Approximate Nearest Line Search in High Dimensions

Sepideh Mahabadi
MIT
mahabadi@mit.edu

Abstract

We consider 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, given a query point $q \in \mathbb{R}^d$, reports a line $\ell \in L$ such that its distance to the query is within $(1+\epsilon)$ factor of the distance of the closest line to the query point q. The problem is a natural generalization of the well-studied Approximate Nearest Neighbor problem for point sets (ANN), and is a natural first step towards understanding how to build efficient nearest-neighbor data structures for objects that are more complex than points.

Our main result is a data structure that, for any fixed $\epsilon > 0$, reports the approximate nearest line in time $(d + \log N + 1/\epsilon)^{O(1)}$ using $O(N + d)^{O(1/\epsilon^2)}$ space. This is the first high-dimensional data structure for this problem with poly-logarithmic query time and polynomial space. In contrast, the best previous data structure for this problem, due to Magen [16], required quasi-polynomial space. Up to polynomials, the bounds achieved by our data structure match the performance of the best algorithm for the approximate nearest neighbor problem for point sets.

1 Introduction

The *Nearest Neighbor* problem is a fundamental geometric problem which is of major importance in several areas such as databases, information retrieval, pattern recognition and computer vision. 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 the 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. There are several efficient algorithms known for the case when the dimension d is low (e.g., up to 10 or 20), see [18] 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. Fortunately, faster algorithms can be obtained by resorting to approximation (e.g., [6, 13, 12, 15, 10, 14, 9, 8, 17, 1, 2, 4], see also surveys [19] and [11]). In this formulation, the algorithm is allowed to return a point whose distance from the query is at most $1 + \epsilon$ times the distance from the query to its nearest point. The current results for ANN in Euclidean space answer queries in time $(d \log(N)/\epsilon^2)^{O(1)}$ using $(dN)^{O(1/\epsilon^2)}$ space [15, 12]. Other algorithms, with slower query times but lower space bounds are available as well.

The approximate nearest neighbor 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 [7], i.e., model 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, so the space of resulting images spans a one-dimensional line.

However, despite the importance of the problem, we are aware of only two results for the high dimensional variant of the problem (finding an approximate nearest k-flat) [7, 16]. 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 $(d + \log N + 1/\epsilon)^{O(1)}$. However, the space requirement of the algorithm is *quasi-polynomial*, of the form $2^{(\log N)^{O(1)}}$.

Our result. In this paper, 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 $(1 + \epsilon)r$ from q. We show:

Theorem 1.1. *For any sufficiently small fixed $\epsilon > 0$, there exists a data structure using $O(N + d)^{O(1/\epsilon^2)}$ space, and an algorithm that given a query point q, reports a $(1 + c\epsilon)$-approximate nearest line with probability at least $1 - \frac{6}{\log N}$ in time $(d + \log N + 1/\epsilon)^{O(1)}$.*

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 [15, 12]. 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. [1] 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.

[1] We note that the reductions to ANN used in this paper are randomized. However, the best algorithm for ANN for point sets is randomized as well.

Related work. In addition to the results in [7, 16], 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. [3] provided a data structure with query time of $O(d^3 N^{0.5+t})$ and space of $d^2 N^{O(1/\epsilon^2+1/t^2)}$ for any desired $t > 0$. We also note that in low dimensions several approximate algorithms for general polytope membership are known [5]. However, those algorithms have query time and space exponential in the dimension and therefore, our results are specially interesting in high dimensions.

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 1, 2 and 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.

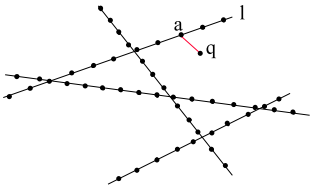

Figure 1: Net

Parallel Module We say that two lines ℓ_1, ℓ_2 are *almost parallel* if $\sin(\alpha) = O(\epsilon)$ where α is the angle between ℓ_1 and ℓ_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.

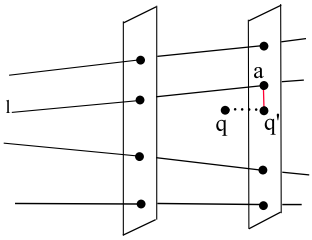

Figure 2: Parallel

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 $R = \frac{r}{\epsilon\delta}$ where $\delta < \epsilon$ so that $R > r$ (δ 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.

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

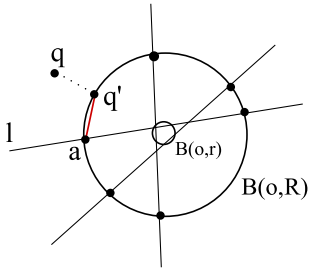

Figure 3: Unbounded

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 ℓ_1 and ℓ_2 s.t. $\sin \text{angle}(\ell_1, \ell_2) \geq \delta$, i.e., whose angle has $\sin$ value greater than δ. This allows us to either find a true *approximate* nearest line, or else find a line ℓ whose angle is close to the true nearest line ℓ^* (i.e. $\sin \text{angle}(\ell^*, \ell) < \delta$). Then we can further restrict our search to lines whose angles are close to ℓ 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 $(1 + O(\epsilon))$ 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 $(1 - \Theta(\epsilon))$ to the query point than ℓ. 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 ℓ. Suppose that the nearest line ℓ^* is not almost parallel to ℓ, i.e., $\sin \text{angle}(\ell, \ell^*) = \Omega(\epsilon)$. Let x be the distance of ℓ 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 $\mathcal{B}$ of polynomial size that depends on the input lines alone. The set $\mathcal{B}$ has the following property: for each query point q and a line ℓ with distance x to q, there exists a ball of radius $x/\epsilon^{O(1)}$ in $\mathcal{B}$ that contains all lines that are not almost parallel to ℓ and their distance to q is at most x. Furthermore, this ball can be retrieved in sublinear time. For each ball in $\mathcal{B}$, 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 ℓ^* is not almost parallel to ℓ. 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 ℓ), then their intersections with ℓ cannot be farther than $O(x/\epsilon)$ from q' (projection of q on ℓ). 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 ℓ and only retrieve the ones which are within $[-O(x/\epsilon), O(x/\epsilon)]$ 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 ℓ 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 ℓ. Then no matter where the query falls, we

[2] We will define the notion of distance we use here, later in section 6.1.

can retrieve the set of relevant lines (which are the lines within distance $O(x)$ of ℓ). Second, in each part, as we move along ℓ 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 δ (where δ depends on the radius of B_1). Suppose the result of this phase is ℓ. We then use the parallel modules to differentiate between the lines whose pairwise angle with ℓ have $\sin$ value smaller than δ. Further we can show that for each part a polynomial number of unbounded modules and parallel modules suffices.

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. [3] Let dist be the Euclidean distance. This induces the distance between two sets of points S_1 and S_2 defined as $\mathrm{dist}(S_1, S_2) = \min_{p \in S_1, p' \in S_2} \mathrm{dist}(p, p')$. We also define the distance between a point and a line $\mathrm{dist}(p, \ell) = \min_{p' \in \ell} \mathrm{dist}(p, p')$ and the distance between two lines $\mathrm{dist}(\ell, \ell') = \min_{p \in \ell, p' \in \ell'} \mathrm{dist}(p, p')$.

Definition 2.1. ***Approximate Nearest Neighbor*** *We define $ANN(P, \epsilon)$, for any point set P and error parameter ϵ, to be the Approximate Nearest Neighbor data structure constructed for P, with error parameter ϵ. Also we let $ANN_P(q)$ denote the approximate nearest point found by $ANN(P, \epsilon)$ given the query point q. Moreover, we let $\mathcal{S}(n, \epsilon)$, $\mathcal{T}(n, \epsilon)$ and $\mathcal{CT}(n, \epsilon)$ (respectively) denote the space bound, query time and data structure construction time (respectively) used by $ANN(P, \epsilon)$ for $|P| = n$.*

Furthermore, for a set of lines S, we define $\boldsymbol{\ell^*_S}$ to be the line in S that is closest to the query point q. Also for a point p, we use $\boldsymbol{\ell_p}$, 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 ℓ_p.

Let $B(o, r)$ denote the **ball** of radius r centered at o, i.e., $B(o, r) = \{p \in \mathbb{R}^d | \mathrm{dist}(o, p) \leq r\}$, and let $C(o, r)$ denote the surface of the ball $B(o, r)$, i.e., $C(o, r) = \{p \in \mathbb{R}^d | \mathrm{dist}(o, p) = r\}$. We also refer to it as a **sphere**. Moreover we define the **angle** between two lines ℓ_1 and ℓ_2 to be the smaller of the two angles between them, and we denote it by $\mathrm{angle}(\ell_1, \ell_2)$.

Definition 2.2. *For any two lines ℓ_1 and ℓ_2, and any value of $\delta \leq \epsilon$, we have the following definitions. ℓ_1 and ℓ_2 are **δ-close** (**strictly δ-close**, respectively) to each other if we have $\sin \mathrm{angle}(\ell_1, \ell_2) \leq \delta$ ($\sin \mathrm{angle}(\ell_1, \ell_2) < \delta$, respectively). Otherwise if $\sin \mathrm{angle}(\ell_1, \ell_2) \geq \delta$ ($\sin \mathrm{angle}(\ell_1, \ell_2) > \delta$, respectively), we say that the lines are **δ-far** (**strictly δ-far**, respectively) from each other. Also we say that the two lines are **almost parallel** if they are strictly 2ϵ-close to each other.*

For a set of lines S, a line ℓ, and $\delta \leq \epsilon$, we define $\boldsymbol{S_{\ell,\delta}} \subset S$ to be the subset of all lines in S that are strictly δ-close to ℓ. For any two lines ℓ_1 and ℓ_2, let the **closest point** $Cl_{\ell_1 \to \ell_2}$ be the point on the line ℓ_1 which is closest to ℓ_2. That is, $Cl_{\ell_1 \to \ell_2}$ and $Cl_{\ell_2 \to \ell_1}$ is the closest pair of points from the two lines. Note that the line $\overline{Cl_{\ell_1 \to \ell_2} Cl_{\ell_2 \to \ell_1}}$ is perpendicular to both ℓ_1 and ℓ_2.

We also generalize the **intersection** of a set of lines $\ell_1, \ell_2, \cdots, \ell_k$, to be the smallest ball which touches all lines. Also, for any collection $S = \{S_1, \cdots, S_t\}$ of subsets of $\mathbb{R}^d$, and another subset C of $\mathbb{R}^d$, we define their intersection as $S \cap C = \{S_1 \cap C, \cdots, S_t \cap C\}$. 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] 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.

3 Basic Modules

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

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 $\bar{oq}$. 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., $S \cap C(o, R)$. 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 $C(o, r)$ be any sphere which intersects all lines in S, i.e., $\forall \ell \in S : C(o, r) \cap \ell \neq \emptyset$. We construct an instance of ANN on the points obtained by intersecting each line with the larger sphere $C(o, R)$, where $R = \frac{r}{\epsilon\delta}$ for some value of $\delta \leq \epsilon$. That is, we construct $ANN(S \cap C(o, R), \epsilon)$. Given any query q outside of ball $B(o, R)$, let q' be the projection of q onto the sphere $C(o, R)$. The data structure and query processing algorithm 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 $ANN_{S \cap C(o,R)}(q')$. Also let ℓ_p be the line in S which corresponds to the point p, i.e., $p \in C(o, R) \cap \ell_p$. 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 $\delta \leq \epsilon$

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

Algorithm 2 Unbounded Module Query Processing

Input a query point q such that $q \notin B(o, R)$
Output a line which is either an approximate closest line to q, or is strictly δ-close to the closest line

1: $q' \leftarrow$ projection of q onto the sphere $C(o, R)$
2: $p \leftarrow ANN_P(q')$
3: $\ell_p \leftarrow$ the line in S which corresponds to p
4: Return ℓ_p

Lemma 3.1. *Let $\ell_{opt} = \ell^*_S$ be the closest line to q. Then for sufficiently small ϵ one of the following holds: either $\mathrm{dist}(q, \ell_p)$ is within $1 + O(\epsilon)$ factor of the closest distance $\mathrm{dist}(q, \ell_{opt})$, or the closest line ℓ_{opt} is strictly δ-close to ℓ_p.*

Proof. The intuition behind the proof is as follows. When the lines extend further than $\Theta(R) = \Theta(\frac{r}{\epsilon\delta})$ from the origin o, if two lines are δ-far from each other, then the distance between them becomes larger than $\Theta(R\delta) = \Theta(r/\epsilon)$. This intuitively means that if we translate the lines inside the smaller ball $C(o, r)$ such that they intersect the origin o, this only changes their distance by a factor of $O(r/(r/\epsilon)) = O(\epsilon)$. 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 $C(o, R)$ and computing the approximate point nearest neighbor on the ball results in an approximate nearest line.

The actual proof is tedious and therefore moved to Appendix A. □

Remark 3.2. *By Definition 2.1, this module uses space $\mathcal{S}(m, \epsilon)$ and has $\mathcal{T}(m, \epsilon)$ query time, where $m = 2|S|$ is the total number of points in the ANN data structure.*

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 ℓ which lies inside the twice larger ball $B(o, 2r)$. We then sample regularly spaced points from the segment s_ℓ. 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 $ANN(P, \epsilon)$. Given a query $q \in B(o, r)$, we find its approximate nearest neighbor p among the samples by calling $ANN_P(q)$. The data structure and query processing algorithm are shown in Algorithms 3 and 4.

Let ℓ_p be the line corresponding to p. Then the following lemma holds.

Algorithm 3 Net Module Data Structure

Input set of lines S
Input $B(o, r)$ which intersects all lines in set S
Input separation parameter x

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

Algorithm 4 Net Module Query Processing

Input a query point q such that $q \in B(o, r)$
Output a line which is either an approximate closest line, or its distance to query is at most x/ϵ

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

Lemma 3.3. *Let $\ell_{opt} = \ell^*_S$ be the closest line to q. Then for sufficiently small value of ϵ, either* $\mathrm{dist}(q, \ell_p) \leq \mathrm{dist}(q, \ell_{opt})(1 + O(\epsilon))$*, or* $\mathrm{dist}(q, \ell_p) < x/\epsilon$.

Proof. Suppose that $\mathrm{dist}(q, \ell_p) \geq x/\epsilon$. Let q_{opt} be the projection of q onto ℓ_{opt} and let p_{opt} be the closest sampled point of ℓ_{opt} to q_{opt}. It can easily be checked that q_{opt} is within the ball $B(o, 2r)$ and therefore $\mathrm{dist}(p_{opt}, q_{opt}) \leq x$. Then we have that

$$
\begin{aligned}
x/\epsilon &\leq \mathrm{dist}(q, \ell_p) \leq \mathrm{dist}(q, p) \\
&\leq \mathrm{dist}(q, p_{opt})(1 + \epsilon) \quad \text{(since } ANN \text{ has chosen } p \text{ over } p_{opt}\text{)} \\
&\leq [\mathrm{dist}(q, q_{opt}) + x](1 + \epsilon) \quad \text{(by triangle inequality)} \\
&\leq \mathrm{dist}(q, \ell_{opt})(1 + O(\epsilon)) \quad \text{(since, by what we just proved in previous line, } \mathrm{dist}(q, q_{opt}) \geq \tfrac{x/\epsilon}{(1+\epsilon)} - x \text{)}
\end{aligned}
$$

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

□

Remark 3.4. *By Definition 2.1, the space bound of this module is $\mathcal{S}(m, \epsilon)$ and its query time is $\mathcal{T}(m, \epsilon)$ where $m = |P| = |S|\lceil \frac{4r}{x} \rceil$ is the total number of points we build the ANN data structure for.*

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 δ-close to some base line ℓ_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 δ-close to ℓ_b for some parameter $\delta \leq \epsilon$. Let g be any hyperplane perpendicular to ℓ_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 $ANN(P, \epsilon)$. 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.

Let ℓ_p be the corresponding line to p. We then have the following lemma.

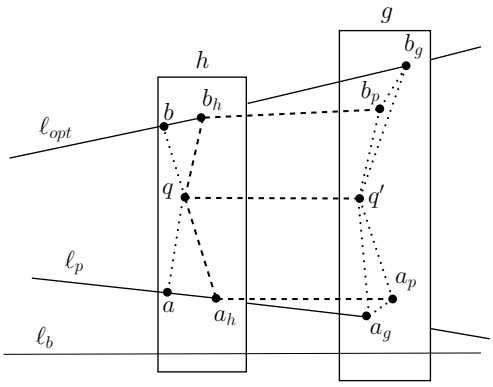

Figure 4: An illustration of Lemma 3.5

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 δ-close to ℓ_b
Input hyperplane g which is perpendicular to ℓ_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 $ANN(P, \epsilon)$

Algorithm 6 Parallel Module Query Processing

Input a query point q such that $\text{dist}(q, g) \leq D\epsilon/\delta$
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 $ANN_P(q')$
3: $\ell_p \leftarrow$ the line in S which corresponds to p
4: Return ℓ_p

Lemma 3.5. *Let $\ell_{opt} = \ell^*_S$ be the closest line to q in the set S. Then for any value of D, if* $\text{dist}(q, g) \leq D\epsilon/\delta$, *then either* $\text{dist}(q, \ell_p) \leq \text{dist}(q, \ell_{opt})(1 + O(\epsilon))$, *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 $(1 + O(\epsilon))$ of the optimum. Let h be the hyperplane parallel to g which passes through q. Let a and b be the closest points on ℓ_p and ℓ_{opt} to q respectively. Let a_h, a_g, b_h, b_g be the intersection of the lines ℓ_p and ℓ_{opt} with hyperplanes h and g as shown in Figure 4. Furthermore, let a_p, b_p be the projections of a_h, b_h on to the hyperplane g. Since $\sin \text{angle}(\ell_p, \ell_b) \leq \delta$ then

$$\text{dist}(a_p, a_g) = \text{dist}(a_h, a_p)(\tan \text{angle}(\ell_p, \ell_b)) \leq D\epsilon/\delta \times \frac{\delta}{\sqrt{1-\delta^2}} \leq \frac{D\epsilon}{\sqrt{1-\epsilon^2}} \leq 2D\epsilon$$

for sufficiently small value of ϵ. For the same reason we have $\text{dist}(b_p, b_g) \leq 2D\epsilon$. Also we have

$$\frac{\text{dist}(q, b_h)}{\text{dist}(q, b)} \leq \frac{1}{\cos(\widehat{bqb_h})} \leq \frac{1}{\sqrt{1-\delta^2}} \leq (1+\epsilon) \tag{1}$$

again for sufficiently small value of ϵ. Now

$D < \text{dist}(q, a)$ (by what we assumed)
$\leq \text{dist}(q, a_h)$ (since a is the closest point on ℓ_p to q)
$\leq \text{dist}(q', a_g) + 2D\epsilon$ (by triangle inequality and that $\text{dist}(q, a_h) = \text{dist}(q', a_p)$)
$\leq \text{dist}(q', a_g)(1 + O(\epsilon))$ (since by what is already proved in previous line, $\text{dist}(q', a_g) \geq D(1 - 2\epsilon)$)
$\leq \text{dist}(q', b_g)(1 + O(\epsilon))$ (since a_g is the result of ANN)
$\leq [\text{dist}(q, b_h) + 2D\epsilon](1 + O(\epsilon))$ (by triangle inequality and that $\text{dist}(q, b_h) = \text{dist}(q', b_p)$)
$\leq \text{dist}(q, b_h)(1 + O(\epsilon))$ (since by what is already proved in previous line, $\text{dist}(q, b_h) \geq D(1 - O(\epsilon))$)
$\leq \text{dist}(q, b)(1 + O(\epsilon))$ (by Equation 1)

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

□

Remark 3.6. *By Definition 2.1, the space bound of this module is $\mathcal{S}(m, \epsilon)$ and its query time is $\mathcal{T}(m, \epsilon)$ where $m = |P| = |S|$ is the total number of points we build the ANN data structure for.*

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 ℓ has distance x to the query point. Then the closest line is either strictly ϵ-close to ℓ or ϵ-far from ℓ. 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 6, for which we present an algorithm with poly-logarithmic time and polynomial space.

In the second case, each line ℓ' which is closer to q than ℓ, must intersect $B(q, x)$. As we show later, this means that $\text{dist}(\ell, \ell') \leq 2x$ and that $Cl_{\ell \to \ell'}$ is within $O(x/\epsilon)$ of the projection of q on ℓ. 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 $O(x/\epsilon)$. 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 $O(\epsilon)$. 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 5

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 ℓ in the input set S, it sorts the other lines in S by their distance to ℓ. This will help us later to retrieve the set of lines which are within a distance at most $2x$ from the given line ℓ. Let S_i be the set of the i lines in $S \setminus \{\ell\}$ that are closest to ℓ. The algorithm sorts the lines in S_i based on the position of the point of ℓ that is closest to them. That is, we orient ℓ arbitrarily in one of the two ways, and for all lines $\ell' \in S_i$ we sort them based on the position of $Cl_{\ell \to \ell'}$ along ℓ towards the chosen direction. This will help us retrieve all lines whose closest point on ℓ lies within some range. Further, let A be any sequence of consecutive lines (an interval) in S_i. Let A^- be the union of ℓ and the set of lines in A that are ϵ-far from ℓ. We then find the smallest ball with a center in ℓ 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 ϵ (note that, this module is used to handle queries outside of the larger ball $B(o_A, Y_A/\epsilon^2)$). Second, we build a net module for the same set of lines A^- and the larger ball $B(o_A, Y_A/\epsilon^2)$ with the separation parameter $Y_A\epsilon^3$. Finally, for each line ℓ, we find all lines that are strictly ϵ-close to it (and therefore strictly (2ϵ)-close to each other) and create the $APNLS$ data structure for them as described in Algorithm 9.

Algorithm 7 NLS Data Structure Construction

Input a set of lines S

$T \leftarrow |S|/2$ lines chosen at random from S.
Recursively build the NLS data structure for T.
for each line $\ell \in S$ **do**
  Sort the lines in $S \setminus \{\ell\}$ based on their distance from ℓ in a non-decreasing order
  for $i = 1$ **to** $|S| - 1$ **do**
    $S_i \leftarrow$ the closest i lines to ℓ in $S \setminus \{\ell\}$
    Sort all lines in S_i based on the position of the closest point $Cl_{\ell \to \ell'}$ on the line ℓ.
    for each of the $\binom{|S_i|}{2}$ intervals of lines A **do**
      $A^- \leftarrow \ell \cup$ (all lines $\ell' \in A$ that are ϵ-far from ℓ)
      $B(o_A, Y_A) \leftarrow$ the smallest ball that intersects all lines in A^- and that $o_A \in \ell$
      $UM_A \leftarrow$ Build unbounded module for lines in A^-, the ball $B(o_A, Y_A)$ and the parameter ϵ
      $NM_A \leftarrow$ Build net module for lines in A^-, the ball $B(o_A, Y_A/\epsilon^2)$, and the separation parameter $Y_A\epsilon^3$
    end for
  end for
  Build $APNLS$ data structure (described in Section 6) for the set of lines $S_{\ell,\epsilon}$
end for

Lemma 4.1. *The data structure uses $(N + d)^{O(1/\epsilon^2)}$ space. (Proof in Appendix B.1)*

Lemma 4.2. *The data structure can be constructed in time $(N + d)^{O(1/\epsilon^2)}$.(Proof in Appendix B.2)*

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 ϵ-close to ℓ, i.e., $S_{\ell,\epsilon}$ and stores it in ℓ_{close}. This is done by invoking $APNLS(S_{\ell,\epsilon}, q)$.

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., $Cl_{\ell\to\ell'}$, lies within a distance of $3x/\epsilon$ 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., $q \notin B(o_A, Y_A/\epsilon^2)$. If so, it uses the module followed by a call to the $APNLS$ of the found line and sets the value of ℓ_{far}. Otherwise the point q lies inside the ball $B(o_A, Y_A/\epsilon^2)$ and thus it uses the net module in order to retrieve a better line and updates the value of ℓ_{far} with it. Note that ℓ_{far} shows the candidate among the set of lines in A which are ϵ-far from ℓ. Then at the end of the iteration, the value of ℓ is updated with the best of ℓ_{close} and ℓ_{far} if any of them can improve ℓ.

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

if $|S| \leq \log^3 N$ **then**
  return the nearest line found by the brute force.
end if
$T \leftarrow$ the set of lines sampled during the preprocessing stage
$\ell \leftarrow$ the result of executing $NLS(T, q)$
for $(\log |S|)$ times **do**
  $\ell_{close} \leftarrow$ the result of running $APNLS(S_{\ell,\epsilon}, q)$
  $x \leftarrow \text{dist}(q, \ell)$
  $q_\ell \leftarrow$ the projection of q on to ℓ
  Use binary search to find the largest i such that for all lines $\ell' \in S_i$, we have $\text{dist}(\ell', \ell) \leq 2x$
  Use binary search to find the largest interval of lines A in S_i, s.t. $\forall \ell' \in A : \text{dist}(q_\ell, Cl_{\ell\to\ell'}) \leq 3x/\epsilon$
  if A is not empty **then**
    if $q \notin B(o_A, Y_A/\epsilon^2)$ **then**
      $\ell_r \leftarrow$ result of the unbounded module $UM_A(q)$
      $\ell_{far} \leftarrow$ the best of ℓ_r and $APNLS(S_{\ell_r,\epsilon}, q)$.
    else
      $\ell_{far} \leftarrow$ result of the net module $NM_A(q)$.
    end if
  end if
  $\ell \leftarrow$ the best of ℓ , ℓ_{close} and ℓ_{far}
end for
return ℓ

Lemma 4.3. *The query processing algorithm runs in time* $(\log N + d + 1/\epsilon)^{O(1)}$. *(Proof in Appendix B.3)*

5 Analysis of NLS Algorithm

Definition 5.1. *An invocation $NLS(L, q)$ 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 APNLS in any level of recursion succeeds, i.e., they correctly report an approximate closest line.*

Lemma 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 $APNLS$ subroutine. By Theorem 6.10, the probability of failure of each call to $APNLS$ is at most $\frac{2}{\log^3 N}$. Since there are at most $\log N$ recursive levels, the total number of times we call $APNLS$ is at most $2 \log^2 N$, and thus the probability that all calls to $APNLS$ are successful is at least $1 - \frac{4 \log^2 N}{\log^3 N}$. So the probability that the invocation is successful is at least $1 - \frac{2}{\log^3 N} - \frac{4}{\log N} \geq 1 - \frac{6}{\log N}$. □

Lemma 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 ℓ is an approximate nearest line ($\mathrm{dist}(q, \ell) \leq \mathrm{dist}(q, \ell^*)(1 + c\epsilon)$), or its distance to the query q has been decreased by a factor of 4ϵ.*

Proof. First observe that if ℓ is a $(1 + c\epsilon)$-approximate nearest line at the beginning of the iteration, it will remain a $(1 + c\epsilon)$-approximate nearest line in all the following iterations as well. This holds since we only perform an update if the new line improves over ℓ. Suppose that ℓ is not yet the approximate closest line. Then one of two cases can occur. The first case is that ℓ^* is strictly ϵ-close to the line ℓ. Since we assume the invocation of the algorithm is successful, the call to $APNLS$ in line 7 correctly finds ℓ_{close}, which is an approximate closest line (so we should set $c \geq c_{\mathrm{aplnn}}$, where c_{aplnn} is defined by Theorem 6.10). Thus ℓ will be updated with an approximate solution at the end of the iteration and the lemma is proved.

Now suppose that ℓ^* is ϵ-far from q. We know that $\mathrm{dist}(q, \ell^*) < \mathrm{dist}(q, \ell) = x$ and therefore by triangle inequality, $\mathrm{dist}(\ell, \ell^*) \leq 2x$. 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 ℓ^* is included in the set A that the algorithm finds in line 11.

Claim 5.4. *If ℓ^* is ϵ-far from the line ℓ, then $\mathrm{dist}(Cl_{\ell \to \ell^*}, q_\ell) \leq 3x/\epsilon$ for sufficiently small value of ϵ.*

Proof. Let q_ℓ be the projection of q onto ℓ and let b be the projection of q onto ℓ^*. Let b_ℓ denote the projection of b onto ℓ (see Figure 5). It is easy to see that

$$\mathrm{dist}(b, b_\ell) \leq \mathrm{dist}(b, q) + \mathrm{dist}(q, \ell) \leq 2x \tag{2}$$

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

Now, suppose that $\mathrm{dist}(q_\ell, a_\ell) > 3x/\epsilon$. Then since $\overline{qq_\ell}$ and $\overline{bb_\ell}$ are perpendicular to ℓ, then we have $\mathrm{dist}(b_\ell, q_\ell) \leq \mathrm{dist}(b, q) \leq x$. Hence by triangle inequality $\mathrm{dist}(b_\ell, a_\ell) > (3x/\epsilon - x) > 2x/\epsilon$. Therefore using the fact that the two lines are ϵ-far, we have

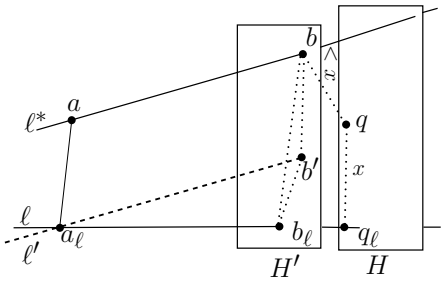

Figure 5: Figure of Claim 5.4

$$\mathrm{dist}(b_\ell, b') = \mathrm{dist}(a_\ell, b_\ell) \tan(\widehat{b_\ell a_\ell b'}) > \frac{2x}{\epsilon} \cdot \frac{\epsilon}{\sqrt{1-\epsilon^2}} > 2x \tag{3}$$

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

Thus the optimal line ℓ^* is 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 5.5. *If ℓ^* is contained in the set A and is ϵ-far from ℓ, then ℓ_{far} is either a $(1 + c\epsilon)$-approximate closest line or its distance to query is at most $4\epsilon x$.*

Proof. If ℓ^* is in the set A and is ϵ-far from ℓ then it is also contained in the set A^-. Then one of the following two cases can occur.

Case 1: q is outside of the ball $B(o_A, Y_A/\epsilon^2)$. In this case, by Lemma 3.1, it suffices to consider two scenarios. If $\mathrm{dist}(q, \ell_r) \leq \mathrm{dist}(q, \ell^*)(1 + c_1\epsilon)$ (where c_1 is determined by Lemma 3.1), then ℓ_r and thus ℓ_{far} will be a $(1 + c_1\epsilon)$-approximate nearest line. So we should set $c \geq c_1$. Second if ℓ^* is strictly ϵ-close to ℓ_r, then in this case since we have a successful invocation, $APNLS$ finds an approximate nearest line and thus ℓ_{far} is a $(1 + c\epsilon)$-approximate closest line. For this we need to set $c \geq c_{\mathrm{aplnn}}$ where c_{aplnn} is the determined by Theorem 6.10 . Note that $APLNN$ reports the $(1 + c_{\mathrm{aplnn}}\epsilon)$-approximate closest line.

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

$$\mathrm{dist}(q_\ell, \ell') \leq \mathrm{dist}(q_\ell, Cl_{\ell \to \ell'}) + \mathrm{dist}(Cl_{\ell \to \ell'}, \ell') \leq 3x/\epsilon + \mathrm{dist}(\ell, \ell') \leq 3x/\epsilon + 2x \leq 4x/\epsilon$$

where we have used the fact that $\ell' \in S_i$ and thus $\mathrm{dist}(\ell, \ell') \leq 2x$ (by the line 10 of the Algorithm 8). Therefore, since $B(q_\ell, 4x/\epsilon)$ touches all lines in A^-, then we have that $Y_A \leq 4x/\epsilon$. Thus, the separation parameter of the net module is $Y_A\epsilon^3 \leq 4x\epsilon^2$. Then by Lemma 3.3 one of the following two cases can occur. First if $\mathrm{dist}(q, \ell_{far}) \leq \mathrm{dist}(q, \ell^*)(1 + c_2\epsilon)$ (where c_2 is determined by Lemma 3.3), then ℓ_{far} is updated with an approximate solution (so we should set $c \geq c_2$). On the other hand, if $\mathrm{dist}(q, \ell_{far}) < 4x\epsilon$, then we are done. □

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

Theorem 5.6. *Let L be a set of N lines in Euclidean space $\mathbb{R}^d$. Then for any sufficiently small fixed $\epsilon > 0$, there exists a data structure using $O(N+d)^{O(1/\epsilon^2)}$ space, and an algorithm that given a query point q, reports a $(1+c\epsilon)$-approximate nearest line with probability at least $1-\frac{6}{\log N}$ in time $(d+\log N+1/\epsilon)^{O(1)}$.*

Proof. The proof follows from Lemmas 4.3, 4.1, 5.2 and 5.3 and the following argument.

Suppose that we have a successful invocation, which by Lemma 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 ℓ_T. By induction we know that ℓ_T is a $(1+c\epsilon)$-approximate closest line in the set T, and thus $\mathrm{dist}(q,\ell_T) \le \mathrm{dist}(q,\ell_{\log n})(1+c\epsilon)$.

Let $c = \max\{c_1, 8\}$ where c_1 is determined by Lemma 5.3 and let $\epsilon < 1/c \le 1/8$. Also let ℓ be as defined in Algorithm 8. Then by the end of the first improvement step, by Lemma 5.3 we have that either ℓ is a $(1+c_1\epsilon)$-approximate (and thus $(1+c\epsilon)$-approximate) closest line in the set S (since $c_1 \le c$), or

$$\mathrm{dist}(q,\ell) \le \mathrm{dist}(q,\ell_T)4\epsilon \le \mathrm{dist}(q,\ell_{\log n})(1+c\epsilon)(4\epsilon) < \mathrm{dist}(q,\ell_{\log n})$$

So by the end of the first iteration, we have improved not only over the line ℓ_T but also on the line $\ell_{\log n}$. In the following $\log|S|-1$ iterations, by Lemma 5.3, if a $(1+c_1\epsilon)$-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 $(1+c\epsilon)$-approximate nearest line to the query.

□

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 ℓ 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 ℓ. For each slab there is a unique ordering of the set of lines by their projective distance from ℓ. This enables the query procedure to identify all lines within a distance $O(x)$ to ℓ . Specifically, we sort the lines in each slab based on their projective distance to ℓ 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 δ-close to ℓ. Then (by Parallel module Lemma 3.5) the required density of parallel modules is proportional to δ. On the other hand, if all lines are δ-far from ℓ, then by Lemma 3.1 we can find an approximate closest line using the unbounded module for queries that are farther than $r/\epsilon\delta$.

Let $\epsilon = \delta_0 > \delta_1 > \delta_2 > \cdots > \delta_t$ 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 $\frac{r}{\epsilon\delta_0}, \frac{r}{\epsilon\delta_1}, \cdots, \frac{r}{\epsilon\delta_t}$ 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 δ_j-far, returning a line ℓ'. We are left with a set of lines that are strictly δ_j-close to ℓ' and therefore δ_{j+1}-close to ℓ'. It now suffices to cover the inner part of B_{j+1} with parallel modules with density proportional to δ_{j+1} as opposed to δ_j. This enables us to bound the space. After finding ℓ' the algorithm uses the parallel module to find the approximate nearest line.

6.1 Data Structure

The construction procedure for the APLNS 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 ℓ in the set S, we build a data structure that supports an iterative improvement procedure.

Definition 6.1. *Let $H(\ell, p)$ denote the hyperplane perpendicular to a line ℓ which passes through a point p . Furthermore, for any sets of points a and b, let $\mathrm{dist}_H(a, b) = \mathrm{dist}(H \cap a, H \cap b)$ be the distance between the two sets on the hyperplane H (note that the intersection should not be empty). It is clear that $\mathrm{dist}_H(a, b) \geq \mathrm{dist}(a, b)$.*

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

Definition 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 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 ℓ 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 $\mathrm{dist}_{H(\ell,p)}(\ell, \ell_1) \geq \mathrm{dist}_{H(\ell,p)}(\ell, \ell_2)$, or it is always the case that $\mathrm{dist}_{H(\ell,p)}(\ell, \ell_1) \leq \mathrm{dist}_{H(\ell,p)}(\ell, \ell_2)$.*

Lemma 6.3. *There exists a partitioning with at most $O(n^2)$ segments.*

Proof. For simplicity suppose that ℓ is identified with the X-axis. Given two lines ℓ_1 and ℓ_2 which are strictly $O(\epsilon)$-close to the X-axis, let $p_1(x)$ and $p_2(x)$ be the points on ℓ_1 and ℓ_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 ℓ 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 $7\binom{n}{2} \leq 4n^2$. □

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 ℓ 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 $\mathrm{dist}_{H(\ell,p)}(\ell, \ell_i)$ 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. Second, for all other lines $\ell' \in S_i$, we have $\mathrm{dist}_{H(\ell,o)}(\ell, \ell') \leq \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$. Therefore, the ball $B(o, r)$ intersects all lines in the prefix set S_i, where $r = \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$.

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 $\delta_0 = \epsilon$ and $\delta_{t+1} = 0$, and we ignore those values that are not between 0 and ϵ. 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 $R_j = r/(\epsilon\delta_j)$ 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

Algorithm 9 APNLS Data Structure

Input a set of lines S that are strictly (2ϵ)-close to each other

$T \leftarrow$ half of the lines in S chosen randomly.
Recursively build the APLNN data structure for T.
for $\ell \in S$ **do**
  Partition ℓ 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.
  for $0 \leq m \leq k-1$ **do**
    sort all lines $\ell' \in S$ by $\mathrm{dist}_{H(\ell,o_m)}(\ell, \ell')$ in a non-decreasing order
    for $i = 1$ **to** $|S|$ **do**
      $S_i \leftarrow$ the first i lines in S
      $o \leftarrow \arg\min_{o \in \{o_m, o_{m+1}\}} \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$
      $r \leftarrow \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$
      let $\delta_0 > \delta_1 > \cdots > \delta_t > \delta_{t+1} = \{\delta < \epsilon \mid \exists\, \ell', \ell" \in S_i, \delta = \sin\mathrm{angle}(\ell', \ell")\} \cup \{0, \epsilon\}$ be the sin values (between 0 and ϵ) of the pairwise angles.
      $\forall\, 0 \leq j \leq t \;:\; R_j \leftarrow \frac{r}{\epsilon\delta_j}$
      $NM_0 \leftarrow$ Build the net module for S_i , the ball $B(o, R_0)$, and separation $r\epsilon^2$
      for $1 \leq j \leq t+1$ **do**
        $UM_{j-1} \leftarrow$ Build the unbounded module for lines S_i and ball $B(o, r)$, and parameter δ_{j-1}
        for $\ell' \in S_i$ **do**
          if $j = t+1$ **then**
            Let $G_{\ell',j}$ be the two hyperplanes perpendicular to ℓ' at positions $\ell' \cap C(o, R_{j-1})$.
          else
            Let $G_{\ell',j}$ be a set of hyperplanes perpendicular to ℓ' with separation $R_j\epsilon^3$, on the segment $\ell' \cap B(o, 2R_j)$.
          end if
          for each hyperplane $g \in G_{\ell',j}$ **do**
            $PM_{\ell',g} \leftarrow$ Parallel Module for the set of lines $(S_i)_{\ell',\delta_{j-1}}$, base line ℓ', and hyperplane g
          end for
        end for
      end for
    end for
  end for
end for

set of lines S_i and with separation parameter $R\epsilon^2$. 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 δ_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 δ_j-close to ℓ'. Note that since there are no other angles between δ_j and δ_{j-1}, it means that this set of lines is equal to the set of lines that are strictly δ_{j-1}-close to ℓ', i.e., $(S_i)_{\ell',\delta_{j-1}}$. We take a set of regularly spaced hyperplanes perpendicular to ℓ', separated by $R_j\epsilon^3$, that cover the part of ℓ' 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 ℓ'. Therefore it is enough to have only one hyperplane perpendicular to ℓ' 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 ℓ' and the hyperplane g.

Lemma 6.4. *The data structure for a set of lines $L' \subset L$ uses $(|L'| + d)^{O(1/\epsilon^2)}$ space. (Proof in Appendix B.4)*

Lemma 6.5. *The data structure for a set of lines $L' \subset L$ can be constructed in time $(|L'|+d)^{O(1/\epsilon^2)}$. (Proof in Appendix B.5)*

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 ℓ be the approximate nearest line in the sampled set T. The algorithm starts with ℓ and keeps improving it so that the distance from the query to ℓ 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 ℓ and let q_ℓ be the projection of q onto the line ℓ. Then using binary search we find the segment $[o_m, o_{m+1}]$ that q_ℓ 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 ℓ is at most $2x$ and its distance to ℓ 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 ℓ 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 ℓ_i and ℓ is minimized on the hyperplane $H(\ell, o)$, and denote this distance by $r = \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$.

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 ℓ_{um}. Then we know that either ℓ_{um} is an approximate nearest line or the nearest line is strictly δ_j-close to ℓ_{um}, or equivalently δ_{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 ℓ_{pm}. We update ℓ with this line ℓ_{pm} (if it provides an improvement). Finally, after all iterations, we report the line ℓ as the approximate nearest line. Figure 6 shows an example of the data structure and the query point.

Lemma 6.6. *The query processing algorithm on the set $L' \subset L$ runs in time $(\log |L'|+d+1/\epsilon)^{O(1)}$. (Proof in Appendix B.6)*

6.3 Analysis of APNLS Algorithm

Definition 6.7. *An invocation $APNLS(L', q)$ (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 6.8. *The probability that a given invocation on set $L' \subset L$ is successful is at least $1 - \frac{2}{\log^3 N}$ 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

Algorithm 10 APNLS Query Processing

Input a set of lines S that are strictly 2ϵ-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: $x \leftarrow \mathrm{dist}(q, \ell)$
8: $q_\ell \leftarrow$ the projection of q onto ℓ.
9: Use binary search to find the segment $[o_m, o_{m+1}]$ containing q_ℓ.
10: Use binary search to find the largest i such that for all lines $\ell' \in S_i$, we have $\mathrm{dist}_{H(\ell,q_\ell)}(\ell', \ell) \leq 3x$
11: $o \leftarrow \arg\min_{o \in \{o_m, o_{m+1}\}} \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$
12: $r \leftarrow \mathrm{dist}_{H(\ell,o)}(\ell, \ell_i)$
13: **if** $q \in B(o, R_0)$ **then**
14: $\ell \leftarrow$ the best of ℓ and ℓ_{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 $q \notin B(o, R_j)$
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 ℓ with the best of ℓ and ℓ_{pm}
21: **end if**
22: **end for**
23: **return** ℓ

$\ell_1, \cdots, \ell_{\log n}$ 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

$$1 - \frac{1}{|L'|} - \frac{1}{|L|/2} - \frac{1}{|L'|/4} - \cdots - \frac{1}{\log^3 N} \geq 1 - \frac{1}{N} - \frac{1}{N/2} - \frac{1}{N/4} - \cdots - \frac{1}{\log^3 N} \geq 1 - \frac{2}{\log^3 N}$$

□

Lemma 6.9. *Suppose that the invocation of the $APNLS(L', q)$ is successful for $L' \subset L$. Let S be the set of lines at some level of the recursion and let $\ell^* = \ell^*_S$ be the closest line in S to the query point. Then by the end of each iteration, either, ℓ is an approximate nearest line ($\mathrm{dist}(q, \ell) \leq \mathrm{dist}(q, \ell^*)(1 + c\epsilon)$), or its distance to the query q has been decreased by a factor of 3ϵ.*

Proof. First observe that if ℓ is a $(1 + c\epsilon)$-approximate nearest line at the beginning of the iteration, it will remain a $(1 + c\epsilon)$-approximate nearest line in the following iterations as well. This is because we only perform an update if the new line improves over ℓ. Suppose that ℓ is not yet the approximate closest line.

Since $\mathrm{dist}(q, \ell) = x$, we know that $\mathrm{dist}(q, \ell^*) \leq x$ and therefore we have $\mathrm{dist}(q_\ell, \ell^*) \leq 2x$. Hence, since ℓ and ℓ^* are strictly 2ϵ-close to each other, we have that $\mathrm{dist}_{H(\ell,q_\ell)}(\ell, \ell^*) \leq \frac{2x}{\sqrt{1-4\epsilon^2}} \leq 3x$ for sufficiently small value of ϵ. Therefore, the set of lines S_i that we retrieve in line 10 contains the optimal line ℓ^*.

Furthermore, the value of $\mathrm{dist}_{H(\ell,p)}(\ell, \ell_i)$ increases as we distance p from o inside the segment $[o_m, o_{m+1}]$. Specifically for $p = q_\ell$, it means that $r \leq 3x$.

Now, if $q \in B(o, R_0)$, then by the net module Lemma 3.3 one of the two cases can occur. The first case occurs if $\mathrm{dist}(q, \ell_{nm}) \leq \mathrm{dist}(q, \ell^*)(1 + c_1\epsilon)$ (where c_1 is defined by Lemma 3.3), which means that the value ℓ 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 $\mathrm{dist}(q, \ell_{nm}) \leq r\epsilon^2/\epsilon \leq 3x\epsilon$, which means that we have reduced the distance $\mathrm{dist}(q, \ell)$ by a factor of 3ϵ.

Otherwise, we are using the unbounded module of UM_j. This means that by Lemma 3.1, either $\mathrm{dist}(q, \ell_{um}) \leq \mathrm{dist}(q, \ell^*)(1 + c_2\epsilon)$ (where c_2 is determined by Lemma 3.1), or $\ell^* \in (S_i)_{\ell_{um},\delta_j}$, i.e. the optimal line is strictly δ_j-close to the line ℓ_{um}. However, in both cases we know that there exists a line ℓ' (ℓ_{um} in the first case and ℓ^* otherwise) that is strictly δ_j-close to ℓ_{um} and that $\mathrm{dist}(q, \ell') \leq \mathrm{dist}(q, \ell^*)(1 + c_2\epsilon)$. Since there is no other pairwise angle α such that $\delta_{j+1} < \sin\alpha < \delta_j$, it means that ℓ' is δ_{j+1}-closet to ℓ_{um}. Now, note that if $j = t$, then the set $(S_i)_{\ell_{um},\delta_j}$ are all parallel to ℓ_{um}, and therefore, using any Parallel module perpendicular to ℓ_{um} returns an approximate nearest line with approximation factor $(1 + c_2\epsilon)(1 + \epsilon) \leq (1 + (c_2 + 2)\epsilon)$. Thus, we have $c \geq c_2 + 2$.

Otherwise if $j < t$, we use Lemma 3.5 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 $R\epsilon^3 = \frac{r\epsilon^2}{\delta_j}$ and therefore Lemma 3.5 applies. Using the lemma one can see that either $\mathrm{dist}(q, \ell_{pm}) \leq \mathrm{dist}(q, \ell')(1 + c_3\epsilon) \leq \mathrm{dist}(q, \ell^*)(1 + (c_2 + c_3 + 1)\epsilon)$ (where c_3 is defined by Lemma 3.5), in which case by the end of the iteration ℓ is an approximate nearest line (we should set $c \geq c_2 + c_3 + 1$), or $\mathrm{dist}(q, \ell_{pm}) \leq r\epsilon \leq 3x\epsilon$ which means that we have decreased the value of $\mathrm{dist}(q, \ell)$ by a factor of 3ϵ. □

Theorem 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 $\epsilon > 0$, there exists a data structure using $O(|L'| + d)^{O(1/\epsilon^2)}$ space, and an algorithm that given a query point q, reports a $(1 + c\epsilon)$-approximate nearest line with probability at least $1 - \frac{2}{\log^3 N}$ in time $(d + \log|L'| + 1/\epsilon)^{O(1)}$.*

Proof. The proof follows from Lemmas 6.6, 6.4, 6.8 and 6.9 and the following argument which is analogous to the proof of Theorem 5.6

Suppose that we have a successful invocation (which by Lemma 6.8 happens with probability $1 - \frac{2}{\log^3 N}$). 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 ℓ_T. By induction we know that ℓ_T is a $(1 + c\epsilon)$-approximate closest line in the set T, and thus $\mathrm{dist}(q, \ell_T) \leq \mathrm{dist}(q, \ell_{\log n})(1 + c\epsilon)$.

Let $c = \max\{c_1, 6\}$ where c_1 is determined by Lemma 6.9 and let $\epsilon < 1/c \leq 1/6$. Also let ℓ be as defined in Algorithm 10. Then by the end of the first improvement step, by Lemma 6.9 we have that either ℓ is a $(1 + c_1\epsilon)$-approximate (and thus a $(1 + c\epsilon)$-approximate) nearest line in the set S or

$$\mathrm{dist}(q, \ell) \leq \mathrm{dist}(q, \ell_T)3\epsilon \leq \mathrm{dist}(q, \ell_{\log n})(1 + c\epsilon)(3\epsilon) < \mathrm{dist}(q, \ell_{\log n})$$

So by the end of the first iteration, we have improved not only over the line ℓ_T but also over the line $\ell_{\log n}$. In the following $\log n - 1$ iterations, by Lemma 6.9, if a $(1 + c_1\epsilon)$-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 $(1 + c\epsilon)$-approximate nearest line to the query.

□

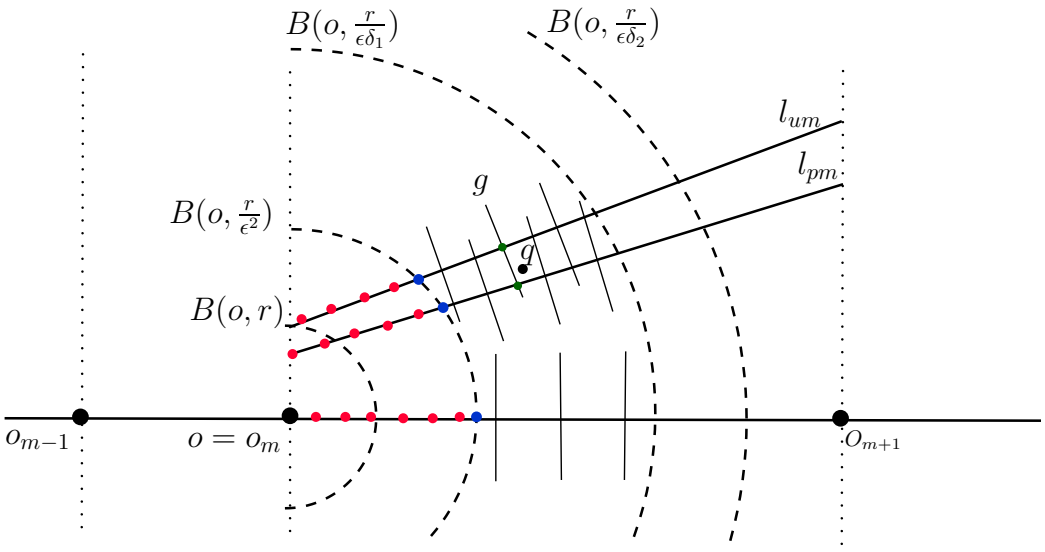

Figure 6: Example: There are only three lines with sin values of $2\epsilon, \delta_1$ and δ_2. $B(o, r)$ touches all lines. The red dots show some of the sampled points in the net module inside $B(o, r/\epsilon^2)$. The blue dots on the ball $B(o, r/\epsilon^2)$ show some of the points in the unbounded module UM_0 which is used for the query point q. Suppose that the output of this module is ℓ_{um}. Then we take the closest hyperplane g to q which is perpendicular to ℓ_{um}. The points of the parallel module on g are depicted by green dots. The output of the parallel module is denoted by ℓ_{pm}. Note that not all the hyperplanes are shown in the picture.

7 Acknowledgements

The results were developed in joint work with Piotr Indyk and the author would like to thank him for his contribution and several discussions and suggestions on this work.
The author would also like to thank Maryam Aliakbarpour, Arturs Backurs, Mohsen Ghaffari, Ilya Razenshteyn, and Ali Vakilian for useful comments and feedbacks on this paper.

References

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 557–563. ACM, 2006.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[3] A. Andoni, P. Indyk, R. Krauthgamer, and H. L. Nguyen. Approximate line nearest neighbor in high dimensions. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 293–301. Society for Industrial and Applied Mathematics, 2009.

[4] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn. Beyond locality-sensitive hashing. *Proceedings of the twenty-fifth ACM-SIAM Symposium on Discrete Algorithms*, 2014.

[5] S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate polytope membership queries. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 579–586. ACM, 2011.

[6] 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 fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 573–582. Society for Industrial and Applied Mathematics, 1994.

[7] R. Basri, T. Hassner, and L. Zelnik-Manor. Approximate nearest subspace search with applications to pattern recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[8] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 473–482. IEEE, 2004.

[9] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[10] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *focs*, volume 1, pages 94–103, 2001.

[11] P. Indyk. Nearest neighbors in high-dimensional spaces. 2004.

[12] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[13] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 599–608. ACM, 1997.

[14] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

[15] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[16] A. Magen. Dimensionality reductions that preserve volumes and distance to affine spaces, and their algorithmic applications. In *Randomization and approximation techniques in computer science*, pages 239–253. Springer, 2002.

[17] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1186–1195. ACM, 2006.

[18] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[19] G. Shakhnarovich, T. Darrell, and P. Indyk. Nearest-neighbor methods in learning and vision. *IEEE Transactions on Neural Networks*, 19(2):377, 2008.

A Proof of Lemma 3.1

We start by defining a set of notations as shown in Figure 7. 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 $C(o, R)$. Let p be the point returned by the ANN and let ℓ_p be the corresponding line to p. Let u be the intersection of ℓ_p with the sphere $C(o, r_q)$ such that both p and u fall on the same side of $C(o, r)$. Also let ℓ_{opt} denote the actual closest line to the query point q and let s and t be its intersection with $C(o, R)$ and $C(o, r_q)$ respectively. Furthermore, let ℓ'_p and ℓ'_{opt} be the lines parallel to ℓ_p and ℓ_{opt} that pass through the origin, and let p', s', u', t' be their intersections with spheres $C(o, R)$ and $C(o, r_q)$. Finally, let $\alpha_1 = \widehat{qou'}, \alpha_2 = \widehat{qot'}$ be the angles between the line oq and the lines ℓ_p and ℓ_{opt} and let $\alpha = \widehat{u'ot'}$ be the angle between ℓ_p and ℓ_{opt} (or equivalently between ℓ'_p and ℓ'_{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 ℓ_p approximately, i.e., $\alpha_1 \leq 2\pi/3$ for small enough ϵ, otherwise the ANN would have chosen the other point of the intersection $\ell_p \cap C(o, R)$ instead of choosing p.

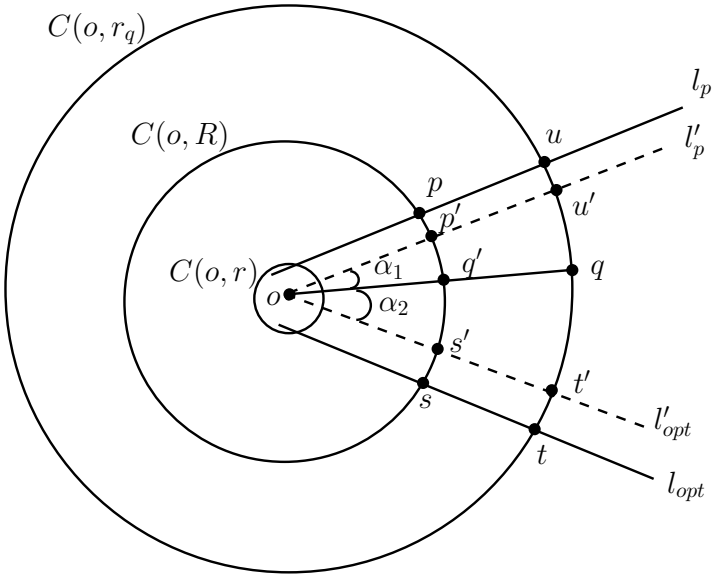

Figure 7: Figure of Lemma 3.1

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

$$\begin{aligned}
\frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} &\leq \frac{\text{dist}(q, \ell'_p)}{\text{dist}(q, \ell'_{opt})}(1 + O(\epsilon)) \quad \text{(by Lemma A.2)} \\
&\leq \frac{\text{dist}(q, u')}{\text{dist}(q, t')}(1 + O(\epsilon)) \quad \text{(by Lemma A.5)} \\
&\leq \frac{\text{dist}(q', p')}{\text{dist}(q', s')}(1 + O(\epsilon)) \quad \text{(immediate)} \\
&\leq (1 + O(\epsilon)) \quad \text{(by Lemma A.6)}
\end{aligned}$$

Let us start by the following observation

Observation A.1. $\mathrm{dist}(p,p') \le 2r$, *(and similarly* $\mathrm{dist}(u,u'), \mathrm{dist}(s,s'), \mathrm{dist}(t,t') \le 2r$*).*

Proof.

$$\mathrm{dist}(p,p') = 2R\sin\left(\frac{\widehat{pop'}}{2}\right) \le 2R\sin\left(\widehat{pop'}\right) \le 2r$$

□

Lemma A.2. $\frac{\mathrm{dist}(q,\ell_p)}{\mathrm{dist}(q,\ell_{opt})} \le \frac{\mathrm{dist}(q,\ell'_p)}{\mathrm{dist}(q,\ell'_{opt})}(1+O(\epsilon))$

Proof. Since the distance between the two pairs of parallel lines is at most r, by triangle inequality

$$\frac{\mathrm{dist}(q,\ell_p)}{\mathrm{dist}(q,\ell_{opt})} \le \frac{\mathrm{dist}(q,\ell'_p)+r}{\mathrm{dist}(q,\ell'_{opt})-r}$$

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

Claim A.3. $\mathrm{dist}(q,\ell'_p) \ge \frac{r}{3\epsilon}$ *for sufficiently small value of* ϵ*.*

Proof. By definition, $\mathrm{dist}(q,\ell_{opt}) \le \mathrm{dist}(q,\ell_p)$ and therefore by triangle inequality

$$\mathrm{dist}(q,\ell'_{opt}) \le \mathrm{dist}(q,\ell_{opt}) + r \le \mathrm{dist}(q,\ell_p) + r \le \mathrm{dist}(q,\ell'_p) + 2r \tag{4}$$

Furthermore,

$$\begin{aligned}
\mathrm{dist}(q,\ell'_p) + \mathrm{dist}(q,\ell'_{opt}) &= \mathrm{dist}(q,u')\cos(\alpha_1/2) + \mathrm{dist}(q,t')\cos(\alpha_2/2) \\
&\ge \mathrm{dist}(q,u')/2 + \mathrm{dist}(q,t')/2 \quad \text{(since } \alpha_1,\alpha_2 \le 2\pi/3\text{)} \\
&\ge \mathrm{dist}(t',u')/2 \quad \text{(by triangle inequality)} \\
&\ge r_q \sin\alpha \ge R\sin\alpha \ge \frac{r}{\epsilon\delta}\delta \ge r/\epsilon \quad \text{(noting that } \ell'_p \text{ and } \ell'_{opt} \text{ are } \delta\text{-far)}
\end{aligned}$$

Therefore, using Equation 4 we can infer that $\mathrm{dist}(q,\ell'_p) + 2r + \mathrm{dist}(q,\ell'_p) \ge r/\epsilon$ and thus $\mathrm{dist}(q,\ell'_p) \ge r/(3\epsilon)$ for sufficiently small value of ϵ.

□

Claim A.4. $\mathrm{dist}(q,\ell'_{opt}) \ge \frac{r}{8\epsilon}$ *for sufficiently small value of* ϵ*.*

Proof. Suppose that, to the contrary, we have $\mathrm{dist}(q,\ell'_{opt}) < \frac{r}{8\epsilon}$. Then

$$\mathrm{dist}(q,t') = \mathrm{dist}(q,\ell'_{opt})/\cos(\alpha_2/2) \le 2\mathrm{dist}(q,\ell'_{opt}) \le \frac{r}{4\epsilon}$$

and thus, $\mathrm{dist}(q',s') \le \frac{r}{4\epsilon}$. However, this means that firstly, by triangle inequality and Observation A.1, and for sufficiently small ϵ

$$\mathrm{dist}(q',s) \le \mathrm{dist}(q',s') + \mathrm{dist}(s,s') \le \frac{r}{4\epsilon} + 2r \le \frac{3r}{8\epsilon}$$

and secondly

$$\mathrm{dist}(q',p') \ge \mathrm{dist}(p',s') - \mathrm{dist}(q',s') \ge R\sin\alpha - \frac{r}{4\epsilon} \ge \frac{3r}{4\epsilon} \quad \text{(since } \ell_p \text{ and } \ell_{opt} \text{ are } \delta\text{-far)} \Rightarrow$$

$$\text{dist}(q', p) \geq \text{dist}(q', p') - \text{dist}(p', p) \geq \frac{3r}{4\epsilon} - 2r \geq \frac{2r}{3\epsilon}$$

which is a contradiction because ANN have chosen p over s and therefore it must be true that $\text{dist}(q', p) \leq \text{dist}(q', s)(1+\epsilon)$, 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 ϵ. Therefore the claim holds. □

Using the above two claims, we have

$$\frac{\text{dist}(q, \ell_p)}{\text{dist}(q, \ell_{opt})} \leq \frac{\text{dist}(q, \ell'_p) + r}{\text{dist}(q, \ell'_{opt}) - r} \leq \frac{\text{dist}(q, \ell'_p)(1 + O(\epsilon))}{\text{dist}(q, \ell'_{opt})(1 - O(\epsilon))} \leq \frac{\text{dist}(q, \ell'_p)}{\text{dist}(q, \ell'_{opt})}(1 + O(\epsilon))$$

□

Lemma A.5. $\frac{\text{dist}(q,\ell'_p)}{\text{dist}(q,\ell'_{opt})} \leq \frac{\text{dist}(q,u')}{\text{dist}(q,t')}(1 + O(\epsilon))$

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 $\frac{\cos(\alpha_1/2)}{\cos(\alpha_2/2)} \leq (1 + O(\epsilon))$. As proved earlier in Equation 4

$$\begin{aligned}
&\text{dist}(q, \ell'_{opt}) \leq \text{dist}(q, \ell'_p) + 2r \Rightarrow \\
&r_q \sin\alpha_2 \leq r_q \sin\alpha_1 + 2r \Rightarrow \\
&\sin\alpha_2 - \sin\alpha_1 \leq 2\epsilon\delta \Rightarrow \\
&(\sin\alpha_2 - \sin\alpha_1)(\sin\alpha_2 + \sin\alpha_1) \leq 4\epsilon\delta \Rightarrow \\
&\cos^2\alpha_1 - \cos^2\alpha_2 \leq 4\epsilon\delta \Rightarrow \\
&(\cos\alpha_1 - \cos\alpha_2)(\cos\alpha_1 + \cos\alpha_2) \leq 4\epsilon\delta
\end{aligned}$$

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

$$\frac{\cos(\alpha_1/2)}{\cos(\alpha_2/2)} = \sqrt{\frac{1+\cos\alpha_1}{1+\cos\alpha_2}} \leq \frac{1+\cos\alpha_1}{1+\cos\alpha_2} \leq 1 + O(\epsilon)$$

Now since $\alpha_1 \leq \alpha_2 \leq \pi/2$, then $\cos\alpha_1 + \cos\alpha_2 \geq 0$. Therefore

$$\cos\alpha_1 - \cos\alpha_2 \leq \frac{4\epsilon\delta}{\cos\alpha_1 + \cos\alpha_2}$$

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

$$\begin{aligned}
&1 + \cos\alpha_1 \leq 1 + \cos\alpha_2 + 4\epsilon \Rightarrow \\
&\frac{1+\cos\alpha_1}{1+\cos\alpha_2} \leq 1 + \frac{4\epsilon}{1+\cos\alpha_2} \leq 1 + 4\epsilon
\end{aligned}$$

□

Lemma A.6. $\frac{\text{dist}(q',p')}{\text{dist}(q',s')} \leq (1+O(\epsilon))$

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 A.7. $\text{dist}(q',s) \geq \frac{r}{2\epsilon}(1-O(\epsilon))$

Proof.

$$\begin{aligned}\text{dist}(q',p)+\text{dist}(q',s) &\geq \text{dist}(q',p')-\text{dist}(p,p')+\text{dist}(q',s')-\text{dist}(s,s') \quad \text{(by triangle inequality)}\\ &\geq \text{dist}(p',s')-4r \quad \text{(by triangle inequality and Observation A.1)}\\ &\geq R\sin\alpha - 4r \geq r/\epsilon - 4r \geq \frac{r}{\epsilon}(1-O(\epsilon))\end{aligned}$$

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

$$\begin{aligned}&\text{dist}(q',s)(2+\epsilon) \geq \text{dist}(q',p)+\text{dist}(q',s) \geq \frac{r}{\epsilon}(1-O(\epsilon)) \quad \Rightarrow\\ &\text{dist}(q',s) \geq \frac{r}{2\epsilon(1+O(\epsilon))}(1-O(\epsilon)) \geq \frac{r}{2\epsilon}(1-O(\epsilon))\end{aligned}$$

□

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

$$\begin{aligned}\frac{\text{dist}(q',p)+2r}{\text{dist}(q',s)-2r} &\leq \frac{\text{dist}(q',s)(1+\epsilon)+2r}{\text{dist}(q',s)-2r} \quad \text{(Since } ANN \text{ has chosen } p \text{ over } s)\\ &\leq \frac{1+\epsilon+2r/\text{dist}(q',s)}{1-2r/\text{dist}(q',s)}\\ &\leq \frac{1+\epsilon+4\epsilon/(1-O(\epsilon))}{1-4\epsilon/(1-O(\epsilon))} \quad \text{(by Claim A.7)}\\ &\leq 1+O(\epsilon)\end{aligned}$$

□

B Time and Space Analysis

B.1 Proof of Lemma 4.1

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 ℓ 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 $\mathcal{S}(2n,\epsilon)$ (Remark 3.2), and the space used by the net module is at most $\mathcal{S}(n\frac{4Y_A/\epsilon^2}{Y_A\epsilon^3},\epsilon) = \mathcal{S}(4n/\epsilon^5,\epsilon)$ (Remark 3.4). Therefore the total space used is $O(n^4\mathcal{S}(4n/\epsilon^5,\epsilon))$. Moreover, as will be proved later in Lemma 6.4, the space needed to build the $APNLS$ data structure for each set is $O(n^7/\epsilon^3 \times \mathcal{S}(n,\epsilon))$ and since we call it for each line $\ell \in S$, the total space used at each level of recursion is $O(n^4\mathcal{S}(4n/\epsilon^5,\epsilon) + n^8/\epsilon^3 \times S(n,\epsilon))$. The recursion formula for the space at each level of recursion is

$$S(n) \leq S(n/2) + n^4\mathcal{S}(4n/\epsilon^5,\epsilon) + O(n^8/\epsilon^3 \times \mathcal{S}(n,\epsilon))$$

and therefore, the total space used by the algorithm is $O(N^4\mathcal{S}(4N/\epsilon^5,\epsilon)+N^8/\epsilon^3\times\mathcal{S}(N,\epsilon))$. Since $\mathcal{S}(N,\epsilon)$ is of the form $O(N+d)^{O(1/\epsilon^2)}$, therefore the lemma holds. □

B.2 Proof of Lemma 4.2

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 $O(dN^6 \log N + N^4 \mathcal{CT}(4N/\epsilon^5, \epsilon))$. Since the construction time of the $APNLS$ by Lemma 6.5 is $O(N^7/\epsilon^3 \times \mathcal{CT}(n, \epsilon))$, the total construction time of NLS is $O(N^4 \mathcal{CT}(4N/\epsilon^5, \epsilon) + N^8/\epsilon^3 \times \mathcal{CT}(N, \epsilon))$ which given that $\mathcal{CT}(N, \epsilon) = O(N + d)^{O(1/\epsilon^2)}$, it is of the form $(N + d)^{O(1/\epsilon^2)}$. □

B.3 Proof of Lemma 4.3

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 6.6 is at most

$$O(d \log^5 N + \log^4 N \times \mathcal{T}(4N/\epsilon^4, \epsilon))$$

Now, at a recursion level with n lines ($n = |S|$), if we have $n \leq \log^3 N$, the 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 3.4 takes $\mathcal{T}(4n/\epsilon^5, \epsilon)$ time. The recursion formula for the running time at each level of recursion is

$$T(n) \leq T(n/2) + \log n \times \mathcal{T}(4n/\epsilon^5, \epsilon), \quad T(\log^3 N) = d \log^3 N$$

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

$$d \log^3 N + \log^2 N \times \mathcal{T}(4N/\epsilon^5, \epsilon)$$

Therefore the total running time is

$$O(d \log^5 N + \log^4 N \times \mathcal{T}(4N/\epsilon^4, \epsilon) + \log^2 N \times \mathcal{T}(4N/\epsilon^5, \epsilon))$$

Since $\mathcal{T}(N, \epsilon)$ is of the form $(\log N + d + 1/\epsilon)^{O(1)}$, the lemma holds. □

B.4 Proof of Lemma 6.4

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 ℓ 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 δ_j. For each of them, the unbounded module uses space $\mathcal{S}(2n, \epsilon)$ (by Remark 3.2). Also for each such δ_j and each line ℓ', the number of hypeplanes $|G_{\ell',j}|$ is at most $\frac{4R_j}{R_j \epsilon^3} = O(1/\epsilon^3)$. 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 3.6) will be $O(n^7/\epsilon^3 \times \mathcal{S}(n, \epsilon))$. The recursion formula for the space at each level of recursion is

$$S(n) \leq S(n/2) + 4n^7/\epsilon^3 \times \mathcal{S}(n, \epsilon)$$

and therefore, the total space used by the algorithm is $O(|L'|^7/\epsilon^3 \times \mathcal{S}(|L'|, \epsilon))$. 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 $\mathcal{S}(|L'|, \epsilon)$ is of the form $O(|L'| + d)^{O(1/\epsilon^2)}$, the space of this data structure is of this form as well. □

B.5 Proof of Lemma 6.5

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

B.6 Proof of Lemma 6.6

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 3.4 takes $\frac{4R_0 n}{r\epsilon^2} = O(n/\epsilon^4)$. Therefore, the total time taken is $\log n \times \mathcal{T}(4n/\epsilon^4, \epsilon)$. The recursion formula for the running time at each level of recursion is

$$T(n) \leq T(n/2) + \log n \times \mathcal{T}(4n/\epsilon^4, \epsilon), \quad T(\log^3 N) = \log^3 N$$

and therefore since $\mathcal{T}(n/\epsilon^4, \epsilon)$ is poly-logarithmic in n and polynomial in $1/\epsilon$, the total running time of the algorithm is $O(d \log^3 N + \log^2 |L'| \times \mathcal{T}(4|L'|/\epsilon^4, \epsilon)) = O(d \log^3 N + \log^2 N \times \mathcal{T}(4N/\epsilon^4, \epsilon))$ which is poly-logarithmic in N and polynomial in $1/\epsilon$ and d. □