Approximate Line Nearest Neighbor in High Dimensions

Alexandr Andoni
MIT
andoni@mit.edu

Piotr Indyk
MIT
indyk@mit.edu

Robert Krauthgamer*
Weizmann Institute of Science
robert.krauthgamer@weizmann.ac.il

Huy L. Nguyen
MIT
hlnguyen@mit.edu

Abstract

We consider the problem of approximate nearest neighbors in high dimensions, when the queries are lines. In this problem, given n points in $\mathbb{R}^d$, we want to construct a data structure to support efficiently the following queries: given a line L, report the point p closest to L. This problem generalizes the more familiar nearest neighbor problem. From a practical perspective, lines, and low-dimensional flats in general, may model data under linear variation, such as physical objects under different lighting.

For approximation $1+\epsilon$, we achieve a query time of $d^3 n^{0.5+t}$, for arbitrary small $t>0$, with a space of $d^2 n^{O(1/\epsilon^2+1/t^2)}$. To the best of our knowledge, this is the first algorithm for this problem with polynomial space and sub-linear query time.

1 Introduction

Nearest neighbor problem is one of the fundamental geometric problems, with a wide range of applications. Given a set S of n points in a d-dimensional Euclidean space, the goal of the problem is to build a data structure which, given a query point, returns the data point closest to the query. Efficient algorithms for this problem exist if the data dimension is "low" [Cla88, Mei93]. In the high-dimensional case, efficient solutions exist as well, provided that the data structure is allowed to report an approximate nearest neighbor, whose distance to the query is at most some factor $c = 1+\epsilon > 1$ larger than the distance from the query to the actual nearest neighbor. In particular, several data structures with query time of $(d+\log n+1/\epsilon)^{O(1)}$ using $n^{(1/\epsilon)^{O(1)}}$ space are known [IM98, KOR98, HP01].

When the query is more complex than a point, however, the complexity of the problem is much less understood. Perhaps the simplest such case is when the query is allowed to be a k-dimensional flat, for some small value of k; we call this the *Approximate k-Flat Nearest Neighbor* problem. This is a natural generalization of the approximate nearest neighbor problem, where $k=0$. From a practical perspective, low-dimensional flats are often used to model data subject to linear variations. For example, one could capture the appearance of a physical object under different lighting or viewpoint [BHZ07].

It is therefore quite surprising that, for the case of high dimension d, there has been *no* prior work on this problem. For the "dual" version of the problem, where the query is a point but the data set consists of k-flats, two results are known [BHZ07, Mag07][1]. The first algorithm is essentially heuristic (although it allows some control of the quality of approximation). The second algorithm provides provable guarantees and very fast query time of $(d+\log n+1/\epsilon)^{O(1)}$. Unfortunately, the space requirement of the algorithm is super-polynomial, of the form $2^{(\log n)^{O(1)}}$.

In this paper we provide the first non-trivial algorithm for the approximate flat nearest neighbor problem, for the case when the query is a 1-flat, i.e., a line. Our data structure has query time sublinear in n and uses polynomial space. Thus, it demonstrates that the flat version of the approximate nearest neighbor problem supports efficient solutions, even in high dimensions.

Specifically, for $1+\epsilon$ approximation, we obtain a query time of $O(d^3 n^{0.5+t})$ with a space of $d^2 n^{O(1/\epsilon^2+1/t^2)}$ for any desired $t>0$.

1.1 Preliminaries and Definitions

A line ℓ in $\mathbb{R}^d$ is described as $\ell = \{a + \lambda \cdot u \mid \lambda \in \mathbb{R}\}$, where a is a vector in $\mathbb{R}^d$ and u is a unit-length vector in $\mathbb{R}^d$.

For a point $p \in \mathbb{R}^d$ and a line q in $\mathbb{R}^d$, we let $d(q,p)$ be the Euclidean distance from p to q, which is the distance from p to the projection of p onto q.

We use the term cylinder, with axis ℓ and radius R, to denote the set $\{p \mid d(p,\ell) \le R\}$. Any plane perpendicular to ℓ will be called base (or bottom) of

*Part of this work was done while at IBM Almaden. This research was supported in part by The Israel Science Foundation (grant #452/08).

[1] It is plausible that both algorithms could be adapted to our problem without sacrificing their complexity.

the cylinder. Furthermore, abusing notation, we will also use the notion of cylinder to denote the part of the (unbounded) cylinder contained between two hyperplanes perpendicular to ℓ, or, in other words, a set $\{p \mid d(p,\ell) \leq R \wedge x_1 \leq \mathrm{pr}_\ell(p) \leq x_2\}$ for some reals $x_1, x_2 \in \mathbb{R}$ where $\mathrm{pr}_\ell(p)$ is the projection of p into ℓ. In such case, we say the height of the cylinder is $x_2 - x_1$ (otherwise, the height is infinite).

For $c = 1+\epsilon$, we define the problem of *c-approximate line nearest neighbor (LNN)* as follows. Given a dataset $S \subset \mathbb{R}^d$ of n points, construct a data structure, which, given a query line q, reports a point $\tilde{p}$ such that $d(q,\tilde{p}) \leq c \cdot d(q,p)$ for all $p \in S$.

We use abbreviations LNN for approximate line nearest neighbor and NN for (standard) approximate nearest neighbor, instead of potentially more natural ALNN and ANN, to make the two acronyms easier to distinguish.

2 Approximate Line Nearest Neighbor

In this section, we describe our algorithm for solving the approximate line nearest neighbor. Our algorithm is mainly based on two data structures, treating two different cases. We call these data structures Cylinder-DS and Projection-DS , and we describe their implementation in later sections.

We first describe a high-level approach to the problem. For simplicity, suppose we are solving a simpler problem of approximate *near-neighbor* search[2] . In the latter problem, we are given a radius $r > 0$ in advance, and the algorithm has to find a point p at distance at most $(1+\epsilon)r$ from the query line q, *if* there exists a point p^* at distance at most r from q. For this problem, our algorithm is as follows. Let $t > 0$ be a small constant. During preprocessing, we repeatedly find (and remove from the data set) cylinders of radius rn^t that contain at least $m = \sqrt{n}$ points. We continue doing so until no such cylinder can be found anymore. For each of these cylinders we construct a "cylinder structure", Cylinder-DS . For the remaining points, we construct a "low-dimensional LNN structure", Projection-DS .

The query algorithm then queries each Cylinder-DS structure, as well as the Projection-DS structure. For each cylinder C_i, the algorithm tries to find a point $p \in C_i$ that is at distance at most $(1+\epsilon)r$ if a point at distance $\leq r$ exists. Each such query will take $\tilde{O}(n^t)$ time. Since there are at most $n/m = \sqrt{n}$ cylinders in total, we obtain a $\tilde{O}(n^{0.5+t})$ time bound for this step.

[2]It is known that if the query is a *point*, then the approximate near and nearest neighbor problems are equivalent up to polylogarithmic factors in the query time and space complexity [IM98, HP01]. Unfortunately, the reduction does not generalize to higher-dimensional queries.

The query algorithm also queries once the Projection-DS structure. This structure finds an (exact) r-near neighbor, but under the condition that there are at most $m = \sqrt{n}$ points at distance rn^t from the query line. Note that this condition holds since we removed all cylinders of radius rn^t that contain more than m points. The Projection-DS structure has query time of $m \cdot \mathrm{poly}(d, \log n) = \tilde{O}(\sqrt{n})$ and the space requirement is $n^{O(1/t^2)}$.

We now state precise conditions that we require of the structures Cylinder-DS and Projection-DS . We construct them in later sections.

Lemma 2.1. (Cylinder-DS) *There exists a data structure with the following guarantees. The input to the preprocessing stage is an m-point set $C \subset \mathbb{R}^d$, which is contained at distance at most R from an axis line ℓ. It is also given an approximation factor $1+\epsilon$ for $\epsilon > 0$. The data structure supports the following query:*

- *for a query line q, if the closest point $p^\star$ is at distance at least $R/2n^{2t}$, then the structure returns a point $p \in C$ at distance at most $(1+\epsilon)d(q,p^\star)$.*

Cylinder-DS *has space and preprocessing time bound of $\tilde{O}(d^2 m^{O(1/\epsilon^2)})$. The query time is $\tilde{O}(n^{2t}d^3)$ and achieves a constant success probability.*

Lemma 2.2. (Projection-DS) *There exists a data structure with the following guarantees. The input to the preprocessing stage is an n-point set $S \subset \mathbb{R}^d$ and an approximation factor $T = n^t/2$ for constant $t \in (0, 1/8]$. The data structure supports two types of queries:*

- *T-approximate line nearest neighbor: for a query line q, report a point $p \in S$ that is a T-approximate nearest neighbor;*
- *T-approximate line near neighbor reporting: for a query line q and threshold R, report a set $W \subset S$ that includes all points at distance R from q but does not include any point $p \in S$ at distance more than $T \cdot R$ from q.*

Projection-DS *has space and preprocessing time bound of $\tilde{O}(n^{O(1/t^2)} + dn)$. The query time is $d \log^{O(1)} n$ for the query of the first type and $|W| \cdot d \log^{O(1)} n$ for the query of the second type. Each query algorithm has a constant success probability.*

2.1 Full Algorithm Our actual algorithm differs only slightly from the above description. First, we find cylinders as above by considering only cylinders whose axis passes through pairs of points in the dataset S. As we will show later, it is enough to consider such cylinders only.

The second difference is that we need to solve near*est* neighbor problem and, thus, we do not have an estimate of r in advance. Instead, we use a PROJECTION-DS structure to find a n^t-approximate nearest neighbor $\tilde{p} \in S$. The distance from $\tilde{p}$ to the query line q, denoted $\tilde{r}$, is a very rough estimate for r. Additionally, we remove the dependence on r in the construction of cylinder structures. To accomplish this, we construct cylinders as follows: pick the thinnest cylinder that contains m points and remove it from the dataset; then pick the next thinnest cylinder and so forth. Finally, for each "suffix" of cylinders (thickest x cylinders), we construct a PROJECTION-DS structure on the points contained in these cylinders. During the query stage, we use the computed value $\tilde{r}$ to "zoom in" on the correct (prefix of) cylinders and the correct PROJECTION-DS structure containing the rest of the points.

The resulting preprocessing and query algorithms are described in Algorithms 1 and 2.

`Preprocessing(`S`):`

$U \leftarrow S$

for $i \in \{1, 2, \ldots n/m\}$ **do**

  For every two points p_1 and p_2 in U, consider the line ℓ_{p_1,p_2} going through p_1 and p_2, and let $F_{\ell_{p_1,p_2}}$ be the minimum radius of the cylinder with axis ℓ_{p_1,p_2} containing precisely m points

  Choose the line $\ell = \ell_{p_1,p_2}$ that minimizes $F_{\ell_{p_1,p_2}}$ and set $R_i = F_{\ell_{p_1,p_2}}$

  Remove from U the set A_i of all the points in U within distance R_i from ℓ

  Construct CYLINDER-DS structure C_i for the cylinder (ℓ, R_i) on the set A_i

end

For each $i \in \{0, 1, \ldots n/m\}$, construct a PROJECTION-DS structure L_i on points $\cup_{j=i+1}^{n/m} A_j$

Algorithm 1: Preprocessing algorithm. The input is a set $S \subset \mathbb{R}^d$ and an approximation ratio $1 + \epsilon$.

`Query-Processing(`q`):`

Query PROJECTION-DS structure L_0 to find an n^t-approximate nearest neighbor $\tilde{p}$

Let $\tilde{r}$ be the distance from q to $\tilde{p}$

Let i^* be the minimum $i \in [n/m]$ such that $R_i > \tilde{r}n^t$ or set $i^* = n/m + 1$ if no such i exists

for $i \in \{1, 2, \ldots, i^* - 1\}$ **do**

  Query CYLINDER-DS structure C_i

end

Query PROJECTION-DS structure L_{i^*-1} to find all $\frac{n^t}{2}$-approximate line near neighbors for threshold $\tilde{r}$

Output the point nearest to q from the ones found above

Algorithm 2: Query algorithm. The input is a line q and the output is a point p that is a $(1+\epsilon)$-approximate line nearest neighbor.

2.2 Correctness and Run-time Analysis

Correctness. We argue that the algorithm returns a $(1 + \epsilon)$-approximate nearest neighbor. Let $p^\star$ be the closest point to q, and let $r^\star = d(q, p^\star)$. Then, by the guarantee of PROJECTION-DS , we conclude that $\tilde{r}$ such that $\tilde{r}/n^t \leq r^\star \leq \tilde{r}$.

Suppose $p^\star$ is in a cylinder C_i with $i \leq i^* - 1$. Then $R_i \leq \tilde{r} \cdot n^t$ and thus $r^\star \geq \tilde{r}/n^t \geq R_i/n^{2t}$. In this case the structure CYLINDER-DS C_i guarantees to return a c-approximate nearest neighbor.

Now suppose $p^\star$ belongs to a cylinder C_i for $i \geq i^*$. Then the structure PROJECTION-DS L_{i^*} is guaranteed to report the exact nearest neighbor $p^\star$.

Runtime. First we prove the following lemma.

LEMMA 2.3. *If there is a line ℓ within distance R from a set A of m points then there is a line through two of the points that is within distance $2R$ from all m points.*

Proof. Consider the cylinder of radius R with axis at ℓ, and, w.l.o.g., assume ℓ coincides with the x-axis. Take the point $a \in A$ with the smallest x-value and the point $b \in A$ with the largest x-value. Let line ℓ' be the line passing through a and b. We claim that any point $c \in A$ is within distance $2R$ from ℓ'. Consider the point z obtained through the intersection of the line ℓ' and the plane through c perpendicular to x-axis. This point z is inside the cylinder, and thus, $d(c, \ell') \leq d(c, z) \leq d(c, \ell) + d(z, \ell) \leq 2R$. ■

The first query to PROJECTION-DS structure, and the queries to CYLINDER-DS structure, each run in time $\tilde{O}(n^{2t})$. Since there are at most $n/m = \sqrt{n}$ calls to CYLINDER-DS structure, these calls take $\tilde{O}(d^3 n^{.5+2t})$ time. It remains to argue that the last call to PROJECTION-DS takes $\tilde{O}(dn^{.5+2t})$ time. Indeed, consider the structure PROJECTION-DS L_{i^*}. By the construction of the cylinder C_{i^*} and above lemma, we know that there is no line ℓ that contains more than m points from L_{i^*} within distance $R_{i^*}/2 \geq \tilde{r} \cdot n^t/2$. The guarantee of the PROJECTION-DS structure for $\frac{n^t}{2}$-approximate reporting then says that the run-time of the query is at most $\tilde{O}(dm) = \tilde{O}(d\sqrt{n})$.

We note that the space is bounded by the space used by at most $O(\sqrt{n})$ CYLINDER-DS and PROJECTION-DS structures, giving a total of $\tilde{O}(d^2 n^{O(1/\epsilon^2+1/t^2)})$ space.

3 Data Structure Cylinder-DS for Cylinder Case

In this section, we consider the problem of finding a c-approximate line nearest neighbor of a query line q among m points inside a cylinder C_1 of radius R with axis ℓ assuming that the nearest neighbor $p^\star$ is at least $\frac{R}{2n^{2t}}$ away from q. We first describe the preprocessing and query answering algorithms in Section 3.1. We then prove their correctness and analyze the query time and space usage in Section 3.2. Finally, we show how the resulting algorithm yields Lemma 2.1.

3.1 The Algorithm In this section, we describe the algorithm for the cylinder case. At a high level, the algorithm works as follows. It first computes the intersection of the query line q and a bigger cylinder containing C_1 and then extends the intersection region so as to include all c-LNNs in the extended region. This extended region is then divided into $\tilde{O}(n^{2t})$ smaller regions so that the part of q in each region is "small" enough to be approximated by a single point with a small additive approximation error. For each smaller region, the algorithm queries a standard data structure for finding approximate nearest neighbors (NN) of points, to find a c-approximate nearest neighbor within that region. Finally, it reports the point closest to q among the ones it finds. We choose to use a data structure for finding approximate nearest neighbor of query points with an efficient query time of $O(d^2(\log m + \frac{1}{\epsilon})^{O(1)})$ using $d^2 m^{O(1/\epsilon^2)}$ space (e.g., [KOR98]).

There are two additional technical problems that the algorithm needs to take care of. First, it is not known at preprocessing time where the intersection between q and the cylinder is. This problem can be rectified as follows. Assume that the points are sorted by their projections on the cylinder axis and labeled $1, 2, \ldots, m$ accordingly. A *dyadic interval* is an interval of the form $[i2^l + 1, (i+1)2^l]$ for some i, l satisfying $0 \le l \le \log m, 0 \le i \le \lceil m/2^l \rceil - 1$. It is well-known that any interval can be divided into $O(\log m)$ dyadic intervals. At preprocessing time, for each dyadic interval, we build a separate data structure for points whose labels constitute that interval. At query time, the algorithm can just query $O(\log m)$ of them to get the answer. Second, there needs to be a special treatment when the nearest neighbor is very far from the query line. It turns out that this case can be reduced to finding the nearest neighbor in one-dimensional space, which can be solved easily.

See algorithms 3 and 4, as well as figures 1 and 2, for a detailed description.

3.2 Correctness and Running Time Analysis When q is parallel to ℓ, the distance between a point and q is exactly the same as the distance between their projections on the subspace orthogonal to ℓ so the correctness follows from the correctness of the NN data structure. For the rest of the section, we consider the case when q is not parallel to ℓ.

Let $r^\star$ be the distance between q and the nearest point $p^\star \in A$. Recall that we are only finding a c-LNN when $r^\star > \frac{R}{2n^{2t}}$. There are two cases of $r^\star$ we need to deal with. In section 3.2.1, we solve the problem when $r^\star$ is "small". In section 3.2.2, we solve the problem when $r^\star$ is "large". Finally, we give a concluding remark for the cylinder case in section 3.3.

3.2.1 Small $r^\star$ In this section, we consider the case $r^\star < \frac{3R}{\epsilon}$.

Lemma 3.1. *Compute E as in algorithm 4. We claim that all c-LNNs of q are contained in E.*

Proof. We prove by showing the contra-positive. Let p be a point in C_1 but outside of E. The distance between p and the part of q outside of C_2 (i.e., set $q \setminus C_2$) is at least $\frac{7R}{\epsilon} - R > \frac{3R(1+\epsilon)}{\epsilon} > r^\star(1+\epsilon)$ for all $\epsilon < 1$. The distance between p and the part of q inside C_2 is at least $\frac{3R(1+\epsilon)}{\epsilon} > r^\star(1+\epsilon)$ because E was extended on both ends by $\frac{3R(1+\epsilon)}{\epsilon}$. Thus, p cannot be a c-LNN of q. ∎

We distinguish two cases for the height of E: E can be either long or short compared to the cylinder radius. Firstly, consider the case E is short i.e. the height of E is $M \le \frac{252R\log n}{\epsilon}$. We start by showing that the distance being used in steps 14 and 15 of algorithm 4 is a good approximation to the true distance. See Figure 1 for reference. Let G be defined as in step 14.

Lemma 3.2. *Let ϵ be a constant smaller than 1. We claim that when $M \le \frac{252R\log n}{\epsilon}$, a $(1+\epsilon-\frac{1}{\log n})$-NN of an end point of some sub-segment from G is a $(1+\epsilon)$-LNN of q.*

Proof. The radius of C_2 is $\frac{7R}{\epsilon}$ so the segment of q limited by two hyperplanes containing the two faces of E has length at most $O(\frac{R(\log n + d)}{\epsilon})$. This segment of q is divided into $\Theta(\frac{dn^{2t}\log^2 n}{\epsilon})$ sub-segments so each sub-segment has length at most $\frac{R}{6n^{2t}\log n}$. Therefore, using an end point as an approximation for any point on the same sub-segment only distorts distance by an additive term of at most $\frac{R}{6n^{2t}\log n}$. Since $r^\star > \frac{R}{2n^{2t}}$, the distance between any point p in S and the closest sub-segment end point is a $(1+\frac{1}{3\log n})$-approximation of the distance

`CylinderDS-Preprocessing`(A):

Sort the points in A by their projections on ℓ and label them $1, 2, \ldots, m$ accordingly

for $D \in$ *dyadic intervals* **do**

  Consider the section of C_1 that contains the points whose labels constitute D, and partition D into $\frac{252n^{2t}\log n}{\epsilon}$ sub-cylinders of equal heights

  For each sub-cylinder, project all the points in it onto the sub-cylinder's bottom face and build a data structure for finding NN of query points among the projections onto the bottom face

end

Build a standard NN data structure, denoted NN_{all}, on all points in $A \subset \mathbb{R}^d$

Project the set A onto the subspace orthogonal to ℓ, and build a standard NN data structure, denoted $NN_{orthogonal}$, on the resulting points (in a $(d-1)$-dimensional space)

Algorithm 3: Algorithm for building CYLINDER-DS . Input is a set of points A.

`CylinderDS-Query-Processing`(q):

1 **if** *q and ℓ are parallel* **then**

2   Query $NN_{orthogonal}$ to find a $(1+\epsilon-\frac{1}{\log n})$-NN of the projection of q (a point) in the subspace orthogonal to ℓ

3 **else**

4   Let C_2 be an infinite cylinder with the same axis ℓ as C_1 and with radius $\frac{7R}{\epsilon}$. Compute the section of C_2 containing the intersection of q and C_2. Extend that section by $\frac{3R(1+\epsilon)}{\epsilon}$ on both ends and call the resulting extended cylinder E. Let M be the height of cylinder E

5   **if** $M > \frac{252R\log n}{\epsilon}$ **then**

6   Use binary search to find the interval I of labels of points inside E

7   Decompose I into at most $O(\log m)$ dyadic intervals

8   **for** $D \in$ *above dyadic intervals* **do**

9   **for** $C \in$ *sub-cylinders of the cylinder containing the points whose labels constitute D* **do**

10   Query the NN structure of the bottom face of C to find a $(1+\epsilon-\frac{1}{\log n})$-NN of the intersection of q and the bottom face.

11   **end**

12   **end**

13   **else**

14   Divide the segment of q inside E into $\Theta(\frac{dn^{2t}\log^2 n}{\epsilon})$ sub-segments of equal length, and let G be the set of the resulting sub-segments

15   Query NN_{all} to find a $(1+\epsilon-\frac{1}{\log n})$-NN of each end point of each segment in G

16   **end**

17   Find the point p_1 on ℓ that is closest to q. Use binary search to find the point in A whose projection on ℓ is closest to p_1

18   Return the point nearest to q among the points found above

19 **end**

Algorithm 4: Query algorithm for CYLINDER-DS of a cylinder C_1 with axis ℓ and radius R. The input is a query line q and the output is a c-LNN of q among the points inside C_1.

between p and q. Therefore, a $(1+\epsilon-\frac{1}{\log n})$-NN of the end point closest to $p^\star$ is a $(1+\epsilon)$-LNN of q. ∎

We can now show that steps 14 and 15 of algorithm 4 finds a c-LNN efficiently when $M \le \frac{252R\log n}{\epsilon}$.

LEMMA 3.3. *When $M \le \frac{252R\log n}{\epsilon}$, a c-LNN of q can be found in $\tilde{O}(d^3n^{2t})$ time.*

Proof. By lemma 3.2, one can find a c-LNN of q by finding a $(1+\epsilon-\frac{1}{\log n})$-NN of each end point of sub-segments of the intersection of q and E and returning the point closest to q. Finding each $(1+\epsilon-\frac{1}{\log n})$-NN requires one query to the NN_{all} data structure, which

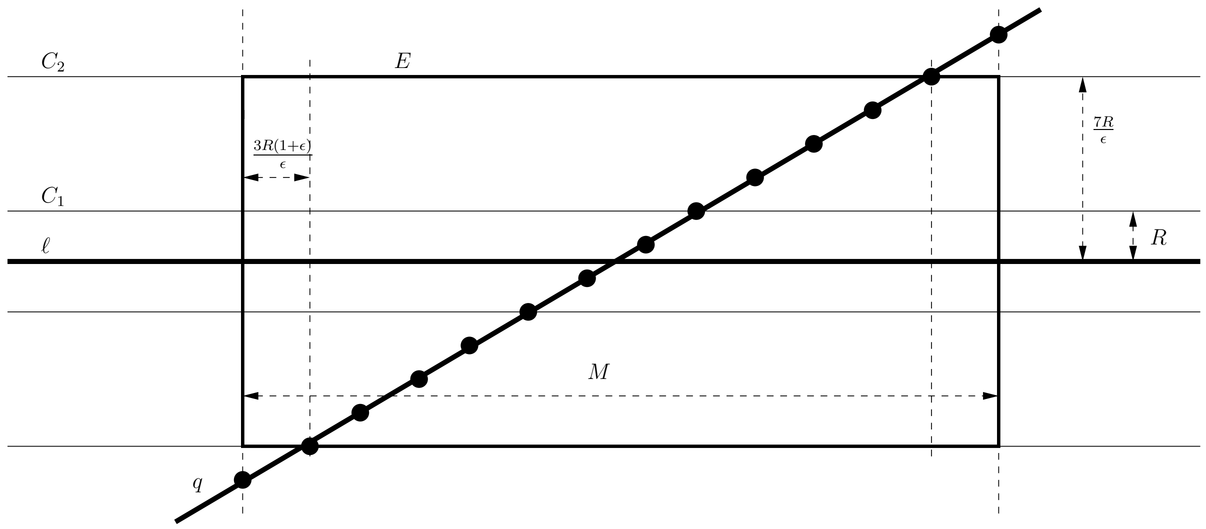

Figure 1: Query algorithm for CYLINDER-DS when $r^\star < \frac{3R}{\epsilon}$ and $M \le \frac{252R\log n}{\epsilon}$. C_1 is the unbounded cylinder with axis ℓ and radius R. C_2 is the unbounded cylinder of radius $\frac{7R}{\epsilon}$. E is a cylinder of radius $\frac{7R}{\epsilon}$ and height M. The dots on the query line q are the endpoints of the segments in G for which we query the NN_{all} data structure.

takes $O(d^2(\log n + \frac{1}{\epsilon})^{O(1)})$ time. There are $\tilde{O}(dn^{2t})$ end points so the total running time is $\tilde{O}(d^3n^{2t})$. ∎

Now consider the case E is long i.e. $M > \frac{252R\log n}{\epsilon}$. We first show the distance being used in step 10 of algorithm 4 is a good approximation of the true distance. See Figure 2 for reference.

LEMMA 3.4. *Consider the case $M > \frac{252R\log n}{\epsilon}$. As in algorithm 4, we project the points in S and the segment of q inside a sub-cylinder onto the sub-cylinder's bottom face. We claim that the distance between a point p and an end point of the segment of q in the projection space is a $(1 + \frac{1}{3\log n})$-approximation of the distance between p and q in the original space.*

Proof. Let P be the hyperplane containing p and orthogonal to ℓ. Let v be the vector from p to the intersection of hyperplane P and line q. Decompose v into $v^\|$ and $v^\perp$, where $v^\|$ is the component parallel to q and $v^\perp$ is perpendicular to q. Note that $v^\perp$ is the distance between p and q. Let u and i be the unit vectors on ℓ and q, respectively. Decompose u into $u^\|$ and $u^\perp$ in the same way as v. Let α be the angle between u and i, where $0 \le \alpha \le \frac{\pi}{2}$. By the definition of α, we have $u^\| = \cos\alpha \cdot i$, and $||u^\perp||_2 = \sin\alpha$. Because v is orthogonal to ℓ, we have $0 = u \cdot v = u^\| \cdot v^\| + u^\perp \cdot v^\perp$. By the Cauchy-Schwarz inequality, $|u^\perp \cdot v^\perp| \le ||u^\perp||_2 ||v^\perp||_2 = \sin\alpha ||v^\perp||_2$. Therefore, $\cos\alpha ||v^\||_2 = |u^\| \cdot v^\|| \le \sin\alpha ||v^\perp||_2$. In other words, $\frac{||v^\||_2}{||v^\perp||_2} \le \tan\alpha$.

Since $M > \frac{252R\log n}{\epsilon}$ and the diameter of the cylinder C_2 is $\frac{14R}{\epsilon}$, we have that $\tan\alpha < \frac{1}{9\log n}$. Thus, $\frac{||v^\||_2}{||v^\perp||_2} < \frac{1}{9\log n}$ and $||v||_2$ is a $1 \pm \frac{1}{9\log n}$-approximation of $||v^\perp||_2$. The length of the segment of q in the projection space is at most $\frac{1}{\frac{252n^{2t}\log n}{\epsilon}} \cdot \frac{14R}{\epsilon} = \frac{R}{18n^{2t}\log n} < \frac{r^\star}{9\log n}$. Therefore, using the projection of an end point of the segment of q as an approximation for the projection of the intersection of q and hyperplane P further distorts distance by at most a factor of $1 + \frac{1}{9\log n}$. Combining the two approximation steps, we get the desired claim. ∎

We can now show that step 10 of algorithm 4 can find a c-LNN of q efficiently when $M > \frac{252R\log n}{\epsilon}$.

LEMMA 3.5. *When $M > \frac{252R\log n}{\epsilon}$, a c-LNN of q can be found in $\tilde{O}(d^2n^{2t})$ time.*

Proof. By lemma 3.4, one can find a c-LNN of q inside each sub-cylinder by finding a $(1 + \epsilon - \frac{1}{\log n})$-NN of the projection of q on the bottom face of the sub-cylinder. This can be done by making one query to the NN data structure, which takes $O(d^2(\log n + \frac{1}{\epsilon})^{O(1)})$ time. There are $\tilde{O}(n^{2t})$ sub-cylinders so the total running time is $\tilde{O}(d^2n^{2t})$. ∎

3.2.2 Large $r^\star$

In this section, we consider the case $r^\star \ge \frac{3R}{\epsilon}$. We will show that in this case, step 17 of algorithm 4 finds a c-LNN of q efficiently.

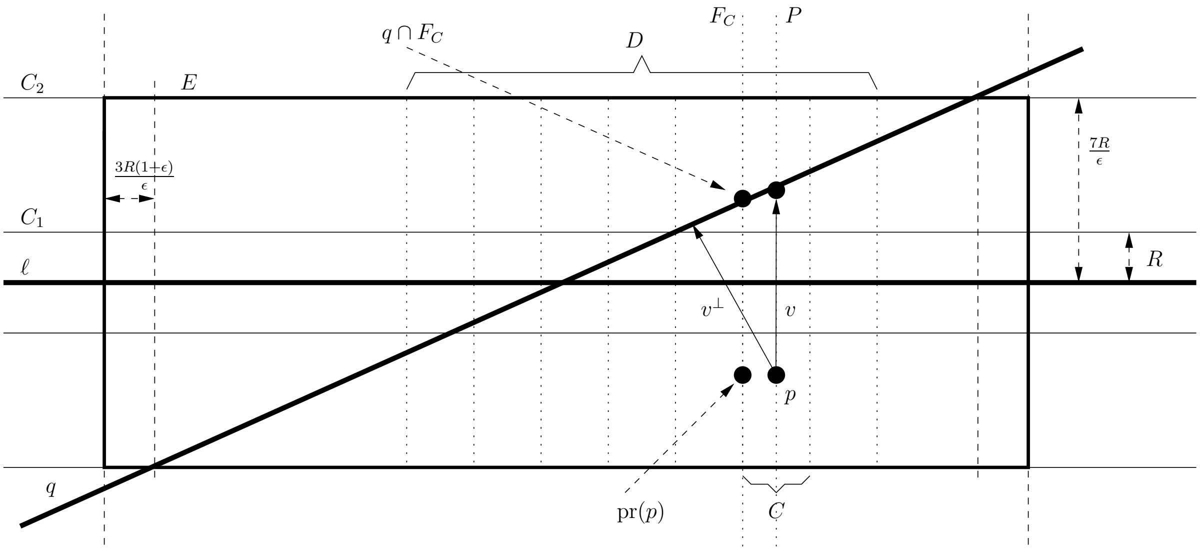

Figure 2: Query algorithm for CYLINDER-DS when $r^\star < \frac{3R}{\epsilon}$ and $M > \frac{252R\log n}{\epsilon}$. While we partition E into up to $O(\log m)$ dyadic intervals, we illustrate just one such cylinder, D. The cylinder D is further partitioned in equal height cylinders. For a particular such sub-cylinder, the algorithm projects points and the line to the plane P.

LEMMA 3.6. *If $r^\star \geq \frac{3R}{\epsilon}$, a c-LNN of q can be found in $O(\log n)$ time.*

Proof. The distance between a point and the cylinder axis is at most R so replacing a point by its projection on ℓ only distorts distance by at most a factor of $1 + \frac{\epsilon}{3}$. Assume that the parametric equation of ℓ and q are $u_\ell + t_\ell v_\ell$ and $u_q + t_q v_q$, respectively, where $t_\ell, t_q \in \mathbb{R}$ and u_ℓ, v_ℓ, u_q, v_q are vectors in $\mathbb{R}^d$. For any value of t_ℓ, we can find the value of t_q corresponding to the point on q that is nearest to the point $u_\ell + t_\ell v_\ell$. The solution is $t_q = \frac{v_q \cdot (u_\ell + t_\ell v_\ell - u_q)}{||v_q||^2}$, which is linear in t_ℓ. Thus, the square of the distance between $u_\ell + t_\ell v_\ell$ and q is a quadratic function in t_ℓ and the minimum is achieved at some $t^\star_\ell$. Therefore, we can find the nearest neighbor of q among points on ℓ by finding the point closest to the point $u_\ell + t^\star_\ell v_\ell$. This can be done by a simple binary search, as in step 17. ■

3.3 Summary of the data structure for CYLINDER-DS

Combining two cases, we have an algorithm for $r^\star > \frac{R}{2n^{2t}}$. We can now finalize the proof of Lemma 2.1.

Proof. [Proof of Lemma 2.1] This lemma follows directly from combining Lemmas 3.3, 3.5, and 3.6. The running time is dominated by the case $\frac{R}{2n^{2t}} < r^\star < \frac{3R}{\epsilon}$ so the total running time is $\tilde{O}(d^3 n^{2t})$. As described in algorithm 3, we have $\tilde{O}(n^{2t})$ NN data structures so the total space is $O(d^2 n^{2t+O(1/\epsilon^2)})$. ■

4 Data Structure PROJECTION-DS

In this section, we describe the structure PROJECTION-DS , proving Lemma 2.2. First we give the algorithm and then prove its correctness.

The outline of the data structure is as follows. First we perform a random projection of the set S into a space of constant dimension, namely dimension $k = 2/t + 1$. Then, we solve the *exact* version of line nearest neighbor problem in the obtained k-dimensional space. For the second step, we follow the approach given by Magen in [Mag07, Section 5].

For completeness, we describe Magen's approach for solving low-dimensional (exact) line nearest neighbor problem. (We note that Magen's approach addresses the dual version of this sub-problem, but it adapts to our case nonetheless.) The idea is to reduce the problem to a ray-shooting problem in some dimension $k' = O(k^2)$. Namely, we transform each point $p \in S$ into a hyperplane $H_p \subset \mathbb{R}^{k'}$ such that the first $k' - 1$ coordinates encode "all possible" lines and the last coordinate encodes the distance-squared from the corresponding line to the point p. The hyperplane H_p is obtained as follows. Fix the point p and consider a line $\ell = \{a + u\lambda \mid \lambda \in \mathbb{R}\}$ for some vector $a \in \mathbb{R}^k$ and unit-length vector $u \in \mathbb{R}^k$. Then, the distance-squared from p to ℓ is equal to

$$\begin{aligned} d^2(p,\ell) &= \sum_i (a_i - p_i + u_i \sum_j (p_j - a_j)u_j)^2 \\ &= \left(\sum_j (p_j - a_j)u_j\right)^2 + \sum_i (a_i - p_i)^2 + \\ &\quad +2\sum_i (a_i - p_i)u_i \sum_j (p_j - a_j)u_j \end{aligned}$$

which is a degree-6 polynomial in variables a_i, u_i, $i = 1, 2, \ldots k$, with $O(k^2)$ monomials. We can linearize this polynomial by assigning a variable to each monomial, and thus view $d^2(p,\ell)$ as a linear function f_p of $O(k^2)$ monomials (of degree at most 6) of u_i, a_i's. The hyperplane H_p is simply defined by the equation $f_p(x_1, \ldots x_{k'-1}) = x_{k'}$. Now to find the closest point p to a query line ℓ (given by $a, u \in \mathbb{R}^k$), we first compute the corresponding monomials to obtain a point $q_\ell \in \mathbb{R}^{k'-1}$, and then shoot a ray from $(q_\ell, -\infty)$ towards $(q_\ell, +\infty)$. Then the first intersected hyperplane H_p gives the closest point p to ℓ. Similarly, to obtain all R-near neighbors of ℓ, we report all hyperplanes intersected by the semi-line from $(q_\ell, -\infty)$ to (q_ℓ, R^2). To solve the ray-shooting problem in $\mathbb{R}^{k'}$ itself, we use a data structure of Meiser [Mei93] that gives $O(n^{k'+1}) = O(n^{O(1/t^2)})$ space and $\mathrm{poly}(k, \log n)$ query time.

It remains to argue that the initial random projection step preserves the distances sufficiently well. We note that we do a dimensionality reduction into a very low dimensional space, and thus we will obtain some high distortions. Nonetheless, we show that this distortion is still bounded by $T = n^t/2$, a factor polynomial in n.

The main part of the analysis for this case is thus the following lemma. We note that Magen [Mag07] proves a similar lemma for small approximations, $c = 1 + \epsilon$.

Lemma 4.1. *Consider a point p and a line l in R^d. For $k \geq 5$, let $P \in M_{k,d}(\mathbb{R})$ be a random projection from $\mathbb{R}^d$ to $\mathbb{R}^k$, scaled by $\frac{1}{2}\sqrt{\frac{d}{k}}$. Let $p' = Pp$ be the projection of p and line $l' \subset \mathbb{R}^k$ is the projection of l under P. Let Δ be the distance between p and q and Δ' be the distance between p' and l'. Then, for any $c > 10$, we have that:*

- $\Pr[\Delta' \leq \Delta] \geq 1 - e^{-k/2}$, *and*
- $\Pr[\Delta/c \leq \Delta'] \geq 1 - (c/15)^{-k/2}$.

Before proving the lemma, we complete the correctness analysis of the algorithm. Indeed, the above lemma implies that:

- if p^* is the distance to nearest point to q, then the distance to p^* does not increase under the projection with constant probability;
- all points p at distance $> TR$ from the line q remain at distance at least R in the projected space since each one remains at distance at least R with probability $\geq 1 - (n^t/15)^{-k/2} \geq 1 - n^{-1-t/4}$.

The above allows us to conclude Lemma 2.2. It only remains to prove Lemma 4.1.

Proof. [Proof of Lemma 4.1] To simplify the proof, we assume here that the random projection is scaled by $\sqrt{d/k}$ (without the 1/2 factor). Then we need to prove that $\Pr[\Delta' \leq 2\Delta] \geq 1 - e^{-k/2}$ and $\Pr[\Delta' \geq \Delta/c] \geq 1 - (c/7.5)^{-k/2}$.

We use standard Johnson-Lindenstrauss Lemma to prove our claim. We note that we use JL lemma for high-distortion regime, as opposed to the more usual $1 + \epsilon$ approximation regime. We use the following form of JL lemma (see, e.g., [DG99] or [IM98]):

Lemma 4.2. (JL Lemma) *Let $P \in M_{k,d}$ be a random projection matrix, scaled by $\sqrt{d/k}$. Then, for every unit-length vector x, and every scalar $\beta \geq 2$, we have that:*

- $\Pr[\|Px\|_2 \geq \beta] \leq O(k) \cdot \exp[-k/2 \cdot (\beta - (1 + \ln \beta))]$ *and*
- $\Pr[\|Px\|_2 \leq 1/\beta] \leq O(k) \cdot \exp[-k/2 \cdot (\ln \beta - 1)]$.

We prove that $\Delta' \leq 2\Delta$ with the desired probability. Let $\hat{p}$ be the projection of p onto l (in $\mathbb{R}^d$), and let $\hat{p}' = P\hat{p}$ be the projection of $\hat{p}$ under P. Then, by the above lemma, $d(p', l') \leq d(p', \hat{p}') \leq 2d(p, l)$ with probability at least $1 - e^{-\Omega(k)}$.

Now we prove that $\Delta'/c \leq \Delta$ with the desired probability. By rescaling, we can assume, wlog, that $d(p, l) = 1$. Take $q \in l$ to be a point on l at distance $\frac{1}{c^2}$ from $\hat{p}$. Also, let $q' = Pq$. Note that q' and $\hat{p}'$ completely define l' in $\mathbb{R}^k$. Finally, let $u = q - \hat{p}$ and $u' = Pu = q' - \hat{p}'$. Then, any point on l' is equal to $\hat{p}' + \lambda u'$ for some real λ. Note that $\|u'\| \leq 1/c$ with probability at least $1 - e^{-k/2(c-1-\ln c)} \geq 1 - e^{-kc/4}$.

The proof plan is to first consider a (infinite) number of dense, equally spaced points on the line l' and prove that none is close to p'. Then we prove that p' cannot be close to a point in between the considered points on the line l'.

In particular, for $i \in \mathbb{Z}$, let $q_i = \hat{p} + iu$ and $q_i' = Pq_i = p' + iu'$. By JL lemma from above, we have that $d(p', q_0') = d(\hat{p}', p') \geq 1/c$ with probability at least $1 - (e/c)^{k/2}$. For $i \neq 0$, we have that $d(p, q_i) \geq i$, and thus $d(p', q_i') \geq 1/c$ with probability at least

$1-(\frac{e}{c|i|})^{k/2}$. By union bound over all points q'_i for $i \in \mathbb{Z}$, we have that $d(p', q'_i) \geq 2/c$ with probability at least $1-(e/c)^{k/2} \cdot (1+\sum_{i\in\mathbb{Z}} 1/|i|^{k/2}) \geq 1-5(e/c)^{k/2}$.

Now suppose none of the "bad" (low-probability) events from above holds. Then for any point $w \in l'$, we have that $d(p', w) \geq d(p', q'_i) - d(q'_i, w)$, where i is such that q'_i is the closest to w. In such case, we have that $d(p', w) \geq 2/c - 1/c = 1/c$. This concludes the proof of Lemma 4.1. ■

References

[BHZ07] Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. Approximate nearest subspace search with applications to pattern recognition. In *Computer Vision and Pattern Recognition (CPRV'07)*, pages 1–8, June 2007.

[Cla88] K. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17:830–847, 1988.

[DG99] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstrauss lemma. *ICSI technical report TR-99-006, Berkeley, CA*, 1999.

[HP01] S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Proceedings of the Symposium on Foundations of Computer Science*, 2001.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, pages 604–613, 1998.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *Proceedings of the Symposium on Theory of Computing*, pages 614–623, 1998.

[Mag07] A. Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Discrete and Computational Geometry*, 38(1):139–153, July 2007. Previously in RANDOM'02.

[Mei93] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106:286–303, 1993.