Approximate Nearest Neighbor And Its Many Variants

by

Sepideh Mahabadi

B.S., 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

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author . . . .
Department of Electrical Engineering and Computer Science
May 22, 2013

Certified by . . . .
Piotr Indyk
Professor
Thesis Supervisor

Accepted by . . . .
Leslie Kolodziejski
Chairman, Department Committee on Graduate Students

Approximate Nearest Neighbor And Its Many Variants

by

Sepideh Mahabadi

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

This thesis investigates two variants of the approximate nearest neighbor problem.

First, motivated by the recent research on diversity-aware search, we investigate the k-diverse near neighbor reporting problem. The problem is defined as follows: given a query point q, report the *maximum diversity* set S of k points in the ball of radius r around q. The diversity of a set S is measured by the minimum distance between any pair of points in S (the higher, the better). We present two approximation algorithms for the case where the points live in a d-dimensional Hamming space. Our algorithms guarantee query times that are sub-linear in n and only polynomial in the diversity parameter k, as well as the dimension d. For low values of k, our algorithms achieve sub-linear query times even if the number of points within distance r from a query q is linear in n. To the best of our knowledge, these are the first known algorithms of this type that offer provable guarantees.

In the other variant, we consider the approximate line near neighbor (LNN) problem. Here, the database consists of a set of lines instead of points but the query is still a point. Let L be a set of n lines in the d dimensional euclidean space $\mathbb{R}^d$. The goal is to preprocess the set of lines so that we can answer the Line Near Neighbor (LNN) queries in sub-linear time. That is, given the query point $q \in \mathbb{R}^d$, we want to report a line $\ell \in L$ (if there is any), such that $dist(q, \ell) \leq r$ for some threshold value r, where $dist(q, \ell)$ is the euclidean distance between them.

We start by illustrating the solution to the problem in the case where there are only *two* lines in the database and present a data structure in this case. Then we show a recursive algorithm that merges these data structures and solve the problem for the general case of n lines. The algorithm has polynomial space and performs only a logarithmic number of calls to the approximate nearest neighbor subproblem.

Thesis Supervisor: Piotr Indyk
Title: Professor

Acknowledgments

I would like to express my deepest gratitude to my advisor Piotr Indyk for his excellent guidance, caring, engagement, patience, and for providing me with the right set of tools throughout this work.

Some of the results of this work has appeared in SoCG 2013 and WWW 2013. I would like to thank my coauthors Sihem Amer-Yahia, Sofiane Abbar, Piotr Indyk, and Kasturi R. Varadarajan for their contributions in these publications.

I am also very grateful to my parents and brother for their endless love. I owe much of my academic success to their continuous encouragement and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The *Nearest Neighbor* problem is a fundamental geometry problem which is of major importance in several areas such as databases and data mining, information retrieval, image and video databases, pattern recognition, statistics and data analysis. The problem is defined as follows: given a collection of n points, build a data structure which, given any query point q, reports the data point that is closest to the query. 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 mentioned applications, the features of each object of interest (document, image, etc.) are represented as a point in $\mathbb{R}^d$ and the distance metric is used to measure the similarity of 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 1000×1000 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 [11], and remains one of the most popular data structures used for searching in multidimensional spaces. Many other multidimensional data structures are known, see [30] 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 a query to each point from the database. This phenomenon is often called “the curse of dimensionality”.

In recent years, several researchers have proposed methods for overcoming the running time bottleneck by using approximation (e.g., [9, 24, 22, 26, 20, 25, 13, 12, 28, 5], see also [31, 21]). In this formulation, the algorithm is allowed to return a point whose distance from the query is at most c times the distance from the query to its nearest points; $c > 1$ is called the *approximation factor*. The appeal of this approach 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 user quality, then small differences in the distance should not matter. Moreover, an efficient approximation algorithm can be used to solve the exact nearest neighbor problem by enumerating all approximate nearest neighbors and choosing the closest point.

The *Near Neighbor* Problem is the decision version of the nearest neighbor problem, in which 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 is any). In the *Approximate Near Neighbor Problem*, we the goal is to output any point within distance cr of the query point q, if there is any point within distance r of q. In the case where $c = 1 + \epsilon$, that is when the data structure is allowed to report any point within distance $r(1+\epsilon)$, efficient solutions exist for this problem in high dimensions. In particular, several data structures with query time of $(d + \log n + 1/\epsilon)^{O(1)}$ using $n^{(1/\epsilon)^{O(1)}}$ space are known. [22, 26, 20].

1.1 Our results

In this thesis, we investigate two variants of the approximate nearest neighbor problem, namely the *diverse near neighbor problem* and the *line near neighbor problem*. In the diverse near neighbor problem, we are given an additional output size parameter k. Given a query point q, the goal is to report the *maximum diversity* set S of k points in the ball of radius r around q. The diversity of a set S is measured by the minimum distance between any pair of points in S. The line near neighbor problem is another natural variation of the near neighbor problem in which the database consists of a set of *lines* instead of a set of points, and given a query point q, the goal is to report a line whose distance to the query is at most r (if one exists).

In Chapter 2, we present two efficient approximate algorithms for the *k-diverse near neighbor* problem. The key feature of our algorithms is that they guarantee query times that are sub-linear in n and polynomial in the diversity parameter k and the dimension d, while at the same time providing constant factor approximation guarantees[1] for the diversity objective. Note that for low values of k our algorithms have sub-linear query times *even if* the number of points within distance r from q is linear in n. To the best of our knowledge, these are the first known algorithms of this type with provable guarantees. One of the algorithms (Algorithm A) is closely related to algorithms investigated in applied works [2, 32]. However, those papers did not provide rigorous guarantees on the answer quality. The results of our work on this problem are published in [2, 1].

The line near neighbor problem is studied in Chapter 3. The problem has been previously investigated in [10, 27]. The best known algorithm for this problem achieved a very fast query time of $(d+\log n+1/\epsilon)^{O(1)}$, but the space requirement of the algorithm was super-polynomial, of the form $2^{(\log n)^{O(1)}}$. In contrast, our algorithm has space bound that is *polynomial* in n, d, $\log \Delta$ and super-exponential in $(1/\epsilon)$, and achieves the query time of $(d + \log n + \log \Delta + 1/\epsilon)^{O(1)}$ where we assume that the input is contained in a box $[0, \Delta]^d$. This is the first non-trivial algorithm with polynomial space for this problem.

We start the description by providing an efficient algorithm for the case where we have only two lines in the database. It is achieved by considering two exhaustive cases: one when where the two lines are almost parallel to each other, and the case where the two lines are far from being parallel. In both cases the problem is reduced to a set of approximate point nearest neighbor data structures. Then we show how

[1]Note that approximating the diversity objective is inevitable, since it is NP-hard to find a subset of size k which maximizes the diversity with approximation factor $a < 2$ [29].

to merge the data structures constructed for each pair of lines to get an efficient algorithm for the general case.

Chapter 2

Diverse Near Neighbor Problem

The near neighbor reporting problem (a.k.a. range query) is defined as follows: given a collection P of n points, build a data structure which, given any query point, reports all data points that are within a given distance r to the query. The problem is of major importance in several areas, such as databases and data mining, information retrieval, image and video databases, pattern recognition, statistics and data analysis. In those application, the features of each object of interest (document, image, etc) are typically represented as a point in a d-dimensional space and the distance metric is used to measure similarity of 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 thousands.

One of the major issues in similarity search is how many answers to retrieve and report. If the size of the answer set is too small (e.g., it includes only the few points closest to the query), the answers might be too homogeneous and not informative [14]. If the number of reported points is too large, the time needed to retrieve them is high. Moreover, long answers are typically not very informative either. Over the last few years, this concern has motivated a significant amount of research on diversity-aware search [16, 36, 8, 23, 35, 34, 15] (see [14] for an overview). The goal of that work is to design efficient algorithms for retrieving answers that are both *relevant* (e.g., close to the query point) and *diverse*. The latter notion can be defined in several ways. One of the popular approaches is to cluster the answers and return only the cluster

	Algorithm A	Algorithm B
Distance approx factor	$c > 2$	$c > 1$
Diversity approx factor	6	6
Space	$O((n \log k)^{1+\frac{1}{c-1}} + nd)$	$O(\log k \cdot n^{1+\frac{1}{c}} + nd)$
Query Time	$O((k^2 + \frac{\log n}{r})d \cdot (\log k)^{\frac{c}{c-1}} \cdot n^{\frac{1}{c-1}})$	$O((k^2 + \frac{\log n}{r})d \cdot \log k \cdot n^{1/c})$

Table 2.1: Performance of our algorithms

centers [14, 16, 32, 2]. This approach however can result in high running times if the number of relevant points is large.

Our results In this chapter we present two efficient approximate algorithms for the k*-diverse near neighbor* problem. The problem is defined as follows: given a query point, report the *maximum diversity* set S of k points in the ball of radius r around q. The diversity of a set S is measured by the minimum distance between any pair of points in S. In other words, the algorithm reports the approximate solution to the k-center clustering algorithm applied to the list points that are close to the query. The running times, approximation factors and the space bounds of our algorithms are given in Table 2.1. Note that the Algorithm A is dominated by Algorithm B; however, it is simpler and easier to analyze and implement, and we have used it in applications before for diverse news retrieval [2].

The key feature of our algorithms is that they guarantee query times that are sub-linear in n and polynomial in the diversity parameter k and the dimension d, while at the same time providing constant factor approximation guarantees[1] for the diversity objective. Note that for low values of k our algorithms have sub-linear query times *even if* the number of points within distance r from q is linear in n. To the best of our knowledge, these are the first known algorithms of this type with provable guarantees. One of the algorithms (Algorithm A) is closely related to algorithms investigated in applied works [2, 32]. However, those papers did not provide rigorous guarantees on the answer quality.

[1]Note that approximating the diversity objective is inevitable, since it is NP-hard to find a subset of size k which maximizes the diversity with approximation factor $a < 2$ [29].

2.0.1 Past work

In this section we present an overview of past work on (approximate) near neighbor and diversity aware search that are related to the results in this chapter.

Near neighbor The near neighbor problem has been a subject of extensive research. There are several efficient algorithms known for the case when the dimension d is "low". However, despite decades of intensive effort, the current solutions suffer from either space or query time that is exponential in d. Thus, in recent years, several researchers proposed methods for overcoming the running time bottleneck by using approximation. In the approximate near neighbor reporting/range query, the algorithm must output *all* points within the distance r from q, and can also output *some* points within the distance cr from q.

One of the popular approaches to near neighbor problems in high dimensions is based on the concept of *locality-sensitive hashing* (LSH) [18]. The idea is to hash the points using several (say L) hash functions so as to ensure that, for each function, the probability of collision is much higher for objects which are close to each other than for those which are far apart. Then, one can solve (approximate) near neighbor reporting by hashing the query point and retrieving all elements stored in buckets containing that point. This approach has been used e.g., for the E2LSH package for high-dimensional similarity search [7].

The LSH algorithm has several variants, depending on the underlying distance functions. In the simplest case when the dis-similarity between the query points is defined by the Hamming distance, the algorithm guarantees that (i) each point within the distance r from from q is reported with a constant (tunable) probability and (ii) the query time is at most $O(d(n^{1/c} + |P_{cr}(q)|))$, where $P_R(q)$ denotes the set of points in P with the distance R from q. Thus, if the size of the answer set $P_{cr}(q)$ is large, the efficiency of the algorithm decreases. Heuristically, a speedup can be achieved [32] by clustering the points in each bucket and retaining only the cluster centers. However, the resulting algorithm did not have any guarantees (until now).

Diversity In this work we adopt the "content-based" definition of diversity used e.g., in [14, 16, 32, 2, 15]. The approach is to report k answers that are "sufficiently different" from each other. This is formalized as maximizing the minimum distance between any pair of answers, the average distance between the answers, etc. In this thesis we use the minimum distance formulation, and use the greedy clustering algorithm of [17, 29] to find the k approximately most diverse points in a given set.

To the best of our knowledge, the only prior work that explicitly addresses our definition of the k-diverse near neighbor problem is [2]. It presents an algorithm (analogous to Algorithm A in this paper, albeit for the Jaccard coefficient as opposed to the Hamming metric) and applies it to problems in news retrieval. However, that paper does not provide any formal guarantees on the accuracy of the reported answers.

2.0.2 Our techniques

Both of our algorithms use LSH as the basis. The key challenge, however, is in reducing the dependence of the query time on the size of the set $P_{cr}(q)$ of points close to q. The first algorithm (Algorithm A) achieves this by storing only the k most diverse points per each bucket. This ensures that the total number of points examined during the query time is at most $O(kL)$, where L is the number of hash functions. However, proving the approximation guarantees for this algorithm requires that no outlier (i.e., point with distance $> cr$ from q) is stored in any bucket. Otherwise that point could have been selected as one of the k diverse points for that bucket, replacing a "legitimate" point. This requirement implies that the algorithm works only if the distance approximation factor c is greater than 2.

The 6-approximation guarantee for diversity is shown by using the notion of *core-sets* [4]. It is easy to see that the maximum k-diversity of a point set is within a factor of 2 from the optimal cost of its $(k-1)$-center clustering cost. For the latter problem it is known how to construct a small subset of the input point set (a *coreset*) such that for any set of cluster centers, the costs of clustering the coreset is within a constant factor away from the cost of clustering the whole data set. Our algorithm then simply computes and stores only a coreset for each LSH bucket. Standard coreset properties imply that the union of coresets for the buckets touched by the query point q is a coreset for all points in those buckets. Thus, the union of all coresets provides a sufficient information to recover an approximately optimal solution to all points close to q.

In order to obtain an algorithm that works for arbitrary $c > 1$, we need the algorithms to be robust to outliers. The standard LSH analysis guarantees that that the number of outliers in all buckets is at most $O(L)$. Algorithm B achieves the robustness by storing a *robust* coreset [19, 3], which can tolerate some number of outliers. Since we do not know a priori how many outliers are present in any given bucket, our algorithm stores a sequence of points that represents a coreset robust to an increasing number of outliers. During the query time the algorithm scans the list until enough points have been read to ensure the robustness.

2.1 Problem Definition

Let $(\Delta, dist)$ be a d-dimensional metric space. We start from two definitions.

Definition 2.1.1. *For a given set $S \in \Delta$, its* ***diversity*** *is defined as the minimum pairwise distance between the points of the set, i.e., $div(S) = \min_{p,p' \in S} dist(p, p')$*

Definition 2.1.2. *For a given set $S \in \Delta$, its* ***k-diversity*** *is defined as the maximum achievable diversity by choosing a subset of size k, i.e., $div_k(S) = \max_{S' \subset S, |S'|=k} div(S')$. We also call the maximizing subset S' the* ***optimal k-subset*** *of S. Note that k-diversity is not defined in the case where $|S| < k$.*

To avoid dealing with k-diversity of sets of cardinality smaller than k, in the following we adopt the convention that all points p in the input point set P are duplicated k times. This ensures that for all non-empty sets S considered in the rest of this paper the quantity $div_k(S)$ is well defined, and equal to 0 if the number of distinct points in S is less than k. It can be easily seen that this leaves the space bounds of our algorithms unchanged.

The k-**diverse Near Neighbor Problem** is defined as follows: given a query point q, report a set S such that: (i) $S \subset P \cap B(q,r)$, where $B(q,r) = \{p | dist(p,q) \leq r\}$ is the ball of radius r, centered at q; (ii) $|S| = k$; (iii) $div(S)$ is maximized.

Since our algorithms are approximate, we need to define the **Approximate k-diverse Near Neighbor Problem**. In this case, we require that for some approximation factors $c > 1$ and $\alpha > 1$: (i) $S \subset P \cap B(q, cr)$; (ii) $|S| = k$; (iii) $div(S) \geq \frac{1}{\alpha} div_k(P \cap B(q,r))$.

2.2 Preliminaries

2.2.1 GMM Algorithm

Suppose that we have a set of points $S \subset \Delta$, and want to compute an optimal k-subset of S. That is, to find a subset of k points, whose pairwise distance is maximized. Although this problem is NP-hard, there is a simple 2-approximate greedy algorithm [17, 29], called GMM.

In this work we use the following slight variation of the GMM algorithm [2]. The algorithm is given 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. Note that the convention that all points have k duplicates implies that if the input point set S contains less than k distinct points, then the output S' contains all of those points.

Lemma 2.2.1. *The running time of the algorithm is $O(k \cdot |S|)$, and it achieves an approximation factor of at most 2 for the k-diversity $div_k(S)$.*

[2]The proof of the approximation factor this variation achieves is virtually the same as the proof in [29]

Algorithm 1 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 $i = 2 \to k$ do
3:     find $p \in S \setminus S'$ which maximizes $\min_{x \in S'} dist(p, x)$
4:     $S' \leftarrow S' \cup \{p\}$
5: end for
6: return $S'$
```

2.2.2 Coresets

Definition 2.2.2. *Let $(P, dist)$ be a metric. For any subset of points $S, S' \subset P$, we define the k-center cost, $KC(S, S')$ as $\max_{p \in S} min_{p' \in S'} dist(p, p')$. The **Metric k-center Problem** is defined as follows: given S, find a subset $S' \subset S$ of size k which minimizes $KC(S, S')$. We denote this optimum cost by $KC_k(S)$.*

k-diversity of a set S is closely related to the cost of the best $(k-1)$-center of S. That is,

Lemma 2.2.3. $KC_{k-1}(S) \leq div_k(S) \leq 2KC_{k-1}(S)$

Proof. For the first inequality, suppose that S' is the optimal k-subset of S. Also let $a \in S'$ be an arbitrary point and $S'_{-} = S' \setminus \{a\}$. Then for any point $b \in S \setminus S'$, we have $min_{p \in S'_{-}} dist(b, p) \leq min_{p \in S'_{-}} dist(a, p)$, otherwise b was a better choice than a, i.e., $div(b \cup S'_{-}) > div(S')$. Therefore, $KC(S, S'_{-}) \leq div_k(S)$ and the inequality follows.

For the second part, let $C = \{a_1, \cdots, a_{k-1}\}$ be the optimum set of the $(k-1)$-center for S. Then since S' has size k, by pigeonhole principle, there exists $p, p' \in S'$ and a, such that

$$a = \arg\min_{c \in C} dist(p, c) = \arg\min_{c \in C} dist(p', c)$$

and therefore, by triangle inequality we get

$$\begin{aligned} div_k(S) = div(S') \le dist(p, p') &\le dist(p, a) + dist(a, p') \\ &\le 2KC_{k-1}(S) \end{aligned}$$

□

Definition 2.2.4. *Let $(P, dist)$ be our metric. Then for $\beta \le 1$, we define a **β-coreset** for a point set $S \subset P$ to be any subset $S' \subset S$ such that for any subset of $(k-1)$ points $F \subset P$, we have $KC(S', F) \ge \beta KC(S, F)$.*

Definition 2.2.5. *Let $(P, dist)$ be our metric. Then for $\beta \le 1$ and an integer ℓ, we define an **ℓ-robust β-coreset** for a point set $S \subset P$ to be any subset $S' \subset S$ such that for any set of outliers $O \subset P$ with at most ℓ points, $S' \setminus O$ is a β-coreset of $S \setminus O$.*

2.2.3 Locality Sensitive Hashing

Locality-sensitive hashing is a technique for solving approximate near neighbor problems. The 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 2.2.6. *A family $\mathcal{H} = h : \Delta \to U$ is (r_1, r_2, p_1, p_2)-sensitive for $(\Delta, dist)$, if for any $p, q \in \Delta$, we have*

- *if $dist(p, q) \le r_1$, then $Pr_{\mathcal{H}}[h(q) = h(p)] \ge p_1$*
- *if $dist(p, q) \le r_2$, then $Pr_{\mathcal{H}}[h(q) = h(p)] \le p_2$*

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 $g_1, g_2, \cdots, g_L$, as well as the corresponding hash arrays $A_1, A_2, \cdots, A_L$. Each hash function is of the form $g_i =< h_{i,1}, \cdots, h_{i,K} >$, where $h_{i,j}$ is chosen uniformly at random from $\mathcal{H}$. Then each point p is stored in bucket $g_i(p)$ of A_i for all $1 \le i \le L$. 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, from each array, we only have to look into the single bucket which corresponds to the query point q.

In this paper, for simplicity, we consider the LSH for the Hamming distance. However, similar results can be shown for general LSH functions. We recall the following lemma from [18].

Lemma 2.2.7. *Let $dist(p, q)$ be the Hamming metric for $p, q \in \Sigma^d$, where Σ is any finite alphabet. Then for any $r, c \geq 1$, there exists a family $\mathcal{H}$ which is $(r, rc, p_1, p_2)-$ sensitive, where $p_1 = 1 - r/d$ and $p_2 = 1 - rc/d$. Also, if we let $\rho = \frac{\log 1/p_1}{\log 1/p_2}$, then we have $\rho \leq 1/c$. Furthermore, by padding extra zeros, we can assume that $r/d \leq 1/2$.*

2.3 Algorithm A

The algorithm (first introduced in [2]) is based on the LSH algorithm. During the preprocessing, LSH creates L hash functions $g_1, g_2, \cdots, g_L$, and the arrays $A_1, A_2, \cdots, A_L$. Then each point p is stored in buckets $A_i[g_i(p)]$, for all $i = 1 \cdots L$. Furthermore, for each array A_i, the algorithm uses GMM to compute a 2-approximation of the optimal k-subset of each bucket, and stores it in the corresponding bucket of A'_i. This computed subset turns out to be a 1/3-coreset of the points of the bucket.

Given a query q, the algorithm computes the union of the buckets $Q = A'_1(g_1(q)) \cup \cdots \cup A'_L(g_L(q))$, and then it removes from Q all **outlier** points, i.e., the points which are not within distance cr of q. In the last step, the algorithm runs GMM on the set Q and returns the approximate optimal k-subset of Q.

The pseudo codes are shown in Algorithm 2 and 3. In the next section we discuss why this algorithm works.

2.3.1 Analysis

In this section, first we determine the value of the parameters L and K in terms of n and $\rho \leq 1/c$, such that with constant probability, the algorithm works. Here, L is the total number of hash functions used, and K is the number of hash functions $h_{i,j}$ used

Algorithm 2 Preprocessing

Input $G = \{g_1, \cdots, g_L\}$: set of L hashing functions, P: collection of points, k
Output $A' = \{A'_1, \cdots, A'_L\}$

```
 1: for all points $p \in P$ do
 2:     for all hash functions $g_i \in G$ do
 3:         add $p$ to the bucket $A_i[g_i(p)]$
 4:     end for
 5: end for
 6: for $A_i \in A$ do
 7:     for $j = 1 \to size(A_i)$ do
 8:         $A'_i[j] = GMM(A_i[j], k)$ // only store the approximate k-diverse points in
            each bucket
 9:     end for
10: end for
```

in each of the g_i. We also need to argue that limiting the size of the buckets to k, and storing only the approximate k most diverse points in A', works well to achieve a good approximation. We address these issues in the following.

Lemma 2.3.1. *For $c > 2$, There exists hash functions $g_1, \cdots, g_L$ of the form $g_i =< h_{i,1}, \cdots, h_{i,K} >$ where $h_{i,j} \in \mathcal{H}$, for $\mathcal{H}$, p_1 and p_2 defined in 2.2.7, such that by setting $L = (\log(4k)/p_1)^{1/(1-\rho)} \times (4n)^{\rho/(1-\rho)}$, and $K = \lceil \log_{1/p_2}(4nL) \rceil$, the following two events hold with constant probability:*

- *$\forall p \in Q^* : \exists i$ such that $p \in A_i[g_i(q)]$, where Q^* denotes the optimal solution (the optimal k-subset of $P \cap B(q, r)$).*
- *$\forall p \in \bigcup_i A_i[g_i(q)] : dist(p, q) \leq cr$, i.e., there is no outlier among the points hashed to the same bucket as q in any of the hash functions.*

Proof. For the first argument, consider a point $p \in Q^*$. By Definition 2.2.6 the probability that $g_i(p) = g_i(q)$ for a given i, is bounded from below by

$$p_1^K \geq p_1^{\log_{1/p_2}(4nL)+1} = p_1(4nL)^{-\frac{\log 1/p_1}{\log 1/p_2}} = p_1(4nL)^{-\rho}$$

Algorithm 3 Query Processing

Input q: The query point, k
Output Q : The set of k-diverse points.

```
 1: $Q \leftarrow \emptyset$
 2: for $i = 1 \rightarrow L$ do
 3:     $Q \leftarrow Q \cup A'_i[g_i(q)]$
 4: end for
 5: for all $p \in Q$ do
 6:     if $dist(q, p) > cr$ then
 7:         remove $p$ from Q // since it is an outlier
 8:     end if
 9: end for
10: $Q \leftarrow GMM(Q, k)$
11: return $Q$
```

Thus the probability that no such g_i exists is at most

$$\zeta = (1 - p_1(4nL)^{-\rho})^L \leq (1/e)^{L \cdot \frac{p_1}{(4nL)^\rho}} = (1/e)^{L^{(1-\rho)} \cdot \frac{p_1}{(4n)^\rho}}$$
$$= (1/e)^{(\log(4k)/p_1(4n)^\rho) \cdot \frac{p_1}{(4n)^\rho}} \leq \frac{1}{4k}$$

Now using union bound, the probability that $\forall p \in Q^* : \exists i,$ such that $p \in A_i[g_i(q)]$ is at least $\frac{3}{4}$.

For the second part, note that the probability that $g_i(p) = g_i(q)$ for $p \in P \backslash B(q, cr)$ is at most $p_2^K = \frac{1}{4nL}$. Thus, the expected number of elements from $P \backslash B(q, cr)$ colliding with q under fixed g_i is at most $\frac{1}{4L}$, and the expected number of collisions in all g functions is at most $\frac{1}{4}$. Therefore, with probability at least $\frac{3}{4}$, there is no outlier in $\bigcup_i A_i[g_i(q)]$.

So both events hold with probability at least $\frac{1}{2}$. □

Corollary 2.3.2. *Since each point is hashed once in each hash function, the total space used by this algorithm is at most*

$$nL = n(\log(4k)/p_1(4n)^\rho)^{1/(1-\rho)} = O((\frac{n \log k}{1 - r/d})^{\frac{1}{1-\rho}})$$
$$= O((n \log k)^{1+\frac{1}{c-1}})$$

where we have used the fact that $c > 2$, $\rho \leq 1/c$, and $r/d \leq 1/2$. Also we need $O(nd)$ space to store the points.

Corollary 2.3.3. *The query time is $O(((\log n)/r + k^2) \cdot (\log k)^{\frac{c}{c-1}} \cdot n^{\frac{1}{c-1}} d)$.*

Proof. The query time of the algorithm for each query is bounded by $O(L)$ hash computation each taking $O(K)$

$$\begin{aligned}
O(KL) &= O((\log_{1/p_2}(4nL)) \cdot L) = O(\frac{\log n}{\log(1/p_2)} L) \\
&= O(\frac{d}{r} \log n \cdot (\frac{\log k}{1 - r/d})^{\frac{c}{c-1}} \cdot n^{\frac{1}{c-1}}) \\
&= O(\frac{d}{r} (\log k)^{\frac{c}{c-1}} n^{\frac{1}{c-1}} \log n)
\end{aligned}$$

Where we have used the approximation $\log p_2 \approx 1 - p_2 = \frac{cr}{d}$, $c \geq 2$ and $r/d \leq 1/2$.

Also in the last step, we need to run the GMM algorithm for at most kL number of points in expectation. This takes

$$\begin{aligned}
O(k^2 L d) &= O(k^2 \cdot (\log k / p_1 (4n)^{\rho})^{1/(1-\rho)} d) \\
&= O(k^2 (\log k)^{\frac{c}{c-1}} \cdot n^{\frac{1}{c-1}} d)
\end{aligned}$$

□

Lemma 2.3.4. *$GMM(S,k)$ computes a $1/3$-coreset of S.*

Proof. Suppose that the set of k points computed by GMM is S'. Now take any subset of $k-1$ points $F \subset P$. By pigeonhole principle there exist $a, b \in S'$ whose closest point in F is the same, i.e., there exists $c \in F$, such that

$$c = \arg\min_{f \in F} dist(a, f) = \arg\min_{f \in F} dist(b, f)$$

and therefore, by triangle inequality we get

$$div(S') \leq dist(a, b) \leq dist(a, c) + dist(b, c) \leq 2KC(S', F)$$

Now take any point $s \in S$ and let s' be the closest point of S' to s and f be the closest point of F to s'. Also let $a \in S'$ be the point added in the last step of the GMM algorithm. Then from definitions of s' and a, and the greedy choice of GMM, we have

$$dist(s, s') \leq \min_{p \in S' \setminus \{a\}} dist(p, s) \leq \min_{p \in S' \setminus \{a\}} dist(p, a) \leq div(S')$$

and thus by triangle inequality,

$$\begin{aligned} dist(s, f) &\leq dist(s, s') + dist(s', f) \leq div(S') + KC(S', F) \\ &\leq 3KC(S', F) \end{aligned}$$

Since this holds for any $s \in S$, we can infer that $KC(S, F) \leq 3KC(S', F)$ which completes the proof. □

Lemma 2.3.5. *Suppose $S_1, \cdots, S_m$ are subsets of P, and let $S = \bigcup_i S_i$. Also suppose that $T_i = GMM(S_i, k)$ is the 2-approximation of the optimal k-subset of S_i which is achieved by running the GMM algorithm on S_i. Also define $T = \bigcup_i T_i$, and let $T' = GMM(T, k)$ be the 2-approximation of the optimal k-subset of T. Then we have $div(T') \geq \frac{1}{6} div_k(S)$.*

Proof. Let S' denote the optimal k-subset of S. Also let $a \in T'$ be the added point at the last step of algorithm $GMM(T, k)$ and $T'_- = T' \setminus \{a\}$. By pigeonhole principle there exists $p, p' \in S'$ and $c \in T'_-$ such that

$$c = \arg\min_{t \in T'_-} dist(t, p) = \arg\min_{t \in T'_-} dist(t, p')$$

Therefore, by triangle inequality, lemma 2.3.4, and the fact that union of β-coresets of S_i is a β-coreset for the union S, we have

$$\begin{aligned} div_k(S) = div(S') &\leq dist(p, p') \leq dist(p, c) + dist(p', c) \\ &\leq 2KC(S, T'_-) \leq 6KC(T, T'_-) \end{aligned}$$

And since for any $b \in T$ we have

$$\min_{t \in T'_{-}} dist(t, b) \le \min_{t \in T'_{-}} dist(t, a) \le div(T')$$

And thus, $KC(T, T'_{-}) \le div(T')$ and the lemma follows. □

Corollary 2.3.6. *With constant probability, the approximation factor achieved by the algorithm is 6.*

Proof. Let $S_i = A_i[g_i(q)]$ and $S = \bigcup_i S_i$. Also let $T_i = A'_i[g_i(q)]$ and $T = \bigcup_i T_i$. Furthermore define Q^* be the optimal k-subset of $P \cap B(q, r)$, $T' = \text{GMM}(T, k)$ and Q be our returned set. From the description of the algorithm, it is obvious that $Q \subset B(q, cr)$. So, we only have to argue about its diversity.
By Theorem 2.3.1, with constant probability the two following statements hold:

- $S \subset B(q, cr)$. Therefore, we have $T \subset B(q, cr)$, which shows that when we run the Algorithm 3 for q, since T contains no outlier, the GMM algorithm in Line 10 is called on the set T itself and thus, $Q = T'$.
- $Q^* \subset S$. So we have $div_k(S) \ge div_k(Q^*) = div(Q^*)$

Therefore, by Lemma 2.3.5 we have

$$div(Q) \ge \frac{1}{6} div_k(S) \ge \frac{1}{6} div(Q^*)$$

□

2.4 Algorithm B

In this section, we introduce and analyze a modified version of Algorithm A which also achieves a constant factor approximation. Suppose that we knew the total number of outliers in any bucket is at most ℓ. Then, we can store for each single bucket of the array A, an ℓ-robust $\frac{1}{3}$-coreset in the corresponding bucket of array A'. First we show in Algorithm 4, how to find an ℓ-robust β-coreset if we know how to find a β-coreset.

This is the algorithm of [3] that "peels" coresets β-coresets, and its analysis follows [33].

Algorithm 4 (ℓ, β)-coreset

Input S: set of points
Output S': An array which is a (ℓ, β)-coreset of S

```
1: $S' \leftarrow \emptyset$
2: for $i = 1 \rightarrow (\ell + 1)$ do
3:     $R_i \leftarrow \beta$-coreset of $S$
4:     Append $R_i$ to the end of $S'$
5:     $S \leftarrow S \setminus R_i$
6: end for
7: return $S'$
```

Lemma 2.4.1. *Let $O \subset P$ be the set of outliers and S'_j denote set of the first (kj) points in S' which is S' after the jth round of the algorithm. Then for any $0 \leq j \leq \ell$ that satisfies $\left|S'_{j+1} \cap O\right| \leq j$, we have that $S'_{j+1} \setminus O$ is a β-coreset for $S \setminus O$.*

Proof. Let $F \subset P$ be any subset of $(k-1)$ points, and q be the furthest point from F in $(S \setminus O)$, i.e.,

$$q = \arg\max_{p \in S \setminus O} \min_{f \in F} dist(p, f)$$

Now for any $i \leq (j+1)$, if q is a point in R_i, then the lemma holds since $KC(S, F) = KC(S'_{j+1}, F)$. Otherwise, because q has not been chosen in any of the first $(j+1)$ rounds, each of the R_i's (for $i \leq j+1$) contains an r_i such that $KC(\{r_i\}, F) \geq \beta KC(\{q\}, F)$. Of these $(j+1) r_i$'s, at least one is not in O and therefore, $KC(S'_{j+1} \setminus O, F) \geq \beta KC(S \setminus O, F)$. □

Corollary 2.4.2. *Algorithm 4 computes the (ℓ, β)-coreset of S correctly.*

Proof. Note that here for any set of outliers $O \subset P$ such that $|O| \leq \ell$, the condition in lemma 2.4.1 is satisfied for $j = \ell$. Thus when the algorithm returns, it has computed an ℓ-robust β-coreset correctly. □

Since by lemma 2.3.4, we know that GMM(S, k) computes a $\frac{1}{3}$-coreset of size k for S, it is enough to replace line 3 of the algorithm 4 with $R \leftarrow GMM(S, k)$, in order to achieve an ℓ-robust $\frac{1}{3}$-coreset of size $k(\ell + 1)$ for the set S.

Then the only modification to the preprocessing part of Algorithm A is that now, each bucket of A'_i keeps an ℓ-robust $\frac{1}{3}$-coreset of the corresponding bucket of A_i. So the line 8 of Algorithm 2 is changed to $A'_i[j] = (\ell, \beta)\text{-coreset}(A_i[j], \ell = 3L, \beta = 1/3)$.

The pseudo-code of processing a query is shown in Algorithm 5. For each bucket that corresponds to q, it tries to find the smallest value of ℓ such that the total number of outliers in the first $k(\ell+1)$ elements of $A'_i[g_i(q)]$ does not exceed ℓ. It then adds these set of points to T and returns the approximate optimal k subset of non-outlier points of T.

Algorithm 5 Query Processing

Input q: The query point
Output Q : The set of k-diverse points.

```
1: $T \leftarrow \emptyset$
2: $O \leftarrow$ set of outliers
3: for $i = 1 \to L$ do
4:     for $\ell = 0 \to 3L$ do
5:         $U_i^{\ell+1}$ = the first $k(\ell+1)$ points of $A'_i[g_i(q)]$
6:         if $\left|U_i^{\ell+1} \cap O\right| \leq \ell$ then
7:             $\ell_i \leftarrow \ell$
8:             $T_i \leftarrow U_i^{\ell+1}$
9:             break
10:        end if
11:    end for
12:    $T \leftarrow T \cup T_i$
13: end for
14: $Q \leftarrow GMM(T \setminus O, k)$
15: return $Q$
```

Note that the inner loop (lines 4 to 11) of Algorithm 5 can be implemented efficiently. Knowing the number of outliers in U_i^j, there are only k more elements to check for being outliers in U_i^{j+1}. Also, each point can be checked in $O(d)$ if it is an outlier. So in total the inner loop takes $O(k\ell_i d)$.

2.4.1 Analysis

We first state the following theorem which is similar to Theorem 2.3.1. The proof is very similar to the original proof of correctness of the LSH algorithm given in [18].

Theorem 2.4.3. *There exists hash functions $g_1, \cdots, g_L$ of the form $g_i =< h_{i,1}, \cdots, h_{i,K} >$ where $h_{i,j} \in \mathcal{H}$, for $\mathcal{H}$, p_1 and p_2 defined in 2.2.7 such that by setting $L = \log(4k) \times n^\rho / p_1$, and $K = \lceil \log_{1/p_2} n \rceil$, with constant probability the following two events hold:*

- *$\forall p \in Q^* : \exists i$ such that $p \in A_i[g_i(q)]$, where Q^* denotes the optimal solution (the optimal k-subset of $P \cap B(q, r)$).*
- *$|\{p \in \bigcup_i A_i[g_i(q)] : dist(p, q) > cr\}| \leq 3L$, i.e. the number of outliers among points hashed to the same bucket as q, is at most $3L$.*

Proof. For the first argument, consider a point $p \in Q^*$, the probability that $g_i(p) = g_i(q)$ for a given i is bounded from below by

$$p_1^K \geq p_1^{\log_{1/p_2} n + 1} = p_1 n^{-\frac{\log 1/p_1}{\log 1/p_2}} = p_1 n^{-\rho}$$

Thus the probability that no such g_i exists is at most

$$\zeta = (1 - p_1 n^{-\rho})^L \leq (1/e)^{\log(4k)} = \frac{1}{4k}$$

Now using union bound, the probability that $\forall p \in Q^* : \exists i,$ such that $p \in A_i[g_i(q)]$ is at least $\frac{3}{4}$.

For the second part, note that the probability that $g_i(p) = g_i(q)$ for $p \in P \backslash B(q, cr)$ is at most $p_2^K = \frac{1}{n}$. Thus, the expected number of elements from $P \setminus B(q, cr)$ colliding with q, under fixed g_i is at most 1, and the expected number of collisions in all g functions, is at most L. Using Markov's inequality, the probability that we get less that $3L$ outliers is at least $2/3$.

So both events hold with constant probability $\frac{5}{12}$. $\square$

Corollary 2.4.4. *Since each point is hashed once in each hash function and each bucket of A'_i is a subset of the corresponding bucket of A_i, the total space used by this algorithm is at most*

$$nL = n\log(4k) \cdot n^{\rho}/p_1 = O(\log k \cdot n^{1+1/c})$$

where we have used the fact that $\rho \leq 1/c$, and that $p_1 = 1 - r/d \geq 1/2$. Also we need $O(nd)$ space to store the points.

Theorem 2.4.5. *With constant probability, the approximation factor achieved by the algorithm is* 6*.*

Proof. First we start by defining a set of notations which is useful in the proof.

- Let $S_i = A[g_i(q)]$, and $S = \bigcup_i S_i$.
- Let O be the set of outliers, i.e., $O = S \setminus B(q, cr)$. We know that with constant probability $|O| \leq 3L$
- Let S' be the optimal k-subset of $S \setminus O$.
- Let $U_i = A'_i[g_i(q)]$ and U_i^j be the first jk elements of U_i (note that since Algorithm 4 returns an array, the elements of U are ordered). We define $T_i = U_i^{(\ell_i+1)}$ where ℓ_i is chosen such that for any $\ell'_i < \ell_i$, we have $\left|U_i^{(\ell'_i+1)} \cap O\right| > \ell'_i$. This is exactly the T_i variable in Algorithm 5. Moreover, let $T = \bigcup_i T_i$.
- Define Q^* to be the optimal k-subset of $P \cap B(q, r)$, and Q be our returned set, i.e., $Q = GMM(T \setminus O, k)$. Let $a \in Q$ be the added point at the last step of GMM, then define $Q_- = Q \setminus \{a\}$.

From the description of algorithm it is obvious that $Q \subset B(q, cr)$. Also by Theorem 2.4.3, with constant probability we have $Q^* \subset S$, and that the total number of outliers does not exceed $3L$. Thus we have $div_k(Q^*) = div(Q^*) \leq div_k(S \setminus O)$, and therefore it is enough to prove that under these conditions, $div(Q) \geq div_k(S \setminus O)/6 = div(S')/6$.

By pigeonhole principle, since $|S'| = k$ and $|Q_-| = k - 1$, then there exist $p, p' \in S'$ whose closest point in Q_- is the same, i.e., there exists $c \in Q_-$ such that $KC(\{p\}, Q_-) = dist(p, c)$ and $KC(\{p'\}, Q_-) = dist(p', c)$. Therefore, by triangle inequality, we have

$$\begin{aligned} div(S') &\leq dist(p, p') \leq dist(p, c) + dist(p', c) \\ &\leq 2KC(S \setminus O, Q_-) \end{aligned} \tag{2.1}$$

By lemma 2.4.1 $T_i \setminus O$ is a $\frac{1}{3}$-coreset for $S_i \setminus O$, and therefore their union $T \setminus O$, is a $\frac{1}{3}$-coreset for $S \setminus O$, and thus we have

$$KC(S \setminus O, Q_-) \leq 3KC(T \setminus O, Q_-) \tag{2.2}$$

Now note that a is chosen in the last step of GMM$(T \setminus O, k)$. Thus for any point $b \in (T \setminus O) \setminus Q$, since it is not chosen by GMM, b should be closer to Q_- than a, i.e., we should have $KC(\{b\}, Q_-) \leq KC(\{a\}, Q_-)$. This means that

$$KC(T \setminus O, Q_-) \leq KC(\{a\}, Q_-) \leq div(Q) \tag{2.3}$$

Putting together equations 2.1, 2.2 and 2.3 finishes the proof. □

Lemma 2.4.6. *With constant probability the query time is* $O((k^2 + \frac{\log n}{r})d \cdot \log k \cdot n^{1/c})$

Proof. The query time of the algorithm for each query, has three components. First there are $O(L)$ hash computations each taking $O(K)$

$$\begin{aligned} O(KL) = O((\log_{1/p_2} n) \cdot L) &= O(\frac{d}{r} \log n \cdot \frac{\log k}{1 - r/d} \cdot n^{1/c}) \\ &= O(\log k \cdot n^{1/c} \cdot \log n \cdot \frac{d}{r}) \end{aligned}$$

Where we have used the approximation $\log p_2 \approx 1 - p_2 = \frac{cr}{d}$, $c \geq 1$ and $r/d \leq 1/2$. Second, in the last step of Algorithm 5, with constant probability the total number of outliers is at most $3L$. Therefore, we need to run the GMM algorithm for at most

$O(kL)$ number of points, i.e., $|T| \leq 3L$. Then GMM takes

$$O(k^2 Ld) = O(d \cdot k^2 \log k \cdot n^{1/c})$$

Finally, as mentioned before, the inner loop (steps $4 - 11$) of the algorithm 5 can be implemented incrementally such that the total time it takes is $O(k\ell_i d)$.Thus the total running time of the loop is $O(kd \sum_i \ell_i) = O(kLd)$. $\square$

Chapter 3

Line Near Neighbor Problem

The approximate nearest neighbor problem can be generalized to the case where database and/or the query objects are more complex than points and are essentially k-dimensional flats. From the practical perspective, lines and low dimensional flats in general, are used to model data under linear variations such as physical objects under different lighting [10]. Despite their practical importance, these problems have not been investigated in depth.

There are two basic ways of extending the nearest neighbor problem to k-flats. One way of extension is to let the query be a k-dimensional flat for some small value of k. It is defined in [6] and is called the Approximate k-Flat Nearest Neighbor problem which is a generalization of ANN problem where $k = 0$. For the case when the query is a 1-flat, i.e., a line, they provide a data structure which has query time sub-linear in n and uses polynomial space. Specifically, for $1+\epsilon$ approximation, they obtained a query time of $O(d^3 n^{0.5+t})$ with a space of $d^2 n^{O(1/\epsilon^2+1/t^2)}$ for any desired $t > 0$. In the dual problem, the query is still a point but the data set consists of k-flats. Two results are known in this case [10, 27]. The first algorithm is essentially heuristic (although it allows some control of the quality of approximation). The second algorithm provides provable guarantees and fast query time of $(d + \log n + 1/\epsilon)^{O(1)}$. Unfortunately, the space requirement of the algorithm is *super-polynomial*, of the form $2^{(\log n)^{O(1)}}$.

In this thesis, we consider the problem of Approximate Line Near Neighbor (LNN) in which the database is a set of n lines L in the d dimensional euclidean space $\mathbb{R}^d$.

We want to preprocess the set of lines so that we can answer the Line Near Neighbor (LNN) queries in sub-linear time. That is, given a query point $q \in \mathbb{R}^d$, We want to report a line $\ell \in L$ (if there is any), such that $dist(q, \ell) \leq r$ for some threshold value r, where $dist(q, \ell)$ is the euclidean distance between them. However, since solving the exact problem seems to require scanning all lines, we focus on the approximate version of the problem. More specifically we present a $(1+\epsilon)$-approximation algorithm which uses polynomial space, i.e., $O(n + \log \Delta + d)^{O(1/\epsilon^2)}$, and has the running time of the form $(d + \log n + 1/\epsilon + \log \Delta)^{O(1)}$, where we assume that all points of interest (query points and the closest point of the closest line) live in $[0, \Delta]^d$. This will be polynomial in n when ϵ is fixed and Δ is at most exponential in n and therefore is an improvement over the existing results.

We first solve the problem in the special case where we have only two lines in the database. Based on the relative angle of the two lines, we design two different algorithms. The first case is when the angle between two lines is not too small for some appropriate definition of small. This mean that the lines diverge quickly and their distance grows fast as we go farther from their intersection (or the point where their distance is minimal). The approach here is to sample a set of points from the lines which can almost represent those lines and thus calling ANN on the union of those points can lead us to the closest line. We show that in this case, the number of samples we need is not too large.

However in the second case where the angle between the two lines is too small, we can consider the lines to be almost parallel. Then we can define a set of hyperplanes which are almost perpendicular to both of them and project the lines on tho them. We then build an instance of ANN on each hyperplane using the points of intersection of lines and the hyperplane. Given the query point q, we find the closest hyperplane and project q onto it and solve ANN on that hyperplane and report the corresponding line of the approximate nearest neighbor of the projected query point.

In the second section, we show how to merge the data structures for different pairs of lines and provide a two phase algorithm. In the first phase, we find a line which is either close to the query point or almost parallel to the optimum line. Then in the second phase, we recursively improve the current line and prove that the we can eventually find the approximate near line.

3.1 Problem Definition

Let L be a set of n lines in the d dimensional euclidean space $\mathbb{R}^d$. We want to preprocess the set of lines so that we can answer the Line Near Neighbor (LNN) queries in sub-linear time. That is, given the query point $q \in \mathbb{R}^d$, We want report a line $\ell \in L$ (if there is any), such that $dist(q, \ell) \leq r$, where $dist(q, \ell)$ is the euclidean distance between them. The approximate version of the problem defined below.

Definition 3.1.1. ***Approximate Line Near Neighbor Problem*** *$LNN(L, \epsilon)$, given $n = |L|$ lines in $\mathbb{R}^d$ and an error parameter ϵ, we want to build a data structure such that given a query point q and a search radius r, if there is a line $\ell^* \in L$ such that $dist(q, \ell^*) \leq r$, then it reports a line $\ell \in L$ such that $dist(q, \ell) \leq r(1 + \epsilon)$.*

Furthermore, we assume that the query point q and the closest point $p^* \in L$ on the closest line lie in the bounding box $[0, \Delta]^d$ and provide an algorithm whose running time depends on Δ. Also it is easy to see that we can scale everything (including Δ) by r and assume that the search radius is just 1.

Moreover, for ease of notation, we assume that $\log d = O(\log \Delta)$ and thus $\log(\Delta\sqrt{d}) = O(\log \Delta)$, otherwise the term corresponding to $\log \Delta$ will be dominated by the d term in the running time and space bounds. This assumption is used since the length of any line that falls in the bounding box $[0, \Delta]^d$ is at most $\Delta\sqrt{d}$.

Let us also define another notation. By $ANN(P, \epsilon)$ for any point set P and error parameter ϵ, we mean the approximate nearest neighbor data structure on P and error parameter ϵ. Also let $ANN_P(q, \epsilon)$, denote finding the approximate closest point using the previously built data structure $ANN(P, \epsilon)$. Moreover, let $S(n, \epsilon)$ and $T(n, \epsilon)$ respectively denote the space bound used by $ANN(P, \epsilon)$, and the query time of $ANN_P(q, \epsilon)$, when $|P| = n$. We will use this notation in describing and analyzing our algorithms.

3.2 Two lines

In this section we consider the case where we want to distinguish the closest among two lines to the query point. We present two different approaches for two different cases. The first case is when the angle between two lines is not too small and the second case is when the angle is too small so that we can consider the lines to be almost parallel.

Let the two lines be ℓ_1 and ℓ_2. Also let $D = dist(\ell_1, \ell_2)$ be their distance which is the length of the shortest segment ab for $a \in \ell_1$ and $b \in \ell_2$, e.g. $D = 0$ when they collide. It can be easily seen that ab is perpendicular to both of ℓ_1 and ℓ_2. Furthermore, let α be the smaller of the two angles between the two lines. We introduce the following definition:

Definition 3.2.1. *We say that the two lines ℓ_1 and ℓ_2 are δ-close to parallel for some value of δ, if we have $(\sin\alpha \leq \delta)$, otherwise we say that they are δ-far from being parallel. Also we say that the two lines are almost parallel if $(\sin\alpha \leq \epsilon)$, and otherwise they are non-parallel.*

3.2.1 Non-parallel lines

Here we consider the case where the lines are non-parallel. The approach here is to sample a set of points from each line and show that they can almost represent the lines and thus calling ANN on them will suffice. It is shown in the following lemma.

Lemma 3.2.2. *If the two lines are non-parallel, i.e., $(\sin\alpha > \epsilon)$, then we can solve $LNN(L, 4\epsilon)$ for $L = \{\ell_1, \ell_2\}$ and $\epsilon \leq 1/7$, within space and time bounds of $S(O(\frac{\log\Delta}{\epsilon^2}), \epsilon)$ and $T(O(\frac{\log\Delta}{\epsilon^2}), \epsilon)$.*

Proof. We sample points from ℓ_1 as follows. Let $p_0 = a$, and for any integer $1 \leq i \leq \log(\Delta\sqrt{d})$, let $p_i \in \ell$ be the point with distance 2^{i-1} from a (recall that $(\Delta\sqrt{d})$ is the maximum length of a line which falls into the bounding box $[0, \Delta]^d$). For $0 \leq i \leq \log(\Delta\sqrt{d})$, let S_i be the set of $1/\epsilon^2$ points that divide the interval $[p_i, p_{i+1})$ to equal segments, that is their distance from p_i is $\{0, 2^{i-1}\epsilon^2, 2^{i-1}(2\epsilon^2), 2^{i-1}(3\epsilon^2), \cdots, 2^{i-1}((\frac{1}{\epsilon^2}) -$

$1)\epsilon^2\}$. Note that we should sample the line ℓ_1 on both sides of a. We also sample from ℓ_2 in the same way. Let S denote the union of all these samples. It is clear that the total number of samples is at most $\frac{4}{\epsilon^2}\log(\Delta\sqrt{d}) = O(\frac{\log\Delta}{\epsilon^2})$.

Now build an approximate nearest neighbor data structure on the set S with parameter ϵ, i.e., $ANN(S, \epsilon)$. Given the query point q, we report the line corresponding to the call of $ANN_S(q, \epsilon)$.

Now suppose that line ℓ_1 is closer to q than ℓ_2. Also assume that $p \in \ell_1$ is the closest point to q, and $p' \in \ell_2$ is the closest point to q on the line ℓ_2, and let ρ be the closest distance of q to any of the two lines, i.e., $\rho = dist(q, p)$. Also let p^* be the closest point on ℓ_2 to p. Furthermore assume that p lies in the interval $[p_k, p_{k+1})$.

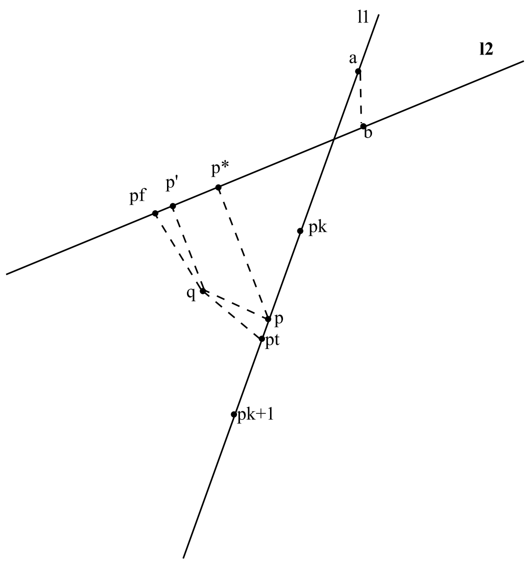

Figure 3-1: Non-parallel lines

Now suppose that the ANN algorithm does not find the correct line and instead returns some point which lies on ℓ_2. Let this point be p_f and let p_t be the closest point in $S_k \cap \ell_1$ to q, as shown in figure 3-1. This means that p_t is either the immediate point of $S_k \cap \ell_1$ to the right of p, or the one to the left of it and thus $dist(p, p_t) \leq 2^{k-1}\epsilon^2$. Then by properties of ANN algorithm and triangle inequality, we have

$$dist(q, p_f) \leq dist(q, p_t) * (1+\epsilon) \leq (1+\epsilon)(dist(q, p) + dist(p, p_t)) \leq (1+\epsilon)(\rho + 2^{k-1}\epsilon^2)$$

However see that since $p_t \in S_k$, we have $dist(p,a) \geq 2^{k-1}$ and thus $dist(p,p^*) \geq 2^{k-1}\sin\alpha \geq 2^{k-1}\epsilon$. Therefore we get

$$2^{k-1}\epsilon \leq dist(p,p^*) \leq dist(p,p') \leq dist(p,q) + dist(q,p') \leq \rho + dist(q,p_f)$$

So we have

$$dist(q,p_f) \leq (1+\epsilon)(\rho + (\rho + dist(q,p_f))\,\epsilon)$$

$$dist(q,p_f) \leq \frac{\rho(1+\epsilon)(1+\epsilon)}{1-\epsilon-\epsilon^2} \leq \rho(1+4\epsilon)$$

where the last inequality holds for $\epsilon \leq 1/7$. □

3.2.2 Almost parallel lines

Here we consider the case where the lines are almost parallel. However instead of having the condition ($\sin\alpha \leq \epsilon$), we consider an equivalent formulation which is going to be directly useful in solving LNN in the general case. Suppose that there is a base line ℓ. Let $\alpha_1 = angle(\ell,\ell_1)$ and $\alpha_2 = angle(\ell,\ell_2)$. Then we consider the case where ($\sin\alpha_1 \leq \delta$), ($\sin\alpha_2 \leq \delta$) and ($\sin\alpha \geq \delta/2$) for some $\delta \leq \epsilon$. That is, ℓ_1 and ℓ_2 are δ-close to ℓ but they are $\delta/2$-far from each other. Clearly by setting $\ell = \ell_1$ and $\delta = \epsilon 2^{-i}$ for some integer value $0 \leq i$, this generalizes the original condition $\sin\alpha \leq \epsilon$.

By rotation and translation, without loss of generality we can assume that ℓ is the x axis and that a has its x-coordinate equal to 0 and that b has its x-coordinate equal to $-w$ for some $w \geq 0$. Also since $dist(a,b) \leq D$, we have $w \leq D$. Now we define three sets of hyperplanes which are perpendicular to ℓ.

- Let $H_1, H_2, \cdots, H_k$ be the set of hyperplanes with positive x coordinate which are perpendicular to ℓ such that $dist(a,H_1) = \epsilon^2$, and $dist(a,H_i) = (1+\epsilon)dist(a,H_{i-1})$. Note that k is at most $\log_{1+\epsilon}(\Delta\sqrt{d}/\epsilon^2) \approx \log(\Delta/\epsilon)/\epsilon$.

- Let $G_1, \cdots, G_m$ be the set of hyperplanes between a and b such that $dist(a,G_1) = 0$ and we have $dist(G_i,G_{i-1}) = D\epsilon$ and with the last hyperplane G_m passing through b, i.e. $dist(G_m,b) = 0$. Then since clearly the x-coordinate of a and b

differ by at most D, then we have that $m = O(1/\epsilon)$.

- Let $H'_1, \cdots H'_{k'}$ be defined the same as the first type but for the interval -infinity to b.

Algorithm For any hyperplane $h \in (\bigcup_{i=1}^{k} H_i) \cup (\bigcup_{i=1}^{m} G_i) \cup (\bigcup_{i=1}^{k'} H'_i)$, we build an instance $ANN_h(\{\ell_1, \ell_2\} \cap h, \epsilon)$ on the intersection of the two lines with the hyperplane in the preprocessing step. Given the query point q, first we find the closest hyperplane $g \in (\bigcup_{i=1}^{k} H_i) \cup (\bigcup_{i=1}^{m} G_i) \cup (\bigcup_{i=1}^{k'} H'_i)$ to the query point and project q onto g to get the point q_g. Then we report the line corresponding to the point returned by $ANN_g(q_g, \epsilon)$.

In order to prove the correctness of the algorithm, we consider three cases separately: whether the closest hyperplane to q is an H-type, G-type or H'-type hyperplane.

Case q falls between H-type hyperplanes, This case is equivalent to the condition that $(x_a + D\epsilon^2 \leq x_q \leq \infty)$ (where x_p denotes the value of the x-coordinate of a point p). Suppose that the x-coordinate of q is between the x-coordinate of H_{i-1} and the x-coordinate of H_i for some value of $i > 1$. Let h denote the hyperplane which is perpendicular to ℓ and passes through q. Let $a_h = \ell_1 \cap h$ and $b_h = \ell_2 \cap h$ be the intersection of the lines with the hyperplane h. Also let c_1 and c_2 be the points on ℓ_1 and ℓ_2 with the minimum distance to q. Define $\rho = ||a_h - b_h||_2$ to be the distance of the lines on the hyperplane h, and t to be the distance of H_{i-1} from the point a (x-coordinate of H_{i-1}).

We prove a more general argument in this case. Let g be *any* hyperplane perpendicular to ℓ whose x coordinate is between H_{i-1} and H_i. We define $a_g = g \cap \ell_1$ and $b_g = g \cap \ell_2$ to be the intersections of the lines with the hyperplane g. Suppose that we have the data structure for $ANN_g(\{a_g, b_g\}, \epsilon)$ on the points a_g and b_g. Let q_g be the projection of the point q onto g. Furthermore let e_1 and e_2 be the projections of a_h and b_h onto g.

The following lemmas will prove that if we project q onto g and find the approximate point nearest neighbor of q_g using the data structure we had, i.e., $ANN_g(q_g, \epsilon)$, then the corresponding line to this point will be an approximate nearest line for q.

Lemma 3.2.3. *$dist(q, a_h) \leq dist(q, c_1)(1 + \delta)$ for $\delta \leq 2/3$. (Similarly $dist(q, b_h) \leq dist(q, c_2)(1 + \delta)$).*

Proof. Let p be the projection of a onto h, and o be the projection of a onto line $\overline{qa_h}$. Then since $\triangle qa_hc_1$ and $\triangle aa_ho$ are similar, we have

$$\frac{dist(a_h, q)}{dist(q, c_1)} = \frac{dist(a, a_h)}{dist(a, o)} = \frac{dist(a, a_h)}{dist(a, p)} \frac{dist(a, p)}{dist(a, o)} \leq \frac{1}{\cos \alpha_1} \cdot 1 \leq \frac{1}{\sqrt{1 - \delta^2}} \leq (1 + \delta)$$

The last inequality holds for $\delta \leq 2/3$. $\square$

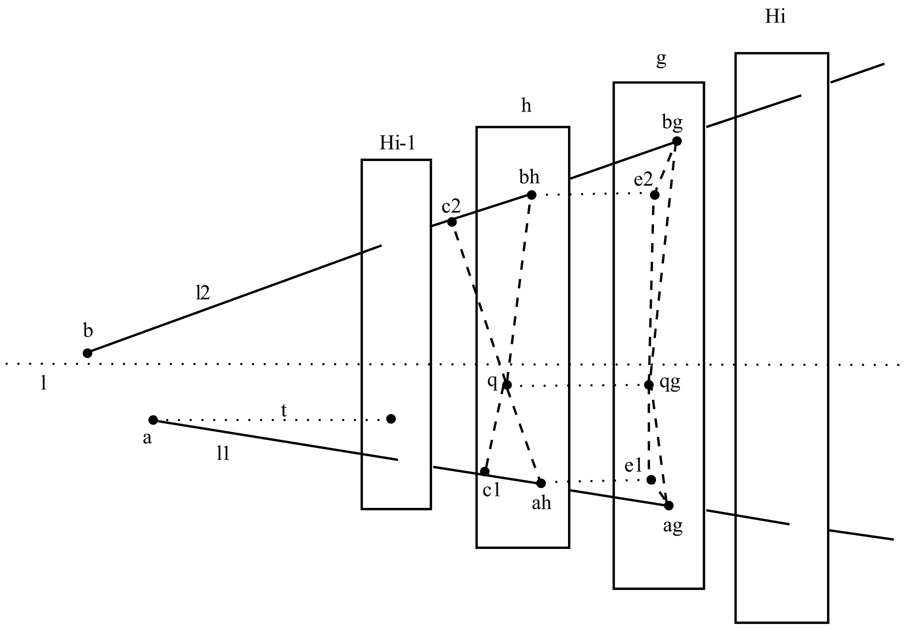

Figure 3-2: Almost parallel lines

Lemma 3.2.4. *$\rho \geq t\delta/4$ for sufficiently small δ.*

Proof. Let p be the projection of a onto h. Let h' be the hyperplane that passes through a_h and is perpendicular to ℓ_1. Let $b_{h'} = \ell_2 \cap h'$ be the intersection of ℓ_2 with this hyperplane. Furthermore, let ℓ_2' be the translated ℓ_2 along the vector $\vec{ba}$. This translation will move b to a and also since $\vec{ba}$ is perpendicular to ℓ_1, it means that $\vec{ba}$ is parallel to h'. So this translation will move $b_{h'}$ to some other point inside h' such as $b'_{h'}$. Now clearly we have

$$dist(a_h, b_{h'}) \geq dist(a_h, b'_{h'}) \geq dist(a, a_h) \tan\alpha \geq dist(a, p) \sin\alpha \geq t\delta/2$$

Let f be the projection of $b_{h'}$ onto h. Then we have

$$\rho = dist(b_h, a_h) \geq dist(f, a_h) - dist(f, b_h) \geq dist(b_{h'}, a_h) \cos\alpha_1 - dist(f, b_{h'}) \tan\alpha_2$$

$$\geq dist(b_{h'}, a_h)(\cos\alpha_1 - \sin\alpha_1 \tan\alpha_2) \geq t\delta(1 - 2\delta)/2 \geq t\delta/4$$

where the last inequality holds for sufficiently small δ. □

Lemma 3.2.5. *If $dist(q, \ell_1) \leq dist(q, \ell_2)$ then we have $(1 - O(\epsilon)) \leq \frac{dist(q_g, b_g)}{dist(q, b_h)} \leq (1 + O(\epsilon))$ for sufficiently small ϵ.*

Proof. First note that using lemma 3.2.3 we can get

$$dist(q, a_h) \leq (1 + \delta)dist(q, c_1) \leq (1 + \delta)dist(q, c_2) \leq (1 + \delta)dist(q, b_h)$$

and thus by triangle inequality we have

$$dist(q, b_h) \geq \frac{\rho}{2 + \delta} \geq \frac{\rho}{2}(1 - \delta) \tag{3.1}$$

Also using triangle inequality we have

$$dist(q_g, e_2) - dist(b_g, e_2) \leq dist(q_g, b_g) \leq dist(q_g, e_2) + dist(b_g, e_2)$$

and since $dist(q_g, e_2) = dist(q, b_h)$, we get

$$1 - \frac{dist(b_g, e_2)}{dist(q, b_h)} \leq \frac{dist(q_g, b_g)}{dist(q, b_h)} \leq 1 + \frac{dist(b_g, e_2)}{dist(q, b_h)}$$

Thus it is enough to bound $\frac{dist(b_g,e_2)}{dist(q,b_h)}$.

$$\begin{aligned}
\frac{dist(b_g,e_2)}{dist(q,b_h)} &\le \frac{dist(b_h,e_2)\tan\alpha_2}{\rho(1-\delta)/2} \quad \text{(using Equation 3.1)}\\
&\le \frac{t\epsilon\delta/\sqrt{1-\delta^2}}{t\delta(1-\delta)/8} \quad \text{(by Lemma 3.2.4 and since } dist(h,g) \le dist(H_i,H_{i-1}) = t\epsilon)\\
&\le \frac{8\epsilon}{(1-\delta)(\sqrt{1-\delta^2})} \le 16\epsilon
\end{aligned}$$

where the inequalities hold for sufficiently small ϵ.

□

Lemma 3.2.6. *If q falls in the H-type hyperplane, that is if the x-coordinate of q is greater or equal to the x-coordinate of H_1, then the output of the algorithm is within a factor of $(1+O(\epsilon))$ from the optimum for sufficiently small value of ϵ.*

Proof. Suppose that the result of our algorithm was not the actual closest line that it had to output. We then prove that we are not off by more than a multiplicative factor of $(1+O(\epsilon))$. Without loss of generality suppose that $dist(q,\ell_1) \le dist(q,\ell_2)$ but we output ℓ_2 instead. We prove a bound on the distance of q to ℓ_2. First we see the following claim.

Claim 3.2.7. $dist(q_g,a_g) \ge \rho(1-O(\epsilon))/2$.

Proof.

$$\begin{aligned}
dist(q_g,a_g) &\ge \frac{dist(q_g,b_g)}{1+\epsilon} \quad \text{by properties of ANN}\\
&\ge \frac{(1-O(\epsilon))}{(1+\epsilon)} dist(q,b_h) \quad \text{by Lemma 3.2.5}\\
&\ge \frac{(1-O(\epsilon))(1-\delta)}{2(1+\epsilon)}\rho \quad \text{by Equation 3.1}\\
&\ge \rho(1-O(\epsilon))/2
\end{aligned}$$

□

Claim 3.2.8. $dist(q_g,a_g) \le dist(q,a_h)(1+O(\epsilon))$ *for sufficiently small value of ϵ.*

Proof.

$$\begin{aligned}\frac{dist(q_g,a_g)}{dist(a_g,e_1)} &\geq \frac{\rho(1-O(\epsilon))/2}{dist(a_h,e_1)\tan\alpha_1} \quad \text{(by claim 3.2.7)}\\ &\geq \frac{t\delta(1-O(\epsilon))/8}{t\epsilon\delta/\sqrt{1-\delta^2}} \quad \text{(by Lemma 3.2.4)} \qquad (3.2)\\ &\geq \frac{(1-O(\epsilon))}{8\epsilon}\end{aligned}$$

$$\begin{aligned}\frac{dist(q_g,a_g)}{dist(q,a_h)} &\leq 1+\frac{dist(a_g,e_1)}{dist(q_g,e_1)} \quad \text{(by triangle inequality and since } dist(q_g,e_1)=dist(q,a_h)\text{)}\\ &\leq 1+\frac{1}{\frac{dist(q_g,a_g)}{dist(a_g,e_1)}-1} \quad \text{(by triangle inequality)}\\ &\leq 1+\frac{8\epsilon}{(1-O(\epsilon))} \quad \text{(by equation 3.2)}\\ &\leq 1+8\epsilon(1+O(\epsilon))\\ &\leq 1+O(\epsilon)\end{aligned}$$

where the last two inequalities work for sufficiently small value of ϵ □

Now we prove the final part of the lemma.

$$\begin{aligned}dist(q,\ell_2)=dist(q,c_2) &\leq dist(q,b_h)\\ &\leq \frac{dist(q_g,b_g)}{(1-O(\epsilon))} \quad \text{(by lemma 3.2.5)}\\ &\leq \frac{(1+\epsilon)}{(1-O(\epsilon))}dist(q_g,a_g) \quad \text{(by properties of ANN)}\\ &\leq \frac{(1+\epsilon)(1+O(\epsilon))}{(1-O(\epsilon))}dist(q,a_h) \quad \text{(by claim 3.2.8)}\\ &\leq \frac{(1+\epsilon)(1+O(\epsilon))(1+\epsilon)}{(1-O(\epsilon))}dist(q,c_1) \quad \text{(by lemma 3.2.3)}\\ &\leq dist(q,\ell_1)(1+O(\epsilon))\end{aligned}$$

where the inequalities hold for sufficiently small ϵ. □

Case q falls between H'-type hyperplanes That is equivalent to the condition that $(-\infty \leq x_q \leq x_b - D\epsilon^2)$. The proof in this case is exactly the same as the proof for H-type hyperplanes.

Case q falls between G-type hyperplanes. This case is equivalent to the condition that $(x_b - D\epsilon^2 \leq x_q \leq x_a + D\epsilon^2)$. Suppose that the x-coordinate of q is between the x-coordinate of G_{i-1} and the x-coordinate of G_i for some value of $(1 \leq i \leq m+1)$ where we define $G_0 = H_1$ and $G_{m+1} = H_1'$. Note that we have $dist(G_0, G_1) = dist(G_m, G_{m+1}) = \epsilon^2$. Similar to the proof for H-type, let us define h,a_h, b_h, c_1, c_2, ρ (Note that t will be defined later).

Again, let g be *any* hyperplane perpendicular to ℓ whose x coordinate is between G_{i-1} and G_i. We define a_g, b_g, q_g, e_1 and e_2 similarly. Suppose that we have the data structure for $ANN_g(\{a_g, b_g\}, \epsilon)$ on the points a_g and b_g. We will prove that reporting the line corresponding to the point returned by $ANN_g(q_g, \epsilon)$, is an approximate near line for q.

To prove the correctness, we consider two different possibilities.

First, suppose that $D \leq \epsilon$. Without loss of generality assume that q is closer to ℓ_1 rather than ℓ_2. Since we want to answer the decision problem whether there is a line within distance 1 of the query point q, we assume that $dist(q, \ell_1) \leq 1$ (otherwise we don't have to report the correct line). Then we have

$$\begin{aligned} dist(q,\ell_2) \leq dist(q,b) &\leq dist(q,a_h) + dist(a_h,a) + dist(a,b) \quad \text{(by triangle inequality)} \\ &\leq (1+\epsilon)dist(q,c_1) + dist(G_1,h)/\cos\alpha_1 + D \quad \text{(by lemma 3.2.3)} \\ &\leq dist(q,\ell_1)(1+\epsilon) + (D+\epsilon^2)/(\sqrt{1-\epsilon^2}) + \epsilon \\ &\leq (1+\epsilon) + \epsilon(1+\epsilon)/(\sqrt{1-\epsilon^2}) + \epsilon \\ &\leq 1 + 4\epsilon \end{aligned}$$

Where the last inequality holds for $\epsilon \leq 1/3$. Therefore we just showed that it does not really matter which line to report in this case. So whatever we report is an approximate near line.

Next suppose that $D \geq \epsilon$. The proof of this case is almost similar to the case in H-type hyperplanes. It is enough to define $t = D$ in the proof. Note that since a and

b are the closest points of ℓ_1 and ℓ_2, we have that

$$\rho = dist(a_h, b_h) \geq D = t \geq t\delta/4$$

So Lemma 3.2.4 still holds. The only other fact using t was that the distance between two successive hyperplanes (and thus the distance of g and h) is at most $t\epsilon$. This is also true in our case where the distance of two consecutive G-type hyperplanes is at most $D\epsilon$, and also we have $dist(G_0, G_1) = dist(G_m, G_{m+1}) = \epsilon^2 \leq D\epsilon$. So we get the same bounds as for the H-type hyperplanes.

This finishes the proof of the correctness of the algorithm. We summarize the results of this section in the following theorem.

Theorem 3.2.9. *In the case where* $(\sin\alpha_1 \leq \delta)$, $(\sin\alpha_2 \leq \delta)$ *and* $(\sin\alpha \geq \delta/2)$, *the presented algorithm works correctly within a multiplicative factor of* $(1 + O(\epsilon))$ *for sufficiently small value of* ϵ, *the space it uses is* $O(m * S(2, \epsilon))$ *and its query time is equal to* $O(\log m + T(2, \epsilon))$ *where* $m = O(\frac{\log(\Delta/\epsilon)}{\epsilon})$ *is the total number of hyperplanes.*

Remark 3.2.10. *The set of hyperplanes presented in this section is sufficient for approximately distinguishing between the two lines. Furthermore, adding extra hyperplanes to this set does not break the correctness of the algorithm. This holds since we proved that if q falls between two successive hyperplanes in this set, then projecting onto any other parallel hyperplane between them also works.*

3.3 General case

The main algorithm for the general case of this problem consists of two phases. As shown in Lemma 3.2.2, for any pair of lines in L whose angle is not too small, we have come up with a set of points which almost represent the lines and they are enough to almost distinguish which line is closer to the query point q. Now in the first phase of the algorithm we merge these sets of points for any such pair of lines to get the set U and build an instance of $ANN(U, \epsilon)$ as described in Algorithm 6. Given the query point q, we find the approximate nearest neighbor among U, which we denote by u. Let the initial value of ℓ to be the line corresponding to u, i.e., ℓ_u.

In the second phase of the algorithm, we only look at the lines which have similar angle to that of ℓ and using the method described in Section 3.2.2, we recursively update ℓ. The query processing part is shown in Algorithm 7.

More formally, we keep a data structure for each such line. For each base line $\ell \in L$ and each value of $\delta = \epsilon 2^{-i}$ for $0 \leq i$, we keep the subset of lines $L_{\ell,\delta} \subset L$ such that for each $\ell' \in L_{\ell,\delta}$ we have $(\sin angle(\ell, \ell') \leq \delta)$. By Theorem 3.2.9, we know how to approximately distinguish between any two lines $\ell_1, \ell_2 \in L_{\ell,\delta}$ that have angle greater than $\delta/2$. That is, it is enough to have $O(\frac{\log(\Delta/\epsilon)}{\epsilon})$ hyperplanes that are perpendicular to ℓ and look at the intersection of the lines with the hyperplanes. Since by Remark 3.2.10, adding extra hyperplanes only increases the accuracy, we can merge the set of hyperplanes for each such pair $\ell_1, \ell_2 \in L_{\ell,\delta}$ into the set $H_{\ell,\delta}$. Then for each hyperplane $H \in H_{\ell,\delta}$, we build an instance of approximate nearest neighbor $ANN_H(L_{\ell,\delta} \cap H, \epsilon)$.

At the query time, after we find the initial line ℓ in the first phase of the algorithm, in the second phase, we set the initial value of $\delta = \epsilon$. We then find the closest hyperplane $g \in H$ to the query point q and project q onto h to get q_g. We then update ℓ with the line corresponding to the approximate nearest neighbor $ANN_g(q_g, \epsilon)$ and halve the value of δ and repeat this phase again. We continue this process until all the lines we are left with, are parallel to ℓ and report the best of the lines we found in each iteration. The following lemmas establishes a more formal proof for the correctness and time and space bounds of the algorithms.

Lemma 3.3.1. *For a sufficiently small ϵ, the Algorithm 7 computes the approximate line near neighbor correctly.*

Proof. Let ℓ^* be the optimal closest line to q. Then by Lemma 3.2.2 if $(\sin angle(\ell_u, \ell^*) \geq \epsilon)$, then the reported line satisfies the approximate bounds, i.e., if $dist(q, \ell^*) \leq 1$ then $dist(q, \ell_u) \leq (1 + 4\epsilon)$ and since ℓ_{opt} can only improve over ℓ_u, we get the same bound for ℓ_{opt}.

Algorithm 6 Preprocessing

Input The set of lines L

```
1: $U \leftarrow \emptyset$
2: for all pairs of non-parallel lines $\ell_1, \ell_2 \in L$ do
3:     Add the set of $O(\frac{1}{\epsilon^2} \log \Delta)$ points as described in Lemma 3.2.2 to $U$
4: end for
5: Build $ANN(U, \epsilon)$
6: for $\ell \in L$ do
7:     for $0 \leq i$ do
8:         $\delta \leftarrow \epsilon 2^{-i}$
9:         $L_{\ell,\delta} \leftarrow$ all lines $\ell' \in L$ s.t. $\sin angle(\ell, \ell') \leq \delta$
10:        $H_{\ell,\delta} \leftarrow \emptyset$
11:        for $\ell_1, \ell_2 \in L_{\ell,\delta}$ do
12:            if $\sin angle(\ell_1, \ell_2) \geq \delta/2$ then
13:                add the set of hyperplanes perpendicular to $\ell$ to distinguish between $\ell_1$
                   and $\ell_2$ as described in Theorem 3.2.9 to $H_{\ell,\delta}$
14:            end if
15:        end for
16:        sort $H_{\ell,\delta}$ based on their order on $\ell$
17:        for $H \in H_{\ell,\delta}$ do
18:            $P \leftarrow L_{\ell,\delta} \cap H$
19:            build an instance of approximate nearest neighbor on $P$ with parameter $\epsilon$,
               i.e., $ANN_H(P, \epsilon)$.
20:        end for
21:    end for
22: end for
```

Now consider the case where in the first phase we have ($\sin angle(\ell_u, \ell^*) < \epsilon$). In this case we maintain the following invariant before each iteration of the algorithm in the second phase. If ℓ_{opt} is not an approximate near neighbor of q, then $L_{\ell,\delta}$ contains ℓ^*. By the earlier argument, in the first iteration this claim is true, since either ℓ_u is an approximate nearest neighbor of ℓ^*, or $L_{\ell_u,\epsilon}$ contains ℓ^*. For the inductive step, let ℓ_p be the line we find in the iteration. Then if ($\sin angle(\ell^*, \ell_p) \geq \delta/2$), then by Theorem 3.2.9 we should have $dist(q, \ell_p) \leq dist(q, \ell^*)(1 + O(\epsilon))$. That is true, since we have included set of sufficient hyperplanes in $H_{\ell,\delta}$ to be able to distinguish between them and furthermore, by Remark 3.2.10 having extra hyperplanes in $H_{\ell,\delta}$ just increases the accuracy of the line we find and does not hurt. Also if we are in the

Algorithm 7 Query processing

```
Input query point $q$
Output approximate nearest line $\ell_{opt}$
1: $u \leftarrow ANN_U(q, \epsilon)$ , $\ell_u \leftarrow$ the line which $u$ lies on
2: $\ell \leftarrow \ell_u$
3: $\ell_{opt} \leftarrow \ell_u$
4: for $0 \leq i$ do
5:     $\delta \leftarrow \epsilon 2^{-i}$
6:     Find the closest hyperplanes $g \in H_{\ell,\delta}$ to the query point $q$
7:     $q_g \leftarrow$ projection of $q$ onto $g$
8:     $p \leftarrow ANN_g(q_g, \epsilon)$, $\ell_p \leftarrow$ the line which $p$ lies on
9:     update $\ell_{opt}$ with the best of $\{\ell_{opt}, \ell_p\}$
10:    if all lines in $L_{\ell,\delta}$ are parallel to $\ell$ then
11:        break
12:    end if
13:    $\ell \leftarrow \ell_p$
14: end for
15: Output $\ell_{opt}$
```

case that ($\sin angle(\ell^*, \ell_p) \leq \delta/2$), then by definition ℓ^* should be contained in $L_{\ell_p,\delta/2}$ which is exactly $L_{\ell,\delta}$ of the next iteration. By the time we end the algorithm, either there is only one line left in $L_{\ell,\delta}$ and thus we have $\ell = \ell^*$, or there are some parallel lines to ℓ in it. In this case, it is easy to see that the approximate nearest neighbor of the projected q onto any hyperplane perpendicular to ℓ finds the approximate nearest line among $L_{\ell,\delta}$. Thus $dist(q, \ell) = dist(q, \ell^*)(1 + O(\epsilon))$.

Note that since we take the best line we find in each iteration, we output the correct solution in the end. Also if α_{min} denotes the minimum pairwise angle between any two non parallel lines, then since we halve the angle threshold δ in each iteration, at some point it will pass over α_{min} and the loop ends.

$\square$

Lemma 3.3.2. *The space bound of the presented algorithm with parameters $c = 1+\epsilon$ and $r = 1$ is*

$$O(\frac{n^3 \log \Delta \log(\Delta/\epsilon)}{\epsilon}) \times S(n, \epsilon) + S(O(\frac{n^2 \log \Delta}{\epsilon^2}), \epsilon)$$

and the query processing time bound is

$$O(\log \Delta) \times T(n, \epsilon) + T(O(\frac{n^2 \log \Delta}{\epsilon^2}), \epsilon)$$

Proof. In the first phase of the algorithm, the space we use is equal to the space we need to keep $ANN(U, \epsilon)$. By Lemma 3.2.2, the set U contains at most $O(\frac{\log \Delta}{\epsilon^2})$ points per each pair of lines. Moreover the running time of the first phase is bounded by the time needed to find $ANN_U(q, \epsilon)$.

In the second phase, suppose that the minimum pairwise angle in the database is equal to α_{min} and let $\epsilon_{min} = \sin \alpha_{min}$. Then there are at most $n \log(\epsilon/\epsilon_{min})$ different $L_{\ell,\delta}$ sets. By Theorem 3.2.9, for each of them we build $O(n^2 \times \frac{\log(\Delta/\epsilon)}{\epsilon})$ instances of ANN each of size at most n. However, we will only search one of them per iteration, therefore the lemma holds.

So the total space we get is

$$O(\frac{n^3 \log(\epsilon/\epsilon_{min}) \log(\Delta/\epsilon)}{\epsilon}) \times S(n, \epsilon) + S(O(\frac{n^2 \log \Delta}{\epsilon^2}), \epsilon)$$

the the total running time is

$$\log(\epsilon/\epsilon_{min}) \times T(n, \epsilon) + T(O(\frac{n^2 \log \Delta}{\epsilon^2}), \epsilon)$$

However note that if the two lines are at a distance less than ϵ in the entire bounding box $[0, \Delta]^d$, then they are approximately the same. Therefore we have $\epsilon_{min} \geq \frac{\epsilon}{\Delta\sqrt{d}}$ and thus $\log(\epsilon/\epsilon_{min}) \leq \log(\Delta\sqrt{d}) = O(\log \Delta)$ and we get the bounds. □

Bibliography

[1] S. Abbar, S. Amer-Yahia, P. Indyk, S. Mahabadi, and K. Varadarajan. Diverse Near Neighbor Problem. In *SoCG*, 2013.

[2] S. Abbar, S. Amer-Yahia, P. Indyk, and S. Mahabadi. Efficient computation of diverse news. In *WWW*, 2013.

[3] P. K. Agarwal, S. Har-peled, and H. Yu. Robust shape fitting via peeling and grating coresets. In *In Proc. 17th ACM-SIAM Sympos. Discrete Algorithms*, pages 182–191, 2006.

[4] P. K. Agarwal, S. Har-peled, and K. R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry*, volume 52, pages 1–30, 2005.

[5] N. Ailon, and B. Chazelle. Approximate nearest neighbors and the Fast Johnson-Lindenstrauss Transform. In Proceedings of the Symposium on Theory of Computing, 2006.

[6] A. Andoni, P. Indyk, R. Krauthgamer, H. L. Nguyen Approximate line nearest neighbor in high dimensions SODA, pages 293–301, 2009.

[7] A. Andoni. LSH algorithm and implementation (E2LSH). http://www.mit.edu/ andoni/LSH/.

[8] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD*, pages 781–792, 2011.

[9] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms,pages 573–582 , 1994

[10] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search with applications to pattern recognition. In Computer Vision and Pattern Recognition (CVPR07), pages 18, June 2007.

[11] J. L. Bentley. Multidimensional binary search trees used for associative searching. comm. ACM 18, pages 509–517.

[12] A. Chakrabarti, O. and Regev. An optimal randomised cell probe lower bounds for approximate nearest neighbor searching. In Proceedings of the Symposium on Foundations of Computer Science.,2004.

[13] M. Datar, N. Immorlica, P. Indyk, V. and Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the ACM Symposium on Computational Geometry, 2004.

[14] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, pages 41–47, 2010.

[15] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *SIGMOD*, pages 421–432, 2012.

[16] S. Gollapudi and A. Sharma. An axiomatic framework for result diversification. In *WWW*.

[17] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance.

[18] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Of Computing*, 8:321–350, 2012.

[19] S. Har-peled and Y. Wang. Shape fitting with outliers. *SIAM J. Comput.*, 33(2):269–285, 2004.

[20] S. Har-Peled. A replacement for voronoi diagrams of near linear size. In Proc. of FOCS, pages 94103, 2001.

[21] P. Indyk. Nearest neighbors in high-dimensional spaces. In Handbook of Discrete and Computational Geometry, CRC Press, 2003

[22] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. Proc. of STOC, pages 604613, 1998.

[23] A. Jain, P. Sarda, and J. R. Haritsa. Providing diversity in k-nearest neighbor query results. In *PAKDD*, pages 404–413, 2004.

[24] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In Proceedings of the Symposium on Theory of Computing, 1997.

[25] R. Krauthgamer, J. R. and Lee. Navigating nets: Simple algorithms for proximity search. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2004.

[26] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. SIAM J. Comput., 30(2):457 474, 2000. Preliminary version appeared in STOC98.

[27] A. Magen. Dimensionality reductions in ‘2 that preserve volumes and distance to affine spaces. Discrete and Computational Geometry, 38(1):139153, July 2007. Preliminary version appeared in RANDOM 02.

[28] R. Panigrahy. Entropy-based nearest neighbor algorithm in high dimensions. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2006.

[29] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Facility dispersion problems: Heuristics and special cases. *Algorithms and Data Structures*, pages 355–366, 1991.

[30] H. Samet, 2006. Foundations of Multidimensional and Metric Data Structures. Elsevier, 2006

[31] G. Shakhnarovich, T. Darrell, and P. Indyk. Nearest Neighbor Methods in Learning and Vision. Neural Processing Information Series, MIT Press.

[32] Z. Syed, P. Indyk, and J. Guttag. Learning approximate sequential patterns for classification. *Journal of Machine Learning Research.*, 10:1913–1936, 2009.

[33] K. Varadarajan and X. Xiao. A near-linear algorithm for projective clustering integer points. In *SODA*, pages 1329–1342, 2012.

[34] M. J. Welch, J. Cho, and C. Olston. Search result diversity for informational queries. In *WWW*, pages 237–246, 2011.

[35] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *ICDE*, pages 1299–1302, 2009.

[36] C.-N. Ziegler, S. M. Mcnee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.