{3k} \right) \leq \left( \frac{1}{m} \right)^{c+1}.
\]

Thus, with probability at least 1 – \(m^{-c}\) (by the union bound), all the sets that are not picked in the first pass, cover at most \(3|Z|/k = cρ(\frac{n}{k})^δ \log m \log n\) elements of Z. Therefore, we can use 7.4.4 to show that the number of sets in the canonical representation of \((Z, F_Z)\) is at most

\[O(|Z| \left( \frac{3|Z|}{k} \right)^3 \log^2 |Z|) = O(\rho^4 n \log^4 m \log^6 n),\]
as long as \( \delta \leq 1/4 \). To store each set in a canonical representation of \((Z, F)\) only constant space is required. Moreover, by 7.4.4, the space requirement of the second pass is \( \tilde{O}(|F_Z|) = \tilde{O}(n) \). Therefore, the total required space is \( \tilde{O}(n) \) and the lemma follows.

**Theorem 7.4.6.** Given a set system defined over a set \( U \) of \( n \) points in the plane, and a set of \( m \) ranges \( F \) (which are either all disks, axis-parallel rectangles, or fat triangles). Let \( \rho \) be the quality of approximation to the offline set-cover solver we have, and let \( 0 < \delta < 1/4 \) be an arbitrary parameter.

Setting \( \delta = 1/4 \), the algorithm \texttt{algGeomSC}, depicted in 18, with high probability, returns an \( O(\rho) \)-approximate solution of the optimal set cover solution for the instance \((U, F)\). This algorithm uses \( \tilde{O}(n) \) space, and performs constant passes over the data.

**Proof.** As before consider the run of the algorithm in which \(|\text{opt}| \leq k < 2|\text{opt}|\). Let \( V \) be the set of uncovered elements \( L \) at the beginning of the iteration and note that the total number of sets that is picked during the iteration is at most \((1 + c_1 \rho)k\) where \( c_1 \) is the constant defined in 7.4.1. Let \( G \) denote all possible such covers, that is \( G = \{F' \subseteq F \mid |F'| \leq (1 + c_1 \rho)k\} \). Let \( H \) be the collection that contains all possible set of uncovered elements at the end of the iteration, defined as \( H = \{V \setminus \bigcup_{S \in C} S \mid C \in G\} \). Set \( p = (k/n)^\delta \), \( \varepsilon = 1/2 \) and \( q = m^{-c} \). Since for large enough \( c \), \( \frac{m}{2p}(\log |H| \log \frac{1}{p} + \log \frac{1}{q}) \leq c\rho k(n/k)^\delta \log m \log n = |Z| \) with probability at least \( 1 - m^{-c} \), by 7.2.5, the set of sampled elements \( Z \) is a relative \((p, \varepsilon)\)-approximation sample of \((V, H)\).

Let \( C \subseteq F \) be the collection of sets picked in the third pass of the algorithm that covers all elements in \( Z \). By 7.4.4, \(|C| \leq c_1 \rho k\) for some constant \( c_1 \). Since with high probability \( Z \) is a relative \((p, \varepsilon)\)-approximation sample of \((V, H)\), the number of uncovered elements of \( V \) (or \( L \)) after adding \( C \) to sol is at most \( \varepsilon p |V| \leq |U|(k/n)^\delta \). Thus with probability at least \((1 - m^{-c})\), in each iteration and by adding \( O(\rho k) \) sets, the number of uncovered elements reduces by a factor of \((n/k)^\delta\).

Therefore, after 4 iterations (for \( \delta = 1/4 \)) the algorithm picks \( O(\rho k) \) sets and with high probability the number of uncovered elements is at most \( n(k/n)^{\delta/\delta} = k \). Thus,
in the final pass the algorithm only adds \( k \) sets to the solution \( \text{sol} \), and hence the approximation factor of the algorithm is \( O(\rho) \).

**Remark 7.4.7.** The result of 7.4.6 is similar to the result of Agarwal and Pan [AP14] – except that their algorithm performs \( O(\log n) \) iterations over the data, while the algorithm of 7.4.6 performs only a constant number of iterations. In particular, one can use the algorithm of Agarwal and Pan [AP14] as the offline solver.

### 7.5 Lower Bound for Multipass Algorithms

In this section, we give lower bound on the memory space of multipass streaming algorithms for the Set Cover problem. Our main result is \( \Omega(mn^\delta) \) space for streaming algorithms that return an optimal solution of the Set Cover problem in \( O(1/\delta) \) passes for \( m = O(n) \). Our approach is to reduce the communication Intersection Set Chasing\((n,p)\) problem introduced by Guruswami and Onak [GO13] to the communication Set Cover problem.

Consider a communication problem \( \mathcal{P} \) with \( n \) players \( P_1, \ldots, P_n \). The problem \( \mathcal{P} \) is a \((n,r)\)-communication problem if players communicate in \( r \) rounds and in each round they speak in order \( P_1, \ldots, P_n \). At the end of the \( r \)round \( P_n \) should return the solution. Moreover, we assume private randomness and public messages. In what follows, we define the communication Set Chasing and Intersection Set Chasing problems.

**Definition 7.5.1** (Communication Set Chasing Problem). The Set Chasing\((n,p)\) problem is a \((p,p-1)\) communication problem in which the player \( i \) has a function \( f_i : [n] \rightarrow 2^{[n]} \) and the goal is to compute \( \tilde{f}_i(\tilde{f}_2(\cdots \tilde{f}_p(\{1\})\cdots)) \) where \( \tilde{f}_i(S) = \bigcup_{s \in S} f_i(s) \). 7-2(a) shows an instance of the communication Set Chasing \((4,3)\).

**Definition 7.5.2** (Communication Intersection Set Chasing). The Intersection Set Chasing\((n,p)\) is a \((2p,p-1)\) communication problem in which the first \( p \) players have an instance of the Set Chasing\((n,p)\) problem and the other \( p \) players have another instance of the Set Chasing\((n,p)\) problem. The output of the Intersection Set
Chasing\((n,p)\) is 1 if the solutions of the two instances of the Set Chasing\((n,p)\) intersect and 0 otherwise. 7-2(b) shows an instance of the Intersection Set Chasing \((4,3)\). The function \(f_i\) of each player \(P_i\) is specified by a set of directed edges form a copy of vertices labeled \(\{1, \cdots, n\}\) to another copy of vertices labeled \(\{1, \cdots, n\}\).

The communication Set Chasing problem is a generalization of the well-known communication Pointer Chasing problem in which player \(i\) has a function \(f_i : [n] \to [n]\) and the goal is to compute \(f_1(f_2(\cdots f_p(1) \cdots))\). [GO13] showed that any randomized protocol that solves Intersection Set Chasing\((n,p)\) with error probability less than 1/10, requires \(\Omega\left(\frac{n^{1+1/(2p)}}{p^{16\log^{3/2} n}}\right)\) bits of communication where \(n\) is sufficiently large and \(p \leq \frac{\log n}{\log \log n}\). In 7.5.4, we reduce the communication Intersection Set Chasing problem to the communication Set Cover problem and then give the first superlinear memory lower bound for the streaming Set Cover problem.

**Definition 7.5.3** (Communication Set Cover\((U, F, p)\) Problem). The communication Set Cover\((n,p)\) is a \((p, p-1)\) communication problem in which a collection of elements \(U\) is given to all players and each player \(i\) has a collection of subsets of \(U, F_i\). The goal is to solve Set Cover\((U, F_1 \cup \cdots \cup F_p)\) using the minimum number of communication bits.

**Theorem 7.5.4.** Any \((1/2\delta - 1)\) passes streaming algorithm that solves the Set Cover\((U, F)\) optimally with constant probability of error requires \(\widetilde{\Omega}(mn^{\delta})\) memory space where \(\delta \geq \frac{\log \log n}{\log n}\) and \(m = O(n)\).

Consider an instance ISC of the communication Intersection Set Chasing\((n,p)\).
We construct an instance of the communication Set Cover problem such that solving Set Cover optimally determines whether the output of ISC is 1 or not.

The instance ISC consists of 2p players. Each player 1, ⋯, p has a function $f_i : [n] \rightarrow 2^{[n]}$ and each player $p+1, \cdots, 2p$ has a function $f'_i : [n] \rightarrow 2^{[n]}$ (see 7-2). In ISC, each function $f_i$ is shown by a set of vertices $v_i^1, \cdots, v_i^n$ and $v_{i+1}^1, \cdots, v_{i+1}^n$ such that there is a directed edge from $v_{i+1}^j$ to $v_i^j$ if and only if $\ell \in f_i(j)$. Similarly, each function $f'_i$ is denoted by a set of vertices $u_i^1, \cdots, u_i^n$ and $u_{i+1}^1, \cdots, u_{i+1}^n$ such that there is a directed edge from $u_{i+1}^j$ to $u_i^j$ if and only if $\ell \in f'_i(j)$ (see 7-3(a) and 7-3(b)).

In the corresponding communication Set Cover instance of ISC, we add two elements $\text{in}(v_i^j)$ and $\text{out}(v_i^j)$ per each vertex $v_i^j$ where $i \leq p+1$, $j \leq n$. We also add two elements $\text{in}(u_i^j)$ and $\text{out}(u_i^j)$ per each vertex $u_i^j$ where $i \leq p+1$, $j \leq n$. In addition to these elements, for each player $i$, we add an element $e_i$ (see 7-3(c) and 7-3(d)).

Next, we define a collection of sets in the corresponding Set Cover instance of
ISC. For each player $P_i$, where $1 \leq i \leq p$, we add a single set $S_i^j$ containing $\text{out}(v_i^j)$ and $\text{in}(v_i^j)$ for all out-going edges $(v_i^j, v_{i+1}^j)$. Moreover, all $S_i^j$ sets contain the element $e_i$. Next, for each vertex $v_i^j$ we add a set $R_i^j$ that contains the two corresponding elements of $v_i^j$, $\text{in}(v_i^j)$ and $\text{out}(v_i^j)$. In 7-3(c), the red rectangles denote $R$-type sets and the curves denote $S$-type sets for the first half of the players.

Similarly to the sets corresponding to players 1 to $p$, for each player $P_{p+i}$ where $1 \leq i \leq p$, we add a set $S_{p+i}^j$ containing $\text{in}(u_i^j)$ and $\text{out}(u_i^j)$ for all in-coming edges $(u_i^j, u_{i+1}^j)$ of $u_i^j$ (denoting $f_i^{j-1}(j)$). The set $S_{p+i}^j$ contains the element $e_{p+i}$ too. Next, for each vertex $u_i^j$ we add a set $T_{p+i}^j$ that contains the two corresponding elements of $u_i^j$, $\text{in}(u_i^j)$ and $\text{out}(u_i^j)$. In 7-3(d), the red rectangles denote $T$-type sets and the curves denote $S$-type sets for the second half of the players.

At the end, we merge $v_i^j$s and $u_i^j$s as shown in 7-4. After merging the corresponding sets of $v_i^j$s ($R_1^1, \ldots, R_p^1$) and the corresponding sets of $u_i^j$s ($T_1^1, \ldots, T_p^1$), we call the merged sets $T_1^1, \ldots, T_p^1$.

![Figure 7-4](image_url)

**Figure 7-4:** In (a) and (b) two Set Chasing instances merge in their first set of vertices and (c) shows the corresponding gadgets of these merged vertices in the communication SetCover.

The main claim is that if the solution of ISC is 1 then the size of an optimal solution of its corresponding SetCover instance SC is $(2p + 1)n + 1$; otherwise, it is $(2p + 1)n + 2$.

**Lemma 7.5.5.** The size of any feasible solution of SC is at least $(2p + 1)n + 1$.

**Proof.** For each player $i$ ($1 \leq i \leq p$), since $\text{out}(u_i^j)$ are only covered by $R_i^j$ and $S_i^j$, at least $n$ sets are required to cover $\text{out}(v_i^1), \ldots, \text{out}(v_i^n)$. Moreover for player
Figure 7-5: In (a), path $Q$ is shown with black dashed arcs and (b) shows the corresponding cover of path $Q$.

$P_p$, since $\text{in}(v_{p+1}^j)$s are only covered by $R_{p+1}^j$ and $e_p$ is only covered by $S_p^1$, all $n + 1$ sets $R_{p+1}^1, \ldots, R_{p+1}^n, S_p^1$ must be selected in any feasible solution of SC.

Similarly for each player $p+i$ ($1 \leq i \leq p$), since $\text{in}(u_i^j)$s are only covered by $T_{ij}^j$ and $S_{p+i}^j$, at least $n$ sets are required to cover $\text{in}(u_1^1), \ldots, \text{in}(u_p^p)$. Moreover, considering $u_{p+1}^1, \ldots, u_{p+1}^n$, since $\text{in}(u_{p+1}^j)$ is only covered by $T_{p+1}^j$, all $n$ sets $T_{p+1}^1, \ldots, T_{p+1}^n$ must be selected in any feasible solution of SC.

All together, at least $(2p + 1)n + 1$ sets should be selected in any feasible solution of SC.

Lemma 7.5.6. Suppose that the solution of ISC is 1. Then the size of an optimal solution of its corresponding Set Cover instance is exactly $(2p + 1)n + 1$.

Proof. By 7.5.5, the size of an optimal solution of S is at least $(2p + 1)n + 1$. Here we prove that $(2p + 1)n + 1$ sets suffice when the solution of ISC is 1. Let $Q = v_{p+1}^1, v_p^j, \ldots, v_2^j, v_1^j, u_1^1, u_2^1, \ldots, u_p^1, u_{p+1}^1$ be a path in ISC such that $j_1 = \ell_1$ (since the solution of ISC is 1 such a path exists). The corresponding solution to $Q$ can be constructed as follows (See 7-5):

- Pick $S_p^1$ and all $R_{p+1}^j$s ($n + 1$ sets).
- For each $v_i^j$ in $Q$ where $1 < i \leq p$, pick the set $S_{i-1}^{j_i}$ in the solution. Moreover, for each such $i$ pick all sets $R_i^j$ where $j \neq j_i$ ($n(p - 1)$ sets).
- For $u_1^j$ (or $u_1^\ell_1$), pick the set $S_{p+1}^{j_1}$. Moreover, pick all sets $T_{1}^j$ where $j \neq j_1$ ($n$ sets).
For each \( u_i^\ell \) in \( Q \) where \( 1 < i \leq p \), pick the set \( S_{p+i}^\ell \) in the solution. Moreover, for each such \( i \) pick all sets \( T_i^\ell \) where \( \ell \neq \ell_i \) (\( n(p-1) \) sets).

Pick all \( T_{p+1}^j \)s (\( n \) sets).

It is straightforward to see that the solution constructed above is a feasible solution.

\[ \square \]

**Lemma 7.5.7.** Suppose that the size of an optimal solution of the corresponding SetCover instance of ISC, SC, is \((2p+1)n + 1\). Then the solution of ISC is 1.

**Proof.** As we proved earlier in 7.5.5, any feasible solution of SC picks \( R_{p+1}^1, \ldots, R_p^1, S_1^1 \) and \( T_{p+1}^1, \ldots, T_p^1 \). Moreover, we proved that for each \( 1 \leq i < p \), at least \( n \) sets should be selected from \( R_{i+1}^1, \ldots, R_{i+1}^n, S_i^1, \ldots, S_i^n \). Similarly, for each \( 1 \leq i \leq p \), at least \( n \) sets should be selected from \( T_{i}^1, \ldots, T_{i}^n, S_{i+1}^1, \ldots, S_{i+1}^n \). Thus if a feasible solution of SC, opt, is of size \((2p+1)n + 1\), it has exactly \( n \) sets from each specified group.

Next we consider the first half of the players and second half of the players separately. Consider \( i \) such that \( 1 \leq i < p \). Let \( S_i^1, \ldots, S_i^k \) be the sets picked in the optimal solution (because of \( e_i \) there should be at least one set of form \( S_i^j \) in opt). Since each \( \text{out}(v_{i+1}^j) \) is only covered by \( S_i^j \) and \( R_{i+1}^j \), for all \( j \not\in \{j_1, \ldots, j_k\} \), \( R_{i+1}^j \) should be selected in opt. Moreover, for all \( j \in \{j_1, \ldots, j_k\} \), \( R_{i+1}^j \) should not be contained in opt (otherwise the size of opt would be larger than \((2p+1)n + 1\)). Consider \( j \in \{j_1, \ldots, j_k\} \). Since \( R_{i+1}^j \) is not in opt, there should be a set \( S_{i+1}^\ell \) selected in opt such that \( \text{in}(v_{i+1}^j) \) is contained in \( S_{i+1}^\ell \). Thus by considering \( S_i \)s in a decreasing order and using induction, if \( S_i^j \) is in opt then \( v_{i+1}^j \) is reachable form \( v_{p+1}^1 \).

Next consider a set \( S_{p+i}^j \) that is selected in opt (\( 1 \leq i \leq p \)). By similar argument, \( T_i^j \) is not in opt and there exists a set \( S_{p+i-1}^j \) (or \( S_{i}^j \) if \( i = 1 \)) in opt such that \( \text{out}(u_i^j) \) is contained in \( S_{p+i-1}^j \). Let \( u_{i+1}^\ell, \ldots, u_{i+1}^{\ell_k} \) be the set of vertices whose corresponding out elements are in \( S_{p+i}^j \). Then by induction, there exists an index \( r \) such that \( u_r^j \) is reachable from \( v_{p+1}^1 \) and \( u_r^j \) is also reachable from all \( u_{i+1}^\ell, \ldots, u_{i+1}^{\ell_k} \). Moreover, the way we constructed the instance SC guarantees that all sets \( S_{2p}^1, \ldots, S_{2p}^n \) contains \( \text{out}(u_{p+1}^1) \). Hence if the size of an optimal solution of SC is \((2p+1)n + 1\) then the
solution of ISC is 1.

**Corollary 7.5.8.** Intersection Set Chasing\((n, p)\) returns 1 if and only if the size of optimal solution of its corresponding Set Cover instance (as described here) is \((2p + 1)n + 1\).

**Observation 7.5.9.** Any streaming algorithm for Set Cover, \(I\), that in \(\ell\) passes solves the problem optimally with a probability of error \(err\) and consumes \(s\) memory space, solves the corresponding communication Set Cover problem in \(\ell\) rounds using \(O(sl^2)\) bits of communication with probability error \(err\).

*Proof.* Starting from player \(P_1\), each player runs \(I\) over its input sets and once \(P_i\) is done with its input, she sends the working memory of \(I\) publicly to other players. Then next player starts the same routine using the state of the working memory received from the previous player. Since \(I\) solves the Set Cover instance optimally after \(\ell\) passes using \(O(s)\) space with probability error \(err\), applying \(I\) as a black box we can solve \(P\) in \(\ell\) rounds using \(O(sl^2)\) bits of communication with probability error \(err\).

*Proof of 7.5.4:* By 7.5.9, any \(\ell\)-round \(O(s)\)-space algorithm that solves streaming Set Cover \((U, F)\) optimally can be used to solve the communication Set Cover\((U, F, p)\) problem in \(\ell\) rounds using \(O(sl^2)\) bits of communication. Moreover, by 7.5.8, we can decide the solution of the communication Intersection Set Chasing\((n, p)\) by solving its corresponding communication Set Cover problem. Note that while working with the corresponding Set Cover instance of Intersection Set Chasing\((n, p)\), all players know the collection of elements \(U\) and each player can construct its collection of sets \(F_i\) using \(f_i\) (or \(f'_i\)). However, by a result of [GO13], we know that any protocol that solves the communication Intersection Set Chasing\((n, p)\) problem with probability of error less than 1/10, requires \(\Omega\left(\frac{n^{1+1/(2p)}}{p^{16 \log^{3/2} n}}\right)\) bits of communication. Since in the corresponding Set Cover instance of the communication Intersection Set Chasing\((n, p)\), \(|U| = (2p + 1) \times 2n + 2p = O(np)\) and \(|F| \leq (2p + 1)n + 2pn = O(np)\), any \((p - 1)\)-pass streaming algorithm that solves the Set Cover problem optimally with a probability of error...
at most $1/10$, requires $\Omega(\frac{1+1/(2p)}{p \log^{1/2} n})$ bits of communication. Then using 7.5.9, since $\delta \geq \frac{\log \log n}{\log n}$, any $(\frac{1}{2\delta} - 1)$-pass streaming algorithm of SetCover that finds an optimal solution with error probability less than $1/10$, requires $\tilde{\Omega}(|F| \cdot |U|^{\delta})$ space. □

7.6 Lower Bound for Sparse Set Cover in Multiple Passes

In this part we give a stronger lower bound for the instances of the streaming SetCover problem with sparse input sets. An instance of the SetCover problem is $s$-Sparse Set Cover, if for each set $S \in \mathcal{F}$ we have $|S| \leq s$. We can use the same reduction approach described earlier in 7.5 to show that any $(1/2\delta - 1)$-pass streaming algorithm for $s$-Sparse Set Cover requires $\Omega(|\mathcal{F}|s)$ memory space if $s < |U|^\delta$ and $\mathcal{F} = O(U)$. To prove this, we need to explain more details of the approach of [GO13] on the lower bound of the communication Intersection Set Chasing problem. They first obtained a lower bound for Equal Pointer Chasing($n, p$) problem in which two instances of the communication Pointer Chasing($n, p$) are given and the goal is to decide whether these two instances point to a same value or not: $f_p(\cdots f_1(1) \cdots) = f'_p(\cdots f'_1(1) \cdots)$.

**Definition 7.6.1** ($r$-non-injective functions). A function $f : [n] \to [n]$ is called $r$-non-injective if there exists $A \subseteq [n]$ of size at least $r$ and $b \in [n]$ such that for all $a \in A$, $f(a) = b$.

**Definition 7.6.2** (Pointer Chasing Problem). Pointer Chasing($n, p$) is a $(p, p - 1)$ communication problem in which the player $i$ has a function $f_i : [n] \to [n]$ and the goal is to compute $f_1(f_2(\cdots f_p(1) \cdots))$.

**Definition 7.6.3** (Equal Limited Pointer Chasing Problem). Equal Pointer Chasing($n, p$) is a $(2p, p - 1)$ communication problem in which the first $p$ players have an instance of the Pointer Chasing($n, p$) problem and the other $p$ players have another instance of the Pointer Chasing($n, p$) problem. The output of the Equal Pointer Chasing($n, p$) is 1 if the solutions of the two instances of Pointer Chasing($n, p$) have the same value and 0 otherwise. Furthermore in another variant of pointer chasing
problem, Equal Limited Pointer Chasing\((n, p, r)\), if there exists \(r\)-non-injective function \(f_i\), then the output is 1. Otherwise, the output is the same as the value in Equal Pointer Chasing\((n, p)\).

For a boolean communication problem \(P\), \(\text{OR}_t(P)\) is defined to be \(\text{OR}\) of \(t\) instances of \(P\) and the output of \(\text{OR}_t(P)\) is \text{true} if and only if the output of any of the \(t\) instances is \text{true}. Using a direct sum argument, [GO13] showed that the communication complexity of \(\text{OR}_t(\text{Equal Limited Pointer Chasing}(n, p, r))\) is \(t\) times the communication complexity of \(\text{Equal Limited Pointer Chasing}(n, p, r)\).

**Lemma 7.6.4 ([GO13]).** Let \(n, p, t\) and \(r\) be positive integers such that \(n \geq 5p\), \(t \leq \frac{n}{4}\) and \(r = O(\log n)\). Then the amount of bits of communication to solve \(\text{OR}_t(\text{Equal Limited Pointer Chasing}(n, p, r))\) with error probability less than \(1/3\) is \(\Omega\left(\frac{tn}{p^2\log n}\right) - O(pt^2)\).

**Lemma 7.6.5 ([GO13]).** Let \(n, p, t\) and \(r\) be positive integers such that \(t^2pr^{p-1} < \frac{n}{10}\). Then if there is a protocol that solves \(\text{Intersection Set Chasing}(n, p, r)\) with probability of error less than \(1/10\) using \(C\) bits of communication, there is a protocol that solves \(\text{OR}_t(\text{Equal Limited Pointer Chasing}(n, p, r))\) with probability of error at most \(2/10\) using \(C + 2p\) bits of communication.

Consider an instance of \(\text{OR}_t(\text{Equal Limited Pointer Chasing}(n, p, r))\) in which \(t \leq n^\delta, r = \log(n), p = \frac{1}{22} - 1\) where \(\frac{1}{22} = o(\log n)\). By 7.6.4, the required amount of bits of communication to solve the instance with constant success probability is \(\widetilde{\Omega}(tn)\). Then, applying 7.6.5, to solve the corresponding \(\text{Intersection Set Chasing}\), \(\widetilde{\Omega}(tn)\) bits of communication is required.

In the reduction from \(\text{OR}_t(\text{Equal Limited Pointer Chasing}(n, p, r))\) to \(\text{Intersection Set Chasing}(n, p)\) (proof of 7.6.5), the \(r\)-non-injective property is preserved. In other words, in the corresponding \(\text{Intersection Set Chasing}\) instance each player’s functions \(f_i : [n] \to 2^n\) is union of \(t\) \(r\)-non-injective functions \(f_i(a) := f_{i,1}(a) \cup \cdots \cup f_{i,t}(a)^4\).

\(^4\)The \(\text{Intersection Set Chasing}\) instance is obtained by overlaying the \(t\) instances of \(\text{Equal Pointer Chasing}(n, p, r)\). To be more precise, the function of player \(i\) in instance \(j\) is \(\pi_{i,j} \circ f_{i,j} \circ \pi_{i+1,j}^{-1}\) (\(\pi\) are randomly chosen permutation functions) and then stack the functions on top of each other.
Given that none of the $f_{i,j}$ functions is $r$-non-injective, the corresponding Set Cover instance will have sets of size at most $rt$ ($S$-type sets are of size at most $t$ for $1 \leq i \leq p$ and of size at most $rt$ for $p + 1 \leq i \leq 2p$). Since $r = O(\log n)$, the corresponding Set Cover instance is $\tilde{O}(t)$-sparse. As we showed earlier in the reduction from Intersection Set Chasing to Set Cover, the number of elements (and sets) in the corresponding Set Cover instance is $O(np)$. Thus we have the following result for $s$-Sparse Set Cover problem.

**Theorem 7.6.6.** For $s \leq |U|^\delta$, any streaming algorithm that solves $s$-Sparse Set Cover($U, F$) optimally with probability of error less than $1/10$ in $(1/25 - 1)$ passes requires $\tilde{\Omega}(|F|s)$ memory space for $F = O(U)$. 
Chapter 8

Set Cover in the Sublinear Time Model\(^1\)

8.1 Introduction and Background

In this chapter, we study the Set Cover problem from the perspective of sublinear time algorithms. Given access to an instance of Set Cover in the query model, we show that sublinear algorithms derived from existing techniques have almost tight query complexities.

The greedy algorithm for the Set Cover can be implemented to run in time linear in the input size, however, the input size itself could be as large as \(\Theta(mn)\), so for large data sets even reading the input might be infeasible. This raises a natural question: is it possible to compute an approximate solution to minimum set cover in sublinear time?

This question was previously addressed in [NO08, YYI12], who showed that one can design constant running-time algorithms by simulating the greedy algorithm, under the assumption that the sets are of constant size and each element occurs in a constant number of sets. However, those constant-time algorithms have a few drawbacks: they only provide a mixed multiplicative/additive guarantee (the output cover size is guaranteed to be at most \(k \cdot \ln n + \epsilon n\)), the dependence of their running

---

\(^1\)The full version of this chapter appears in [IMR\(^{+}\)].
times on the maximum set size is exponential, and they only output the (approximate) minimum set cover size, not the cover itself. From a different perspective, [KY14] (building on [GK95]) showed that an $O(1)$-approximate solution to the fractional version of the problem can be found in $\tilde{O}(mk^2+nk^2)$ time. Combining this algorithm with the randomized rounding yields an $O(\log n)$-approximate solution to Set Cover with the same complexity.

In this thesis, we initiate a systematic study of the complexity of sublinear time algorithms for set cover with multiplicative approximation guarantees. Our upper bounds complement the aforementioned result of [KY14] by presenting algorithms which are fast when $k$ is large, as well as algorithms that provide more accurate solutions (even with a constant-factor approximation guarantee) that use a sublinear number of queries. Equally importantly, we establish nearly matching lower bounds, some of which even hold for estimating the optimal cover size. Our algorithmic results and lower bounds are presented in Table 8.1.

**Data access model.** As in the prior work [NO08, YYI12] on Set Cover, our algorithms and lower bounds assume that the input can be accessed via the adjacency-list oracle. More precisely, the algorithm has access to the following two oracles:

1. **ElToF:** Given a set $S_i$ and an index $j$, the oracle returns the $j^{th}$ element of $S_i$. If $j > |S_i|$, ⊥ is returned.

2. **SetToF:** Given an element $e_i$ and an index $j$, the oracle returns the $j^{th}$ set containing $e_i$. If $e_i$ appears in less than $j$ sets, ⊥ is returned.

This is a natural model, providing a “two-way” connection between the sets and the elements. Furthermore, for some graph problems modeled by Set Cover (such as Dominating Set or Vertex Cover), such oracles are essentially equivalent to the aforementioned incident-list model studied in sublinear graph algorithms. We also note that the other popular access model employing the membership oracle, where we

---

2The method can be further improved to $\tilde{O}(m+nk)$ (N. Young, personal communication).
3Note that polynomial time algorithm with sub-logarithmic approximation algorithms are unlikely to exist.
4In the context of graph problems, this model is also known as the incidence-list model, and has been studied extensively, see e.g., [CRT05, GKK13, BHNT15].
can query whether an element \( e \) is contained in a set \( S \), is not suitable for Set Cover, as it can be seen that even checking whether a feasible cover exists requires \( \Omega(mn) \) time.

### 8.1.1 Results

In this chapter, we present algorithms and lower bounds for the Set Cover problem. The results are summarized in Table 8.1. The NP-hardness of this problem (or even its \( o(\log n) \)-approximate version [Fei98, RS97, AMS06, Mos12, DS14]) precludes the existence of highly accurate algorithms with fast running times, while (as we show) it is still possible to design algorithms with sublinear query complexities and low approximation factors. The lower bound proofs hold for the running time of any approximation algorithm for set cover assuming the defined data access model.

We present two algorithms with sublinear number of queries. First, we show that the streaming algorithm presented in [HIMV16] (and also the previous chapter) can be adapted so that it returns an \( O(\alpha) \)-approximate cover using \( \tilde{O}(m(n/k)^{1/(\alpha-1)} + nk) \) queries, which could be quadratically smaller than \( mn \). Second, we present a simple algorithm which is tailored to the case when the value of \( k \) is large. This algorithm computes an \( O(\log n) \)-approximate cover in \( \tilde{O}(mn/k) \) time (not just query complexity). Hence, by combining it with the algorithm of [KY14], we get an \( O(\log n) \)-approximation algorithm that runs in time \( \tilde{O}(m + n\sqrt{m}) \).

We complement the first result by proving that for low values of \( k \), the required number of queries is \( \widetilde{\Omega}(m(n/k)^{1/(2\alpha)}) \), even for estimating the size of the optimal cover. This shows that the first algorithm is essentially optimal for the values of \( k \) where the first term in the runtime bound dominates. Moreover, we prove that even the Cover Verification problem, which is checking whether a given collection of \( k \) sets covers all the elements, would require \( \Omega(nk) \) queries. This provides strong evidence that the term \( nk \) in the first algorithm is unavoidable. Lastly, we complement the second algorithm, by showing a lower bound of \( \widetilde{\Omega}(mn/k^2) \) if the approximation ratio is a small constant.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Approximation</th>
<th>Constraints</th>
<th>Query Complexity</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Cover</td>
<td>αρ + ε</td>
<td>α ≥ 2</td>
<td>$O\left(\frac{1}{2}(m(\frac{n}{k})^{\frac{1}{\alpha-1}} + nk)\right)$</td>
<td>8.2.2</td>
</tr>
<tr>
<td></td>
<td>ρ + ε</td>
<td>-</td>
<td>$O\left(\frac{mn}{k^2}\right)$</td>
<td>8.2.3</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>k &lt; $\left(\frac{n}{\log m}\right)^{\frac{1}{2\alpha+1}}$</td>
<td>$\tilde{\Omega}(m(\frac{n}{k})^{1/(2\alpha)})$</td>
<td>8.4</td>
</tr>
<tr>
<td></td>
<td>α</td>
<td>$\alpha \leq 3/2$</td>
<td>$\tilde{\Omega}(\frac{mn}{k^2})$</td>
<td>8.4.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>k = $O\left(\frac{n}{\log m}\right)$</td>
<td>$\Omega(nk)$</td>
<td>8.3</td>
</tr>
<tr>
<td>Cover Verification</td>
<td>-</td>
<td>k ≤ n/2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.1: A summary of our algorithms and lower bounds. We use the following notation: $k \geq 1$ denotes the size of the optimum cover; $\alpha \geq 1$ denotes a parameter that determines the trade-off between the approximation quality and query/time complexities; $\rho \geq 1$ denotes the approximation factor of a “black box” algorithm for set cover used as a subroutine; We assume that $\alpha \leq \log n$ and $m \geq n$.

8.1.2 Related work

Sublinear algorithms for Set Cover under the oracle model have been previously studied as an estimation problem; the goal is only to approximate the size of the minimum set cover rather than constructing one. Nguyen and Onak [NO08] consider Set Cover under the oracle model we employ in this thesis, in a specific setting where both the maximum cardinality of sets in $\mathcal{F}$, and the maximum number of occurrences of an element over all sets, are bounded by some constants $s$ and $t$; this allows algorithms whose time and query complexities are constant, $(2^{(st)^{4}/\varepsilon})^{O(2^t)}$, containing no dependency on $n$ or $m$. They provide an algorithm for estimating the size of the minimum set cover when, unlike our work, allowing both ln s multiplicative and $\varepsilon n$ additive errors. Their result has been subsequently improved to $(st)^{O(s)/\varepsilon^2}$ by Yoshida et al. [YYI12]. Additionally, the results of Kuhn et al. [KMW06] on general packing/covering LPs in the distributed LOCAL model, together with the reduction method of Parnas and Ron [PR07], implies estimating set cover size to within a $O(\ln s)$-multiplicative factor (with $\varepsilon n$ additive error), can be performed in $(st)^{O(\log s \log t)/\varepsilon^4}$ time/query complexities.

Set Cover can also be considered as a generalization of the Vertex Cover problem. The estimation variant of Vertex Cover under the adjacency-list oracle model has
been studied in [PR07, MR06, ORRR12, YYI12].

8.2 SubLinear Algorithms for the Set Cover Problem

In this chapter, we present two different approximation algorithms for Set Cover with sublinear query in the oracle model: smallSetCover and largeSetCover. Both of our algorithms rely on techniques from recent streaming algorithms for Set Cover, but applying them to the oracle model requires novel insights and technical development.

Throughout the description of our algorithms, we assume that we have access to a black box subroutine that given full Set Cover instance (where all members of all sets are revealed), returns a $\rho$-approximate solution. The approximation factor $\rho$ may take on any value between 1 and $\Theta(\log n)$ depending on the computational model one assumes.

The first algorithm (smallSetCover) solves the Set Cover problem with an approximation factor of $(\alpha \rho + \varepsilon)$ using $\tilde{O}(\frac{1}{\varepsilon}(m(\frac{n}{k})^{1/\log n} + nk))$ queries, while the second algorithm (largeSetCover) achieves an approximation factor of $(\rho + \varepsilon)$ using $\tilde{O}(\frac{mn}{k^2})$ queries, where $k$ is the size of the minimum set cover. These algorithms can be combined so that the number of queries of the algorithm becomes asymptotically the minimum of the two:

**Theorem 8.2.1.** The Set Cover problem can be solved w.h.p.\(^5\) in the oracle model with approximation factor $O(\rho \log n)$, using $\tilde{O}(\min\{m\left(\frac{n}{k}\right)^{1/\log n} + nk, \frac{mn}{k}\}) = \tilde{O}(m + n\sqrt{m})$ number of queries.

8.2.1 Preliminaries

Our algorithms make use of two well-known sampling techniques for Set Cover: Element Sampling and Set Sampling. The first technique, Element Sampling, states that in order to find a $(1 - \delta)$-cover of the universe $U$, it is enough to solve Set Cover

---

\(^5\)An algorithm succeeds with high probability (w.h.p.) if its failure probability can be decreased to $n^{-c}$ for any constant $c > 0$ without affecting its asymptotic performance, where $n$ denotes the input size.
on a subset of elements of size $\tilde{O}(\frac{pk\log m}{\delta})$ sampled uniformly at random. It shows that we may restrict our attention to a subproblem with a much smaller number of elements, and our solution to the subproblem will still cover a good fraction of elements in the original instance. The next technique, Set Sampling, shows that if we add $\ell$ sets uniformly at random from $F$ to the cover, then each uncovered element only occurs in less than $\tilde{O}(\frac{m}{\ell})$ sets; that is, we are left with a much sparser remaining subproblem to solve. The formal statements of these sampling techniques are as follows. See [DIMV14] for the proofs.

**Lemma 8.2.2 (Element Sampling).** Consider an instance of Set Cover on $(U, F)$ whose optimal cover has size at most $k$. Let $U_{\text{mp}}$ be a subset of $U$ of size $\Theta(\frac{pk\log m}{\delta})$ chosen uniformly at random, and let $C_{\text{mp}} \subseteq F$ be a $\rho$-approximate cover for $U_{\text{mp}}$. Then, w.h.p. $C_{\text{mp}}$ covers at least $(1 - \delta)|U|$ elements.

**Lemma 8.2.3 (Set Sampling).** Consider an instance $(U, F)$ of Set Cover. Let $F_{\text{rmd}}$ be a collection of $\ell$ sets picked uniformly at random. Then, w.h.p. $F_{\text{rmd}}$ covers all elements that appear in $\Omega(\frac{m\log n}{\ell})$ sets of $F$.

### 8.2.2 The smallSetCover Algorithm

The algorithm presented in this section is adapted from the streaming algorithm of Set Cover in [HIMV16]. Before we proceed to the description of smallSetCover, we first describe its main subroutine, iterSetCover. This subroutine finds an approximate set cover, but its time and query complexities vary greatly depending on the desired approximation factor and the actual minimum set cover size. Later, we will have smallSetCover invoke iterSetCover twice with carefully-chosen parameters, in order to achieve the desired complexities.

iterSetCover receives $\alpha, \epsilon, l$ and $u$ as its arguments, and it is guaranteed that the size of an optimal cover of the input instance, $k$, is in $[l, u]$. Note that the algorithm does not know the correct value of $k$ and the sampling techniques described in Section 8.2.1 rely on $k$. Therefore, the algorithm needs to find a $(1 + \epsilon)$ estimate\(^6\) of

\(^6\)The exact estimate that the algorithm works with is a $(1 + \frac{\epsilon}{2pm})$ estimate.
denoted as $\ell$. This can be done by trying all powers of $(1 + \varepsilon)$ in $[l, u]$. Finally $\alpha$ is a trade-off parameter between the query complexity and the approximation guarantee that the algorithm achieves. We assume that the algorithm has access to a $\rho$-approximate black box solver of SetCover.

Now we are ready to explain the iterSetCover procedure. Assuming $\ell$ is a $(1 + \varepsilon)$ estimate for $k$, the algorithm first performs Set Sampling to cover all elements that occur in too many sets (more precisely, each uncovered element appears only in $O(m/\ell)$ sets). Then it goes through $\alpha - 2$ iterations; in each iteration, it performs Element Sampling with parameter $\delta = \tilde{O}((\ell/n)^{1/(\alpha-1)})$ in order to reduce the number of uncovered elements by a factor of $\delta$. At the end of $\alpha - 2$ iterations, only $\ell (n)^{1/(\alpha-1)}$ elements remain uncovered, for which the algorithm finds a cover by invoking the black box solver. The parameters are set so that all $\alpha - 1$ instances that are required to be solved by the black box solver ($\alpha - 2$ instances in the iterations and the final instance) are of size $\tilde{O}(\ell (n)^{1/(\alpha-1)} \cdot \frac{m}{\ell}) = \tilde{O}(m (n)^{1/(\alpha-1)})$.

**Streaming algorithm of set cover vs. smallSetCover.** Here, we highlight the differences/challenges of smallSetCover compared to its streaming counterpart. Firstly, the algorithm of [HIMV16] relies on the ability to test membership for a set-element pair when executing its set filtering subroutine: making a pass over the sets and pick all sets that cover $n/k$ yet uncovered elements, ensuring that after this pass the size of the sets over the uncovered elements has decreased. Instead, here we employ the set sampling which helps to reduce the number of sets that contain a (yet uncovered) element. Moreover, the algorithm may test multiple values of $\ell$ in parallel in the streaming model because the performance does not depend on $k$ there, but this is costly in our oracle model when $\ell$ is far-off: $m (n)^{1/(\alpha-1)}$ is large when $\ell = o(k)$, and $n\ell$ is large when $\ell = \omega(k)$. To overcome this problem, we have smallSetCover invoke iterSetCover twice. The first call aims at finding only a rough estimate of $k$, by setting $\alpha = \log n$. This ensures that the first term is low ($O(m)$), while checking its guess $\ell$ in an increasing order would also ensure that the second term is at most $nk$. Then, the second call aims for an accurate approximation, making use of a much tighter range of potential values of $k$ obtained during the first call.
In the rest of this section, we show that \texttt{smallSetCover} w.h.p. returns an almost \((\rho \alpha)\)-approximate solution of \texttt{Set Cover}(U, F) with query complexity \(\widetilde{O}(m \left(\frac{n}{k}\right)^{1/\alpha} + nk)\) where \(k\) is the size of a minimum set cover.

**Theorem 8.2.4.** The \texttt{smallSetCover} algorithm outputs a \((\alpha \rho + \varepsilon)\)-approximate solution of \texttt{Set Cover}(U, F) using \(q = \widetilde{O}(\frac{1}{\varepsilon} (m(n/k)^{1/\alpha} + nk))\) number of queries w.h.p., where \(k\) is the size of an optimal solution of the given instance.

To analyze the performance of \texttt{smallSetCover}, we first need to analyze the procedures used by \texttt{smallSetCover}: \texttt{iterSetCover} and \texttt{algOfflineSC}. The procedure \texttt{algOfflineSC}(Z, \ell) receives as an input a subset of elements \(Z\) and an estimate on the size of an optimal cover of \(Z\) using \(F\). \texttt{algOfflineSC} first determines all occurrences of \(Z\) in \(F\). Then it invokes a black box subroutine that returns a cover of size at most \(\rho \ell\) (if there exists a cover of size \(\ell\) for \(Z\)) for the \texttt{Set Cover} instance over \(Z\).

Moreover, we assume that all \texttt{ElToF} and \texttt{SetOf} oracles of \texttt{Set Cover}(U, F), \(|U|\) and \(|F|\) are given to all three subroutines.

**Lemma 8.2.5.** Suppose that each \(e \in Z\) appears in \(\widetilde{O}(\frac{m}{\ell})\) sets of \(F\) and let's assume that there exists a set of \(\ell\) sets in \(F\) that covers \(Z\). Then \texttt{algOfflineSC}(Z, \ell) returns a cover of size at most \(\rho \ell\) of \(Z\) using \(\widetilde{O}(\frac{m|Z|}{\ell})\) queries.

**Proof.** Since each element of \(Z\) is contained by \(\widetilde{O}(\frac{m}{\ell})\) sets in \(F\), the total information required to solve the \texttt{Set Cover} instance on \(Z\) can be obtained by \(\widetilde{O}(\frac{m|Z|}{\ell})\) \texttt{SetOf} queries (i.e. \(\widetilde{O}(\frac{m}{\ell})\) \texttt{SetOf} query per element in \(Z\)).

**Lemma 8.2.6.** After an iteration of the outer loop of \texttt{iterSetCover}(\(\alpha, \varepsilon, l, u\)) with parameter \(\ell > k\), w.h.p., sol\(_\ell\) covers U.

**Proof.** After picking \(\ell\) sets uniformly at random, by Set Sampling (Lemma 8.2.3), w.h.p. each element that is not in any of the sampled sets appears in at most \(\widetilde{O}(\frac{m}{\ell})\) sets of \(F\). Next, by Element Sampling (Lemma 8.2.2 with \(\delta = (\frac{l}{n})^{1/(\alpha - 1)}\)), at the end of each iteration, w.h.p. the number of uncovered elements decreases by a factor of \((\frac{l}{n})^{1/(\alpha - 1)}\). Thus after at most \((\alpha - 2)\) iterations, w.h.p. less than \((\frac{n}{\ell})^{1/(\alpha - 1)}\) elements
\textbf{Algorithm 19: iterSetCover} is the main procedure of the \textbf{smallSetCover} algorithm for the \textbf{SetCover} problem. Here, \textbf{algOfflineSC}(Z, \ell) invokes a black box that returns a cover of size at most $\rho \ell$ (if there exists a cover of size $\ell$ for $Z$) for the \textbf{SetCover} instance that is the projection of $\mathcal{F}$ over $Z$. Moreover, we assume that all \textbf{ELTOF} and \textbf{SETOF} oracles of \textbf{SetCover}(U, \mathcal{F}), |U| and $|\mathcal{F}|$ are given to both subroutines.
remain uncovered. Finally, by solving Set Cover over the remaining elements in $U_{rem}$, 
\textit{iterSetCover} returns a collection of sets whose union covers $U$. \hfill \Box

Next we analyze the query complexity and the approximation guarantee of \textit{iterSetCover}. As we only apply Element Sampling and Set Sampling polynomially many times, all invocations of the corresponding lemmas during an execution of the algorithm must succeed w.h.p., so we assume their guarantees for the remaining of the proof.

\textbf{Lemma 8.2.7.} Given that $l \leq k \leq \frac{u}{1+\frac{\varepsilon}{1+\frac{\alpha}{2}}}w.h.p. \textit{iterSetCover}(\alpha, \varepsilon, l, u)$ finds a $(\rho \alpha + \varepsilon)$-approximate solution of the input instance using $\tilde{O}\left(\frac{1}{\varepsilon}(m\ell^{1/(\alpha-1)} + nk)\right)$ queries.

\textit{Proof.} Let $\ell_k = (1 + \frac{\varepsilon}{2\alpha})\left\lceil \log_{1+\frac{\varepsilon}{2\alpha}} \frac{k}{l} \right\rceil$, the smallest power of $1 + \frac{\varepsilon}{2\alpha}$ greater than or equal to $k$, which is preconditioned to be in $[l, u]$. By Lemma 8.2.6, \textit{iterSetCover} terminates with $\ell \leq \ell_k$. For each set picked by Set Sampling, we spend up to $n\text{ElToF}$ queries learning its members. So, with $O(n\ell)$ queries, we may deduce the set of remaining, uncovered elements $U_{rem}$.

Next, in each iteration of the inner loop, we sample a subset $Z$ of size $\tilde{O}\left(\ell\left(\frac{n}{\ell}\right)^{1/(\alpha-1)}\right)$ from $U_{rem}$. Recall that, by Set Sampling (Lemma 8.2.3), each $e \in Z \subset U_{rem}$ appears in at most $\tilde{O}\left(\frac{m}{n}\right)$ sets. Then, we invoke \textsc{algOfflineSC} which requires $\tilde{O}\left(\ell\left(\frac{n}{\ell}\right)^{1/(\alpha-1)} \cdot \frac{m}{n}\right) = \tilde{O}\left(m\left(\frac{n}{\ell}\right)^{1/(\alpha-1)}\right)$ \textsc{SetOf} queries in total (Lemma 8.2.5). With \textsc{algOfflineSC}, we find a $\rho$-approximate solution $D$ over the sampled elements; by Element Sampling (Lemma 8.2.2), $D$ covers a $(1 - (\ell/n)^{1/(\alpha-1)})$-fraction of $U_{rem}$ at the beginning of the iteration. Finally, at the end of each iteration, we update $U_{rem}$ by removing elements covered by $D$, which requires at most $n\text{ElToF}$ queries per each set in $D$, constituting $\tilde{O}(n\ell)$ queries in total. The Feasibility Test ensures that the finishing steps require roughly the same number of queries. So the total number of queries for an iteration of the outerloop of \textit{iterSetCover} with parameter $\ell$ is $\tilde{O}\left(m\left(\frac{n}{\ell}\right)^{1/(\alpha-1)} + n\ell\right)$.

Since by Lemma 8.2.6, if $\ell_k \leq u$, then the outer loop of \textit{iterSetCover} is executed only for $l \leq \ell \leq \ell_k$ before it terminates. Thus, the total number of queries made by
iterSetCover is:

\[
\sum_{i=\log_{1+\frac{\varepsilon}{2\alpha\rho}}^\log_{1+\frac{\varepsilon}{2\alpha\rho}}(l_k)} \bar{O}(m \left( \frac{n}{(1 + \frac{\varepsilon}{2\alpha\rho})^i} \right) ^{\frac{1}{\alpha-1}} + n \left( 1 + \frac{\varepsilon}{2\alpha\rho} \right) ^i ) \\
= \bar{O}(m \left( \frac{n}{l} \right) ^{\frac{1}{\alpha-1}} (\log_{1+\frac{\varepsilon}{2\alpha\rho}}(l_k) ) + \frac{n\ell_k}{\varepsilon/(\rho\alpha)}) \\
= \bar{O}(\frac{1}{\varepsilon}(m \left( \frac{n}{l} \right) ^{1/(\alpha-1)} + nk)).
\]

Now, we show that the number of sets returned by iterSetCover is not more than 
\((\alpha\rho + \varepsilon)\ell_k\). Set Sampling picks \(\ell\) sets and in each of the iterations, algOfflineSC guarantees that the number of sets selected is at most \(\rho\ell\) (because otherwise, we would stop the algorithm and move on to the next iteration of the outer loop with a larger \(\ell\)). Moreover, the algorithm picks at most \(\rho\ell\) more sets after the Feasibility Test. Thus the size of the solution returned by iterSetCover is at most

\[(1 + (\alpha - 1)\rho)\ell_k < (1 + (\alpha - 1)\rho) \left( 1 + \frac{\varepsilon}{2\alpha\rho} \right) k \leq (\alpha\rho + \varepsilon)k.\]

\(\square\)

Parameter specification. As we showed in the proof of Lemma 8.2.7, the number of queries that iterSetCover makes given parameter \(\ell\) in the specified range \([l, u]\) is \(\bar{O}(m \left( \frac{n}{l} \right) ^{1/(\alpha-1)})\). Let \(k\) be the size of the minimum set cover of SetCover\((U, \mathcal{F})\). Since we do not know the value of \(k\) apriori, we need to examine several values and return the best solution among them (e.g., the set \{(1+\varepsilon)^i \mid \log_{1+\frac{\varepsilon}{2\alpha\rho}} l \leq i \leq \log_{1+\frac{\varepsilon}{2\alpha\rho}} u\} that we examine in iterSetCover). If \(\ell\) is much larger than \(k\), then the \(n\ell\) term dominates \(nk\) in our total number of queries. On the other hand, if \(\ell\) is much smaller than \(k\), then the \(m\left( \frac{n}{l} \right) ^{1/(\alpha-1)}\) term dominates \(m\left( \frac{n}{k} \right) ^{1/(\alpha-1)}\). In order to keep the number of queries bounded by \(\bar{O}(m \left( \frac{n}{k} \right) ^{1/(\alpha-1)} + nk)\), first we need to find a rough estimate of \(k\). As described in smallSetCover (Figure 20), we first find an \(O(\rho \log n)\) estimate of the size of an optimal set cover \(k'\) by performing iterSetCover with \(\alpha = \log n\). This step requires \(\bar{O}(m + nk)\) queries. Then, we perform iterSetCover for the range
where \( l = \left\lfloor \frac{k'}{(\rho \log n)} \right\rfloor \) and \( u = \left\lceil \frac{k'(1 + \frac{\varepsilon}{2\alpha \rho})}{\rho \log n} \right\rceil \). This trick enables us to obtain the query complexity of \( \tilde{O}\left(\frac{1}{\varepsilon} (m \left( \frac{n}{k} \right)^{1/(\alpha - 1)} + nk)\right) \) for computing an approximate \( \text{Set Cover}(U, \mathcal{F}) \) with multiplicative factor of \( \alpha \rho + \varepsilon \). Next, we prove the main theorem of the section.

Proof of Theorem 8.2.4: \( \text{smallSetCover} \) first finds a \((\rho \log n)\)-approximate solution of \( \text{Set Cover}(U, \mathcal{F}) \), \( \text{sol} \), with \( \tilde{O}(m + nk) \) queries by calling \( \text{iterSetCover}(\log n, 1, 1, n) \). Having that \( k \leq k' = |\text{sol}| \leq (\rho \log n)k \), the algorithm calls \( \text{iterSetCover} \) with \( \alpha \) as the approximation factor and \( \frac{k'}{\rho \log n}, \left\lceil \frac{k'(1 + \frac{\varepsilon}{2\alpha \rho})}{\rho \log n} \right\rceil \) as the range containing \( k \). By Lemma 8.2.7, the second call to \( \text{iterSetCover} \) in \( \text{smallSetCover} \) returns a \((\alpha \rho + \varepsilon)\)-approximate solution of \( \text{Set Cover}(U, \mathcal{F}) \) using the following number of queries:

\[
\tilde{O}\left(\frac{1}{\varepsilon} (m \left( \frac{n}{k/\rho \log n} \right)^{1/(\alpha - 1)} + nk)\right) = \tilde{O}\left(\frac{1}{\varepsilon} (m \left( \frac{n}{k} \right)^{1/(\alpha - 1)} + nk)\right)
\]

\( \square \)

8.2.3 The largeSetCover Algorithm

The algorithm \( \text{largeSetCover} \), given in Figure 21, first randomly picks \( \varepsilon \ell/3 \) sets. By the Set Sampling lemma, we have that w.h.p. every element that occurs in more than \( \Theta\left(\frac{m \log n}{\varepsilon \ell} \right) \) sets of \( \mathcal{F} \) will be covered. It then solves the Set Cover instance over the elements that occur less frequently using a black box algorithm of Set Cover; note that this may include some already covered elements. In order to get the promised query complexity, we need to enumerate our estimates \( \ell \) of the size of an optimal set.
cover in the decreasing order. The algorithm is able to construct feasible solutions while $\ell \geq k$ and once it cannot find a feasible solution for $\ell$, it returns the solution constructed by the previous estimate of $k$, i.e., $\ell(1 + \frac{\varepsilon}{3\rho})$. Since we perform Set Sampling for $\tilde{O}(\frac{1}{\varepsilon})$ iterations, w.h.p. the total query complexity of `largeSetCover` is $\tilde{O}(\frac{mn}{k\varepsilon^2})$. The advantage of `largeSetCover` is that it does not need to update the set of uncovered elements at any point and simply avoids the additive $nk$ term in the query complexity bound; the result of Section 8.3 suggests that the $nk$ term may be unavoidable if one wishes to maintain the uncovered elements. Moreover, testing whether the number of occurrences of an element is less than $\frac{cm \log n}{\varepsilon \ell}$ only requires a single query, namely $\text{SetOf}(e, \frac{cm \log n}{\varepsilon \ell})$.

Algorithm 21: A $(\rho + \varepsilon)$-approximation algorithm for the SetCover problem. We assume that the algorithm has access to $\text{ElToF}$ and $\text{SetOf}$ oracles for $\text{SetCover}(U, \mathcal{F})$, as well as $|U|$ and $|\mathcal{F}|$.

We now prove the desired guarantees of `largeSetCover`.

Lemma 8.2.8. `largeSetCover` returns a $(\rho + \varepsilon)$-approximate solution of $\text{SetCover}(U, \mathcal{F})$ w.h.p.

Proof. The algorithm `largeSetCover` tries to construct set covers of decreasing sizes until it fails. Clearly, if $k \leq \ell$ then the black box algorithm finds a cover of size at most $\rho \ell$ for any subset of $U$, because $k$ sets are already sufficient to cover $U$. In other words,
the algorithms does not terminate with \( \ell \geq k \). Moreover, since when the algorithm terminates \( \ell \) is smaller than \( k \), the size of the set cover found by \texttt{largeSetCover} is at most \((\frac{\varepsilon}{3} + \rho)(1 + \frac{\varepsilon}{3})\ell < (\frac{\varepsilon}{3} + \rho)(1 + \frac{\varepsilon}{3})k < (\rho + \varepsilon)k\).

\[ \text{Lemma 8.2.9. The number of queries made by } \texttt{largeSetCover} \text{ is } \tilde{O}(\frac{mn}{k\varepsilon^2}). \]

\[ \text{Proof.} \] The value of \( \ell \) in any iteration of the algorithm is greater than \( k/(\rho + \varepsilon) \), since otherwise the size of the solution constructed by the algorithm is at most \((\rho + \varepsilon)\ell < k \) which is a contradiction.

Set Sampling lemma guarantees (w.h.p.) that each uncovered element appears in at most \( \Theta(\frac{m \log n}{\varepsilon \ell}) \) sets and thus the algorithm needs to perform \( \tilde{O}(\frac{mn}{\varepsilon \ell}) \) \texttt{SetOf} queries to construct \( F_{\text{rare}} \). Moreover, the number of required queries in the size test step is \( O(n) \) because we use one \texttt{SetOf} query per each element in \( U \). Thus, the query complexity of \texttt{largeSetCover}(\( \varepsilon \)) is bounded by

\[ \sum_{i=1}^{\log_1 + \frac{\varepsilon}{3} n} \tilde{O} \left( n + \frac{mn}{\varepsilon (1 + \frac{\varepsilon}{3})^i} \right) = \tilde{O} \left( n + \frac{mn}{\varepsilon k \log_1 + \frac{\varepsilon}{3} n} \right) = \tilde{O} \left( \frac{mn}{k \varepsilon^2} \right). \]

\[ \text{Lemma 8.2.10. The running time of } \texttt{largeSetCover} \text{ is } \tilde{O} \left( \frac{1}{\varepsilon} \cdot T_{sc}(\tilde{O}(\frac{mn}{k\varepsilon}), \rho) + q \right) \text{ where } q \text{ is the number of queries made by the algorithm.} \]

\[ \text{Proof.} \] The only term we need to consider is the run time of the \( \rho \)-approximate black box solver of \texttt{SetCover}, as other terms are dominated by the number of queries that \texttt{largeSetCover} makes. Thus the total runtime is

\[ O(q) + \sum_{i=1}^{\log_1 + \frac{\varepsilon}{3} n} \tilde{O} \left( T_{sc} \left( \tilde{O}(\frac{mn}{k\varepsilon}), \rho \right) \right) = \tilde{O} \left( \frac{1}{\varepsilon} \cdot T_{sc}(\tilde{O}(\frac{mn}{k\varepsilon}), \rho) + q \right). \]

By Lemmas 8.2.8, 8.2.9, and 8.2.10, we establish the following theorem.
Theorem 8.2.11. \( \text{largeSetCover}(\varepsilon) \) finds a \((\rho + \varepsilon)\)-approximate solution of \( \text{SetCover}(U, \mathcal{F}) \) w.h.p. using \( q = \tilde{O}\left(\frac{mn}{k\varepsilon^2}\right) \) queries in time \( \tilde{O}\left(\frac{1}{\varepsilon} \cdot T_{sc}(\tilde{O}\left(\frac{mn}{k\varepsilon^2}\right), \rho) + q\right) \).

8.3 Lower Bound for the Cover Verification Problem

In this section, we give a tight lower bound on a feasibility variant of the Set Cover problem which we refer to as Cover Verification. In Cover Verification\((U, \mathcal{F}, \mathcal{F}_k)\), besides a collection of \( m \) sets \( \mathcal{F} \) and \( n \) elements \( U \), we are given indices of \( k \) sets \( \mathcal{F}_k \subseteq \mathcal{F} \), and the goal is to determine whether they are covering the whole universe \( U \) or not. We note that, throughout this section, the parameter \( k \) is a candidate for, but not necessarily the value of, the size of the minimum set cover. We remark that our result of this section is also applicable to the fractional case: the candidate solution we wish to check is simply the indicator vector \( x \) such that \( x_S = 1 \) for each set \( S \in \mathcal{F}_k \), and \( x_S = 0 \) if \( S \notin \mathcal{F}_k \).

A naive approach for this decision problem is to query all elements in the given \( k \) sets and then check whether they cover \( U \) or not; this approach requires \( O(nk) \) queries. However, in what follows we show that this approach is tight and no randomized protocol can decide whether a given \( k \) sets covers the whole universe with probability of success at least 0.9 using \( o(nk) \) queries.

Theorem 8.3.1. Any (randomized) algorithm for deciding whether a given \( k = \Omega(\log n) \) sets covers all elements with probability of success at least 0.9, requires \( \Omega(nk) \) queries.

While this lower bound does not directly lead to a lower bound on Set Cover, it suggests that verifying the feasibility of a solution these problems may even be more costly than finding the approximate solution itself; any algorithm bypassing this \( \Omega(nk) \) lower bound may not solve Cover Verification as a subroutine. This section also uses a similar framework as the upcoming, more involved lower bound proofs for Set Cover in Section 8.4 where we will be reusing many basic ideas and notations from this section.
We prove our lower bound by designing the Yes and No instances that are hard to distinguish, such that for a Yes instance, the union of the given $k$ sets is $U$, while for a No instance, their union only covers $n - 1$ elements. Each Yes instance is indistinguishable from a good fraction of No instances. Thus any algorithm must unavoidably answer incorrectly on half of these fractions, and fail to reach the desired probability of success.

**Representation of Set Structures.** Before we proceed with the proof, we formally specify the representation of the set structures of input instances, which applies to both Cover Verification and Set Cover. The set structure of each instance is fully described via two functions defining the answers from our oracles: $\text{ElToF}(S, i)$, which returns the $i^{th}$ element of the set $S$, and $\text{SetOf}(e, i)$, which returns the $i^{th}$ set containing the element $e$. Two instances are considered different if the labeling of sets or elements, or the order in which elements or sets appear on these functions, are different. In particular, there can be many instances whose underlying structures of sets and elements are equivalent.

**Definition 8.3.2.** By query-answer history, we denote the sequence of query-answer pairs $\langle (q_1, a_1), (q_2, a_2), \ldots, (q_r, a_r) \rangle$ recording the communication between the algorithm and the oracles, where each new query $q_{i+1}$ may only depend on the query-answer pairs $(q_1, a_1), \ldots, (q_i, a_i)$.

### 8.3.1 Underlying Set Structure

Our instance contains $n$ sets and $n$ elements (so $m = n$), where the first $k$ sets forms $\mathcal{F}_k$, the candidate for the set cover we wish to verify. We first consider the incidence matrix representation, such that the rows represent the sets and the columns represent the elements. We focus on the first $n/k$ elements, and consider a slab, composing of $n/k$ columns of the incidence matrix. We define a basic slab as the structure illustrated in 8-1a (for $n = 12$ and $k = 3$), where the cell $(i, j)$ is white if $e_j \in S_i$, and is gray otherwise. The rows are divided into blocks of size $k$, where first block, the query block, contains the rows whose sets we wish to check for coverage; notice that only the
last element is not covered. More specifically, in a basic slab, the query block contains sets \( S_1, \ldots, S_{n/k} \), each of which is equal to \( \{e_1, \ldots, e_{n/k-1}\} \). The subsequent rows form the swapper blocks each consisting of \( n/k \) sets. The \( r \)th swapper block consists of sets \( S_{(r+1)n/k+1}, \ldots, S_{(r+2)n/k} \), each of which is equal to \( \{e_1, \ldots, e_{n/k}\} \setminus \{e_r\} \).

Next, we define the swap operation that exchanges elements between the sets.

**Definition 8.3.3 (swap operation).** Consider two sets \( S \) and \( S' \). A swap on \( S \) and \( S' \) is defined over two elements \( e,e' \) such that \( e \in S \setminus S' \) and \( e' \in S' \setminus S \), where \( S \) and \( S' \) exchange \( e \) and \( e' \). Formally, after performing \( \text{swap}(e,e') \), \( S = (S \cup \{e'\}) \setminus \{e\} \) and \( S' = (S' \cup \{e\}) \setminus \{e'\} \). As for the representation via \( \text{ElToF} \) and \( \text{SetOf} \), each application of \( \text{swap} \) only modifies 2 entries for each oracle. That is, if previously \( e = \text{ElToF}(S,i), \ S = \text{SetOf}(e,j), \ e' = \text{ElToF}(S',i'), \ and \ S' = \text{SetOf}(e',j') \), then their new values change as follows: \( e' = \text{ElToF}(S,i), \ S' = \text{SetOf}(e,j), \ e = \text{ElToF}(S',i'), \ and \ S = \text{SetOf}(e',j'). \)

We perform one swap in this slab. Consider a parameter \((x,y)\) representing the index of a white cell within the query block. We exchange the color of this white cell with the gray cell on the same row, and similarly exchange the same pair of cells on swapper block \( y \). An example is given in 8-1b; the dashed blue rectangle corresponds to the indices parameterizing possible swaps, and the red squares mark the modified cells. This modification corresponds to a single swap operation; in this example, choosing the index \((3, 2)\) swaps \((e_2, e_4)\) between \( S_3 \) and \( S_9 \). Observe that there are \( k \times (n/k - 1) = n - k \) possible swaps on a single slab, and any single swap allows the query sets to cover all \( n/k \) elements.

Lastly, we may create the full instance by placing all \( k \) slabs together, as shown in 8-2, shifting the elements’ indices as necessary. The structure of our sets may be specified solely by the swaps made on these slabs. We define the structure of our instances as follows.

- For a Yes instance, we make one random swap on each slab. This allows the first \( k \) sets to cover all elements.
- For a No instance, we make one random swap on each slab except for exactly
Figure 8-1: A basic slab and an example of a swapping operation.

Figure 8-2: A example structure of a Yes instance; all elements are covered by the first 3 sets.
one of them. In that slab, the last element is not covered by any of the first $k$ sets.

Now, to properly define an instance, we must describe our structure via \textsc{ElToF} and \textsc{SetOf}. We first create a temporary instance consisting of $k$ basic slabs, where none of the cells are swapped. Create \textsc{ElToF} and \textsc{SetOf} lists by sorting each list in an increasing order of indices. Each instance from the above construction can then be obtained by applying up to $k$ swaps on this temporary instance. 8-3 provides a sample realization of a basic slab with \textsc{ElToF} and \textsc{SetOf}, as well as a sample result of applying a swap on this basic slab; these correspond to the incidence matrices in 8-1a and 8-1b, respectively. Such a construction can be extended to include all $k$ slabs. Observe here that no two distinct swaps modify the same entry; that is, the swaps do not interfere with one another on these two functions. We also note that many entries do not participate in any swap.

### 8.3.2 Proof of Theorem 8.3.1

We assume that the algorithm has the knowledge of our construction; this may only strengthen the lower bound. The algorithm knows the underlying structure, as well as the fact that a swap is chosen at random on each slab (if any). It also knows, for the \textsc{Yes} instance and each possible \textsc{No} instance, the entries of \textsc{ElToF} and \textsc{SetOf}. Its task is only to find out whether the given instance is a \textsc{Yes} instance or a \textsc{No} instance.

We make the following observation. From the property of the construction remarked above, the algorithm may verify, with a single query, whether a certain swap occurs in a certain slab. Namely, it is sufficient to query an entry of \textsc{ElToF} or \textsc{SetOf} that would have been modified by that swap, and check whether it is actually modified or not. For simplicity, we assume that our algorithm does not make multiple queries about the same swap, or make a query that is not corresponding to any swap; otherwise it could have been replaced by a more efficient algorithm that deduces the answer to such a query without asking our oracle.

We employ Yao’s principle as follows: to prove a lower bound for randomized algorithms, we show a lower bound for any deterministic algorithm on a fixed distri-
### Before: `ELToF` table for a basic slab

<table>
<thead>
<tr>
<th>ELToF</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_5$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_8$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_9$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_{10}$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_{11}$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_{12}$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>

### After: `ELToF` table after applying a swap

<table>
<thead>
<tr>
<th>ELToF</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$e_1$</td>
<td>$e_4$</td>
<td>$e_3$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_5$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$e_2$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_8$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_9$</td>
<td>$e_1$</td>
<td>$e_3$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_{10}$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_{11}$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_4$</td>
</tr>
<tr>
<td>$S_{12}$</td>
<td>$e_1$</td>
<td>$e_2$</td>
<td>$e_4$</td>
</tr>
</tbody>
</table>

### Before: SETOF table for a basic slab

<table>
<thead>
<tr>
<th>SETOF</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_7$</td>
<td>$S_8$</td>
<td>$S_9$</td>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$S_{12}$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$S_{12}$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$S_7$</td>
<td>$S_8$</td>
<td>$S_9$</td>
</tr>
<tr>
<td>$e_4$</td>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$S_7$</td>
<td>$S_8$</td>
<td>$S_9$</td>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$S_{12}$</td>
</tr>
</tbody>
</table>

### After: SETOF table after applying a swap

<table>
<thead>
<tr>
<th>SETOF</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_7$</td>
<td>$S_8$</td>
<td>$S_9$</td>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$S_{12}$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$S_{12}$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$S_7$</td>
<td>$S_3$</td>
<td>$S_9$</td>
</tr>
<tr>
<td>$e_4$</td>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$S_7$</td>
<td>$S_8$</td>
<td>$S_9$</td>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$S_{12}$</td>
</tr>
</tbody>
</table>

**Figure 8-3:** Tables illustrating the representation of a slab under `ELToF` and `SETOF` before and after a swap; cells modified by $\text{swap}(e_2, e_4)$ between $S_3$ and $S_9$ are highlighted in red.
distribution of input instances. Let \( s = n - k \) be the number of possible swaps in each slab; assume \( s = \Theta(n) \). We define our distribution of instances as follows: each of the \( s^k \) possible Yes instances occurs with probability \( 1/(2s^k) \), and each of the \( ks^{k-1} \) possible No instances occurs with probability \( 1/(2ks^{k-1}) \). Equivalently speaking, we create a random Yes instance by making one swap on each basic slab. Then we make a coin flip: with probability \( 1/2 \) we pick a random slab and undo the swap on that slab to obtain a No instance; otherwise we leave it as a Yes instance. To prove by contradiction, assume there exists a deterministic algorithm that solves the Cover Verification problem over this distribution of instances with \( r = o(sk) \) queries.

Consider the Yes instances portion of the distribution, and observe that we may alternatively interpret the random process generating them as as follows. For each slab, one of its \( s \) possible swaps is chosen uniformly at random. The algorithm may make queries to inspect these possible swaps, one by one, potentially up to \( s \) times. By the principle of deferred decisions, the swap is equally likely to occur at any index of this sequence of \( s \) queries, independent of swaps in other slabs. Thus we can create a tuple of \( k \) random variables \( \mathcal{T} = (T_1, \ldots, T_k) \) such that for each \( i \leq k \), \( T_i \) is chosen uniformly at random from \( \{1, \ldots, s\} \). The variable \( T_i \) indicates the index of the actual swap in the sequence of queries from the \( i \)th slab; that is, only the \( T_i \)th query to the \( i \)th slab detects the actual swap. A No instance may be described via the same tuple \( \mathcal{T} \), except that one of the \( T_i \)'s is set to \( s + 1 \) when the \( i \)th slab contains no swap, indicating that the swap will never be found.

We now show the following two lemmas in order to bound the number of swaps the algorithm may encounter throughout its execution.

**Lemma 8.3.4.** Let \( b > 3 \) be a fixed constant and define \( \mathcal{T}_{\text{high}} = \{i \mid T_i \geq \frac{s}{b}\} \). If \( k \geq 14b \), then \( |\mathcal{T}_{\text{high}}| \geq (1 - \frac{2}{b})k \) with probability at least 0.99.

**Proof.** Let \( \mathcal{T}_{\text{low}} = \{1, \ldots, k\} \setminus \mathcal{T}_{\text{high}} \). Notice that on the \( i \)th slab, \( \Pr[i \in \mathcal{T}_{\text{low}}] < \frac{1}{b} \) independently of other slabs, and thus \( |\mathcal{T}_{\text{low}}| \) is stochastically dominated by \( \mathcal{B}(k, \frac{1}{b}) \), the binomial distribution with \( k \) trials and success probability \( \frac{1}{b} \). Applying Chernoff
bound, we obtain

\[ \Pr \left[ |T_{\text{low}}| \geq \frac{2k}{b} \right] \leq e^{-\frac{k}{3b}} < 0.01. \]

Hence, \( |T_{\text{high}}| \geq k - \frac{2k}{b} = (1 - \frac{2}{b})k \) with probability at least 0.99, as desired. \( \Box \)

**Lemma 8.3.5.** If the total number of queries made by the algorithm is less than \((1 - \frac{3}{b})\frac{nk}{b}\), then with probability at least 0.99, the algorithm will not see any swaps from at least \( \frac{k}{b} \) slabs.

**Proof.** If the total number of such queries is less than \((1 - \frac{3}{b})\frac{nk}{b}\), then the number of queries from at least \( \frac{3k}{b} \) slabs is less than \( \frac{s}{b} \) each. Assume the condition of Lemma 8.3.4: for at least \((1 - \frac{3}{b})k\) slabs, \( T_i \geq \frac{s}{b} \). That is, the algorithm will not encounter a swap if it makes less than \( \frac{s}{b} \) queries from such a slab. Then, there are at least \( \frac{k}{b} \) slabs with \( T_i \geq \frac{s}{b} \) from which the algorithm makes less than \( \frac{s}{b} \) queries, and thus does not find a swap. Overall this event holds with probability at least 0.99 due to Lemma 8.3.4. \( \Box \)

Suppose that the deterministic algorithm makes less than \((1 - \frac{3}{b})\frac{nk}{b}\) queries, then for a fraction of 0.99 of all possible tuples \( T \), the output of the Yes instance is the same as the output of \( \frac{1}{b} \) fraction of No instances, namely when the slab containing no swap is one of the \( \frac{k}{b} \) slabs that the algorithm has not detected a swap in the corresponding Yes instance; the algorithm must answer incorrectly on half of the corresponding weight in our distribution of input instances. Thus the probability of success for any algorithm with less than \((1 - \frac{3}{b})\frac{nk}{b}\) queries is at most

\[ 1 - \Pr \left[ |T_{\text{high}}| \geq \left( 1 - \frac{2}{b} \right)^k \cdot \left( \frac{1}{b} \right) \cdot \left( \frac{1}{2} \right) \right] \leq 1 - \frac{0.495}{b} < 0.9, \]

for a sufficiently small constant \( b > 3 \) (e.g. \( b = 4 \)). As \( s = \Theta(n) \) and by Yao’s principle, this implies the lower bound of \( \Omega(nk) \) for the Cover Verification problem.
8.4 Simplified Lower Bounds for the Set Cover Problem

In this section, we present the proof of lower bounds for Set Cover. For clarity, here we only describe a simplified version of our approach which applies to approximation protocols for distinguishing between instances with minimum set cover sizes 2 and 3, and show a lower bound of $\tilde{\Omega}(mn)$ (which is tight up to a polylogarithmic factor). We include the full generalization of our approach for other values of approximation factor $\alpha$ and minimum set cover size $k$ in [IMR$^+$$]$. More precisely, in [IMR$^+$], we will show the query complexity lower bound of $\tilde{\Omega}(m(n/k)^{1/2})$ for computing an $\alpha$-approximate solution to the Set Cover problem, where $m$ is the number of sets, $n$ is the number of elements, and $k$ is the size of the minimum set cover of the input instance.

**Theorem 8.4.1.** For $2 \leq k \leq \lfloor \frac{n}{160 \log m} \rfloor^{1/4} + 1$ and $1 < \alpha \leq \log n$, any randomized algorithm that solves the Set Cover problem with approximation factor $\alpha$ and success probability at least $2/3$ requires $\tilde{\Omega}(m(n/k)^{1/2})$ queries.

**High level idea.** Our approach for establishing the lower bound is as follows. First, we construct a median instance $I^*$ for Set Cover, whose minimum set cover size is 3. We then apply a randomized procedure genModifiedInst, which slightly modifies the median instance into a new instance containing a set cover of size 2. Applying Yao’s principle, the distribution of the input to the deterministic algorithm is either $I^*$ with probability $1/2$, or a modified instance generated thru genModifiedInst($I^*$), which is denoted by $D(I^*)$, again with probability $1/2$. Next, we consider the execution of the deterministic algorithm. We show that unless the algorithm asks at least $\tilde{\Omega}(mn)$ queries, the resulting query-answer history generated over $I^*$ would be the same as those generated over instances constituting a constant fraction of $D(I^*)$, reducing the algorithm’s success probability to below $2/3$. More specifically, we will establish the following theorem.

**Theorem 8.4.2.** Any algorithm that can distinguish whether the input instance is $I^*$
or belongs to \( D(I^*) \) with probability of success greater than \( 2/3 \), requires \( \Omega(mn/\log m) \) queries.

**Corollary 8.4.3.** For \( 1 < \alpha < 3/2 \), and \( k \leq 3 \), any randomized algorithm that approximates by a factor of \( \alpha \), the size of the optimal cover for the Set Cover problem with success probability at least \( 2/3 \) requires \( \tilde{\Omega}(mn) \) queries.

For simplicity, we assume that the algorithm has the knowledge of our construction (which may only strengthens our lower bounds); this includes \( I^* \) and \( D(I^*) \), along with their representation via ElToF and SetToF. The objective of the algorithm is simply to distinguish them. Since we are distinguishing a distribution of instances \( D(I^*) \) against a single instance \( I^* \), we may individually upper bound the probability that each query-answer pair reveals the modified part of the instance, then apply the union bound directly. However, establishing such a bound requires a certain set of properties that we obtain through a careful design of \( I^* \) and \texttt{genModifiedInst}. We remark that our approach shows the hardness of distinguishing instances with with different cover sizes. That is, our lower bound on the query complexity also holds for the problem of approximating the size of the minimum set cover (without explicitly finding one).

In addition, in Section 8.4.4 we provide a construction utilizing Theorem 8.4.2 to extend Corollary 8.4.3, establish the following theorem on lower bounds for larger minimum set cover sizes.

**Theorem 8.4.4.** For \( 1 < \alpha < 3/2 \) and \( k = O(n/\log m) \), any randomized algorithm that computes an \( \alpha \)-approximation to the Set Cover problem with success probability at least \( 2/3 \) requires \( \tilde{\Omega}(mn/k^2) \) queries.

### 8.4.1 Construction of the Median Instance \( I^* \)

Let \( \mathcal{F} \) be a collection of \( m \) sets such that (independently for each set-element pair \((S,e)\)) \( S \) contains \( e \) with probability \( 1 - p_0 \), where \( p_0 = \sqrt{\frac{9 \log m}{n}} \) (note that since we assume \( \log m \leq n/c \) for large enough \( c \), we can assume that \( p_0 \leq 1/2 \)). Equivalently, we may consider the incidence matrix of this instance: each entry is either 0
(indicating $e \notin S$) with probability $p_0$, or 1 (indicating $e \in S$) otherwise. We write \( \mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0) \) denoting the collection of sets obtained from this construction.

**Definition 8.4.5** (Median instance). *An instance of Set Cover, \( I \), is a median instance if it satisfies all the following properties.*

(a) No two sets cover all the elements. (The size of its minimum set cover is at least 3.)

(b) For any two sets the number of elements not covered by the union of these sets is at most $18 \log m$.

(c) The intersection of any two sets has size at least $n/8$.

(d) For any pair of elements $e, e'$, the number of sets $S$ s.t. $e \in S$ but $e' \notin S$ is at least $m \sqrt{\frac{9 \log m}{4n}}$.

(e) For any triple of sets $S, S_1$ and $S_2$, $|(S_1 \cap S_2) \setminus S| \leq 6 \sqrt{n \log m}$.

(f) For each element, the number of sets that do not contain that element is at most $6m \sqrt{\frac{\log m}{n}}$.

**Lemma 8.4.6.** There exists a median instance \( \mathcal{I}^* \) satisfying all properties from Definition 8.4.5. In fact, with high probability, an instance drawn from the distribution in which \( \Pr[e \in S] = 1 - p_0 \) independently at random, satisfies the median properties.

*Proof.* The proof follows from applying the union bound and Lemmas 8.4.7–8.4.12. \( \square \)

**Lemma 8.4.7.** With probability at least $1 - m^{-1}$ over \( \mathcal{F} \sim \mathcal{I}(\mathcal{U}, p_0) \), the size of the minimum set cover of the instance \((\mathcal{F}, \mathcal{U})\) is greater than 2.

*Proof.* The probability that an element $e \in \mathcal{U}$ is covered by two sets selected from \( \mathcal{F} \) is at most:

$$\Pr[e \in S_1 \cup S_2] = 1 - p_0^2 = 1 - \frac{9 \log m}{n}.$$ 

Thus, the probability that $S_1 \cup S_2$ covers all elements in \( \mathcal{U} \) is at most $(1 - \frac{9 \log m}{n})^n < m^{-9}$. Applying the union bound, with probability at least $1 - m^{-1}$ the size of optimal set cover is greater than 2. \( \square \)
Lemma 8.4.8. Let $S_1$ and $S_2$ be two sets in $F$ where $F \sim \mathcal{I}(U, p_0)$. Then with probability at least $1 - m^{-1}$, $|U \setminus (S_1 \cup S_2)| \leq 18 \log m$.

Proof. For an element $e$, $\Pr[e \notin S_1 \cup S_2] = p_0^2 = \frac{9 \log m}{n}$. So, $\mathbb{E}[|U \setminus (S_1 \cup S_2)|] = 9 \log m$. By Chernoff bound, $\Pr[|U \setminus (S_1 \cup S_2)| \geq 18 \log m]$ is at most $e^{-9 \log m/3} \leq m^{-3}$. Thus with probability at least $1 - m^{-1}$, for any pair of sets in $F$, the number of elements not covered by their union is at most $18 \log m$. \qed

Lemma 8.4.9. Let $S_1$ and $S_2$ be two sets in $F$ where $F \sim \mathcal{I}(U, p_0)$. Then $|S_1 \cap S_2| \geq n/8$ with probability at least $1 - m^{-1}$.

Proof. For each element $e$, it is either covered by both $S_1, S_2$, one of $S_1, S_2$ or none of them. Since $p_0 \leq 1/2$, the probability that an element is covered by both sets is greater than other cases, i.e., $\Pr[e \in S_1 \cap S_2] > 1/4$. Thus, $\mathbb{E}[|U \setminus (S_1 \cap S_2)|] > n/4$. By Chernoff bound, $\Pr[|U \setminus (S_1 \cap S_2)| \leq n/8]$ is exponentially small. Thus with probability at least $1 - m^{-1}$, the intersection of any pairs of sets in $F$ is greater than $n/8$. \qed

Lemma 8.4.10. Suppose that $F \sim \mathcal{I}(U, p_0)$ and let $e, e'$ be two elements in $U$. With probability at least $1 - m^{-1}$, the number of sets $S \in F$ such that $e \in S$ but $e' \notin S$ is at least $\frac{m \sqrt{9 \log m}}{4 \sqrt{n}}$.

Proof. For each set $S$, $\Pr[e \in S$ and $e' \notin S] = (1 - p_0)p_0 \geq p_0/2$. This implies that the expected number of $S$ satisfying the condition for $e$ and $e'$ is at least $\frac{m}{2} \cdot \sqrt{\frac{9 \log m}{n}}$ and by Chernoff bound, the probability that the number of sets containing $e$ but not $e'$ is less than $\frac{m \sqrt{9 \log m}}{4 \sqrt{n}}$ is exponentially small. Thus with probability at least $1 - m^{-1}$ property (d) holds for any pair of elements in $U$. \qed

Lemma 8.4.11. Suppose that $F \sim \mathcal{I}(U, p_0)$ and let $S_1, S_2$ and $S$ be sets in $F$. With probability at least $1 - n^{-1}$, $|(S_1 \cap S_2) \setminus S| \leq 6 \sqrt{n \log m}$.

Proof. For each element $e$, $\Pr[e \in (S_1 \cap S_2) \setminus S] = (1 - p_0)^2 p_0 \leq p_0$. This implies that the expected size of $(S_1 \cap S_2) \setminus S$ is less than $\sqrt{9n \log m}$ and by Chernoff bound, the probability that $|(S_1 \cap S_2) \setminus S| \geq 6 \sqrt{n \log m}$ is exponentially small. Thus with probability at least $1 - m^{-1}$ property (e) holds for any sets $S_1, S_2$ and $S$ in $F$. \qed
Lemma 8.4.12. For each element, the number of sets that do not contain the element is at most $6m\sqrt{\log m/n}$.

Proof. For each element $e$, $Pr_S[e \notin S] = p_0$. This implies that $\mathbb{E}_S(\{|S| : e \notin S\})$ is less than $m\sqrt{9\log m/n}$ and by Chernoff bound, the probability that $|\{S : e \notin S\}| \geq 2m\sqrt{9\log m/n}$ is exponentially small. Thus with probability at least $1 - m^{-1}$ property (f) holds for any element $e \in U$. \qed

8.4.2 Distribution $\mathcal{D}(I^*)$ of Modified Instances $I'$ Derived from $I^*$

Fix a median instance $I^*$. We now show that we may perform $O(\log m)$ swap operations on $I^*$ so that the size of the minimum set cover in the modified instance becomes 2. Moreover, its incidence matrix differs from that of $I^*$ in $O(\log m)$ entries. Consequently, the number of queries to $\text{ELTOF}$ and $\text{SETOF}$ that induce different answers from those of $I^*$ is also at most $O(\log m)$.

We define $\mathcal{D}(I^*)$ as the distribution of instances $I'$ generated from a median instance $I^*$ by $\text{genModifiedInst}(I^*)$ given below in 22 as follows. Assume that $I^* = (U, F)$. We select two different sets $S_1, S_2$ from $F$ uniformly at random; we aim to turn these two sets into a set cover. To do so, we swap out some of the elements in $S_2$ and bring in the uncovered elements. For each uncovered element $e$, we pick an element $e' \in S_2$ that is also covered by $S_1$. Next, consider the candidate set that we may exchange its $e$ with $e' \in S_2$:

**Definition 8.4.13 (Candidate set).** For any pair of elements $e, e'$, the candidate set of $(e, e')$ are all sets that contain $e$ but not $e'$. The collection of candidate sets of $(e, e')$ is denoted by $\text{Candidate}(e, e')$. Note that $\text{Candidate}(e, e') \neq \text{Candidate}(e', e)$ (in fact, these two collections are disjoint).

We choose a random set $S$ from $\text{Candidate}(e, e')$, and swap $e \in S$ with $e' \in S_2$ so that $S_2$ now contains $e$. We repeatedly apply this process for all initially uncovered $e$ so that eventually $S_1$ and $S_2$ form a set cover. We show that the proposed algorithm, $\text{genModifiedInst}$, can indeed be executed without getting stuck.
Algorithm 22: The procedure of constructing a modified instance of $I^*$. 

Lemma 8.4.14. The procedure genModifiedInst is well-defined under the precondition that the input instance $I^*$ is a median instance.

Proof. To carry out the algorithm, we must ensure that the number of the initially uncovered elements is at most that of the elements covered by both $S_1$ and $S_2$. This follows from the properties of median instances (Definition 8.4.5): $|U \setminus (S_1 \cup S_2)| \leq 18 \log m$ by property (b), and that the size of the intersection of $S_1$ and $S_2$ is greater than $n/8$ by property (c). That is, in our construction there are sufficiently many possible choices for $e'$ to be matched and swapped with each uncovered element $e$. Moreover, by property (d) there are plenty of candidate sets $S$ for performing swap($e, e'$) with $S_2$. \qed 

Bounding the Probability of Modification

Let $D(I^*)$ denote the distribution of instances generated by genModifiedInst($I^*$). If an algorithm were to distinguish between $I^*$ or $I' \sim D(I^*)$, it must find some cell in the ELTOF or SETOF tables that would have been modified by genModifiedInst, to confirm that genModifiedInst is indeed executed; otherwise it would make wrong decisions half of the time. We will show an additional property of this distribution: none of the entries of ELTOF and SETOF are significantly more likely to be modified during the execution of genModifiedInst. Consequently, no algorithm may strategically detect the difference between $I^*$ or $I'$ with the desired probability, unless the number of queries is asymptotically the reciprocal of the maximum probability of modification among any cells.
Define $P_{\text{Elt-Set}} : \mathbb{U} \times \mathcal{F} \to [0, 1]$ as the probability that an element is swapped by a set. More precisely, for an element $e \in \mathbb{U}$ and a set $S \in \mathcal{F}$, if $e \notin S$ in the median instance $I^*$, then $P_{\text{Elt-Set}}(e, S) = 0$; otherwise, it is equal to the probability that $S$ swaps $e$. We note that these probabilities are taken over $I' \sim \mathcal{D}(I^*)$ where $I^*$ is a fixed median instance. That is, as per (22), they correspond to the random choices of $S_1, S_2$, the random matching $\mathcal{M}$ between $\mathbb{U} \setminus (S_1 \cup S_2)$ and $S_1 \cap S_2$, and their random choices of choosing each candidate set $S$. We bound the values of $P_{\text{Elt-Set}}$ via the following lemma.

**Lemma 8.4.15.** For any $e \in \mathbb{U}$ and $S \in \mathcal{F}$, $P_{\text{Elt-Set}}(e, S) \leq \frac{4800 \log mn}{mn}$ where the probability is taken over $I' \sim \mathcal{D}(I^*)$.

**Proof.** Let $S_1, S_2$ denote the first two sets picked (uniformly at random) from $\mathcal{F}$ to construct a modified instance of $I^*$. For each element $e$ and a set $S$ such that $e \in S$ in the basic instance $I^*$,

\[
P_{\text{Elt-Set}}(e, S) = \Pr[S = S_2] \cdot \Pr[e \in S_1 \cap S_2] \cdot \Pr[e \text{ matches to an element in } \mathbb{U} \setminus (S_1 \cup S_2) \mid e \in S_1 \cap S_2] + \Pr[S \notin \{S_1, S_2\}] \cdot \Pr[e \in S \setminus (S_1 \cup S_2) \mid e \in S] \cdot \Pr[S \text{ swaps } e \text{ with } S_2 \mid e \in S \setminus (S_1 \cup S_2)].
\]

where all probabilities are taken over $I' \sim \mathcal{D}(I^*)$. Next we bound each of the above six terms. Since we choose the sets $S_1, S_2$ randomly, $\Pr[S = S_2] = 1/m$. We bound the second term by 1. For the third term, since we pick a matching uniformly at random among all possible (maximum) matchings between $\mathbb{U} \setminus (S_1 \cup S_2)$ and $S_1 \cap S_2$, by symmetry, the probability that a certain element $e \in S_1 \cap S_2$ is in the matching is (by properties (b) and (c) of median instances),

\[
\frac{|\mathbb{U} \setminus (S_1 \cup S_2)|}{|S_1 \cap S_2|} \leq \frac{18 \log m}{n/8} = \frac{144 \log m}{n}.
\]

We bound the fourth term by 1. To compute the fifth term, let $d_e$ denote the number of sets in $\mathcal{F}$ that do not contain $e$. By property (f) of median instances, the probability
that \( e \in S \) is in \( S \setminus (S_1 \cup S_2) \) given that \( S \notin \{S_1, S_2\} \) is at most,

\[
\frac{d_e(d_e - 1)}{(m - 1)(m - 2)} \leq \frac{36m^2 \cdot \frac{\log m}{n}}{m^2/2} = \frac{72 \log m}{n}.
\]

Finally for the last term, note that by symmetry, each pair of matched elements \( ee' \) is picked by \texttt{genModifiedInst} equiprobably. Thus, for any \( e \in S \setminus (S_1 \cup S_2) \), the probability that each element \( e' \in S_1 \cap S_2 \) is matched to \( e \) is \( \frac{1}{|S_1 \cap S_2|} \). By properties (e)–(e) of median instances, the last term is at most

\[
\sum_{e' \in (S_1 \cap S_2) \setminus S} \Pr[ee' \in M] \cdot \frac{1}{\text{Candidate}(e, e')} = |(S_1 \cap S_2) \setminus S| \cdot \frac{1}{|S_1 \cap S_2|} \cdot \frac{1}{\text{Candidate}(e, e')} \leq 6\sqrt{n \log m} \cdot \frac{1}{n/8} \cdot \frac{1}{m \cdot \sqrt{\log m}} = \frac{64}{m}.
\]

Therefore,

\[
P_{\text{Elt-Set}}(e, S) \leq \frac{1}{m} \cdot \frac{144 \log m}{n} + 1 \cdot \frac{72 \log m}{n} \cdot \frac{64}{m} \leq \frac{4800 \log m}{mn}.
\]

\[\square\]

### 8.4.3 Proof of Theorem 8.4.2

Now we consider a median instance \( I^* \), and its corresponding family of modified sets \( \mathcal{D}(I^*) \). To prove the promised lower bound for randomized protocols distinguishing \( I^* \) and \( I' \sim \mathcal{D}(I^*) \), we apply Yao’s principle and instead show that no deterministic algorithm \( \mathcal{A} \) may determine whether the input is \( I^* \) or \( I' \sim \mathcal{D}(I^*) \) with success probability at least \( 2/3 \) using \( r = o(\frac{mn}{\log m}) \) queries. Recall that if \( \mathcal{A} \)'s query-answer history \( \langle (q_1, a_1), \ldots, (q_r, a_r) \rangle \) when executed on \( I' \) is the same as that of \( I^* \), then \( \mathcal{A} \) must unavoidably return a wrong decision for the probability mass corresponding to \( I' \). We bound the probability of this event as follows.

**Lemma 8.4.16.** Let \( Q \) be the set of queries made by \( \mathcal{A} \) on \( I^* \). Let \( I' \sim \mathcal{D}(I^*) \) where \( I^* \) is a given median instance. Then the probability that \( \mathcal{A} \) returns different outputs
on $I^*$ and $I'$ is at most $\frac{4800 \log m}{mn}|Q|$.

**Proof of 1.** Theorem 8.4.2: If $A$ does not output correctly on $I^*$, the probability of success of $A$ is less than $1/2$; thus, we can assume that $A$ returns the correct answer on $I^*$. This implies that $A$ returns an incorrect solution on the fraction of $I' \sim I'(I^*)$ for which $A(I^*) = A(I')$. Now recall that the distribution in which we apply Yao’s principle consists of $I^*$ with probability $1/2$, and drawn uniformly at random from $D(I^*)$ also with probability $1/2$. Then over this distribution, by Lemma 8.4.16,

$$\Pr[A \text{ returns the correct answer}] \leq 1 - \frac{1}{2} \Pr_{I' \sim D(I^*)}[A(I^*) = A(I')]$$

$$\leq 1 - \frac{1}{2} \left(1 - \frac{4800 \log m}{mn}|Q|\right) = \frac{1}{2} + \frac{2400 \log m}{mn}|Q|.$$ 

Thus, if the number of queries made by $A$ is less than $\frac{mn}{14400 \log m}$, then the probability that $A$ returns the correct answer over the input distribution is less than $2/3$ and the proof is complete.

### 8.4.4 Reduction via Replication for Increasing the Minimum Set Cover Size

We give a simple reduction that extends Corollary 8.4.3 to support instances with arbitrarily large $k$, the size of the minimum set cover. We show the following lemma:

**Lemma 8.4.17.** For any positive integer $t$, if there exists an algorithm $A$ that distinguishes set cover instances with $nt$ elements and $mt$ sets, whether the size of the minimum set cover is $xt$ or $yt$, then there exists an algorithm $A'$ that distinguishes set cover instances with $n$ elements and $m$ sets, whether the size of the minimum set cover is $x$ or $y$, with the same number of queries.

**Proof.** Suppose we wish to construct $A'$ for deciding whether the minimum set cover of an instance $(U, F)$ is $x$ or $y$. We create a new instance $(U^t, F^t)$ that replicates the same set structure $t$ times; we use separate sets and elements for different copies, and keep the same structure, as well as its representation via ELTOF and SETOF
throughout every copy. The size of the minimum set cover of this new instance becomes \(xt\) or \(yt\) correspondingly, and we may use \(A\) to solve \((U', F')\) and retain its decision. \(A'\) may easily simulate the oracle queries for \((U', F')\) made by \(A\): simply remove the copy’s index then make the same queries to its oracle for \((U, F)\).

We apply this lemma to Theorem 8.4.2 by replicating its construction \(\Theta(k)\) times to obtain Theorem 8.4.4, complementing our upper bound of \(\tilde{O}(\frac{mn}{k})\). Due to the restriction that \(p_0 \leq 1/2\), the replication argument applies up to \(k = O(n/\log m)\).

### 8.4.5 Reduction via Independent Copies for the Approximate Set Cover Problem

Our construction of the median instance \(I^*\) and its associated distribution \(\mathcal{D}(I^*)\) of modified instances also leads to the lower bound of \(\tilde{\Omega}(\frac{mn}{k})\) for the problem of computing an approximate solution to Set Cover, in contrast to those for approximating the optimal set cover as established in this section so far. This lower bound matches with the performance of our algorithm for large optimal value \(k\) is tight for some range of value \(k\), albeit it only applies to sufficiently small approximation factor \(\alpha < 7/6\).

Employing Yao’s principle, we construct a distribution of Set Cover instances as follows. Consider the distribution \(\mathcal{D}(I^*)\) of modified instances for Set Cover with \(n\) elements and \(m\) sets, and let \(t\) be a positive integer parameter. We draw \(t\) modified instances \(I'_1, \ldots, I'_t\) independently from \(\mathcal{D}(I^*)\), then combine them in order to create a new instance \((U', F')\) of \(n' \triangleq nt\) elements and \(m' \triangleq mt\) sets, in such a way that each instance \(I'_i\) occupies separate elements and sets. Since each of these instances are disjoint and have an optimal set cover of size 2, the optimal set cover of our full set system is of size \(k' = 2t\). We assume that the algorithm has full knowledge of the construction. Then, the algorithm must, at the very least, be able to compute the optimal set cover of \(I'_i\) in at least a \((3 - 2\alpha)\)-fraction of the \(t\) instances to achieve a set cover of size \(2(3 - 2\alpha) + 3(1 - (3 - 2\alpha)))t = 2\alpha t = \alpha k'\), a \(\alpha\)-approximate solution to our full Set Cover instance.

First, we bound the probability that an algorithm correctly finds the optimal set
Lemma 8.4.18. Let $q$ be the number of queries made by $A$ on $I'_i \sim \mathcal{D}(I^*)$ over $n$ elements and $m$ sets, where $I^*$ is a median instance. Then the probability that $A$ correctly computes the optimal set cover (of size 2) to $I'_i$ is at most $\frac{9600q \log m}{mn}$.

Proof. Our proof is essentially similar to that of Theorem 8.4.2, but instead for this lemma, we consider another modified instance $I'_r \sim \mathcal{D}(I^*)$ then bound the probability that the (deterministic) algorithm successfully distinguishes between $I'_i$ and $I'_r$. Recall that our genModifiedInst procedure picks a pair of sets $S_1, S_2$ uniformly at random, then modifies the instance so that $\{S_1, S_2\}$ forms a set cover. Moreover, by the properties of median instances, no other instances in $\mathcal{D}(I^*)$ shares the exact same optimal set cover. Thus, for the algorithm to compute the optimal set cover of $I'_i$, it must at least be able to determine whether $I'_i$ is the same as $I'_r$.

By Lemma 8.4.15, $P_{\text{Elt-Set}}(e, S) \leq \frac{4800 \log m}{mn}$, where $P_{\text{Elt-Set}}(e, S)$ measures the probability that the pair $(e, S)$ is swapped by genModifiedInst in order to create a random modified instance from $\mathcal{D}(I^*)$. As both $I'_i$ and $I'_r$ are chosen uniformly at random from $\mathcal{D}(I^*)$, the probability that the pair $(e, S)$ differs between these two instances is at most $\frac{9600 \log m}{mn}$ by the union bound. Then, via a similar argument as that of Lemma 8.4.16, our claim follows.

Lemma 8.4.18 above bounds the probability that the algorithm computes an optimal set cover for $I'_i$ with a linear function in $q$, the number of queries. This same function may also represents the probability of obtaining the unique “success” sample by making $q$ draws without replacement from a pool of $\frac{mn}{9600 \log m}$ samples. This is the same type of event we considered when establishing Theorem 8.3.1: detecting swaps in a slab. Moreover, we established a lower bound for detecting swaps in a $(1 - \frac{1}{b})$-fraction of slabs for any $b > 3$. Thus, by directly following the same argument as that of Theorem 8.3.1 (where the number of options for each slab is replaced by $\frac{mn}{9600 \log m}$, the number of slabs is replaced by $t$, and the desired fraction $1 - \frac{1}{b} = 3 - 2\alpha$ for any $b > 3$), we conclude that $\Omega(\frac{mn}{\log m})$ queries are required for the algorithm to compute
optimal set covers for a constant fraction of instances $I'_i$, establishing the following theorem.

**Theorem 8.4.19.** For any sufficiently small approximation factor $\alpha < 7/6$ and $k = O(m/\log n)$, any randomized algorithm that computes an $\alpha$-approximation to the Set Cover problem with success probability at least 0.99 requires $\tilde{\Omega}(mn/k)$ queries.
Bibliography


[XCYH06] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. Extracting redundancy-aware top-k patterns. In *Proc. of the 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 444–453. ACM, 2006.

