Approximate Nearest Line Search in High Dimensions

Sepideh Mahabadi

Massachusetts Institute of Technology
The NLS Problem

- Given: a set of $N$ lines $L$ in $\mathbb{R}^d$
The NLS Problem

- Given: a set of $N$ lines $L$ in $\mathbb{R}^d$
- Goal: build a data structure s.t.
  - given a query $q$, find the closest line $\ell^*$ to $q$
The NLS Problem

• Given: a set of $N$ lines $L$ in $\mathbb{R}^d$
• Goal: build a data structure s.t.
  – given a query $q$, find the closest line $\ell^*$ to $q$
  – polynomial space
  – sub-linear query time
The NLS Problem

- Given: a set of $N$ lines $L$ in $\mathbb{R}^d$
- Goal: build a data structure s.t.
  - given a query $q$, find the closest line $\ell^*$ to $q$
  - polynomial space
  - sub-linear query time

Approximation

- Finds an approximate closest line $\ell$
  \[ \text{dist}(q, \ell) \leq \text{dist}(q, \ell^*)(1 + \epsilon) \]
Nearest Neighbor Problems
Motivation
Previous Work
Our result
Notation

BACKGROUND
Nearest Neighbor Problem

NN: Given a set of $N$ points $P$, build a data structure s.t. given a query point $q$, finds the closest point $p^*$ to $q$. 
Nearest Neighbor Problem

NN: Given a set of $N$ points $P$, build a data structure s.t. given a query point $q$, finds the closest point $p^*$ to $q$.

- Applications: database, information retrieval, pattern recognition, computer vision
  - Features: dimensions
  - Objects: points
  - Similarity: distance between points
Nearest Neighbor Problem

NN: Given a set of \( N \) points \( P \), build a data structure s.t. given a query point \( q \), finds the closest point \( p^* \) to \( q \).

- Applications: database, information retrieval, pattern recognition, computer vision
  - Features: dimensions
  - Objects: points
  - Similarity: distance between points
- Current solutions suffer from “curse of dimensionality”:
  - Either space or query time is \text{exponential} in \( d \)
  - Little improvement over linear search
Approximate Nearest Neighbor (ANN)

- ANN: Given a set of $N$ points $P$, build a data structure s.t. given a query point $q$, finds an approximate closest point $p$ to $q$, i.e.,
  \[ \text{dist}(q, p) \leq \text{dist}(q, p^*)(1 + \epsilon) \]
Approximate Nearest Neighbor (ANN)

- ANN: Given a set of $N$ points $P$, build a data structure s.t. given a query point $q$, finds an approximate closest point $p$ to $q$, i.e.,
  \[ \text{dist}(q, p) \leq \text{dist}(q, p^*) (1 + \epsilon) \]

- There exist data structures with different tradeoffs. Example:
  - Space: $(dN)^O\left(\frac{1}{\epsilon^2}\right)$
  - Query time: $\left(\frac{d \log N}{\epsilon}\right)^O(1)$
Motivation for NLS

One of the simplest generalizations of ANN: data items are represented by $k$-flats (affine subspace) instead of points
Motivation for NLS

One of the simplest generalizations of ANN: data items are represented by $k$-flats (affine subspace) instead of points

- Model data under linear variations
- Unknown or unimportant parameters in database
Motivation for NLS

One of the simplest generalizations of ANN: data items are represented by $k$-flats (affine subspace) instead of points

- Model data under linear variations
- Unknown or unimportant parameters in database

- Example:
  - Varying light gain parameter of images
  - Each image/point becomes a line
  - Search for the closest line to the query image
Previous and Related Work

- Magen[02]: Nearest Subspace Search
  - Query time is fast: \( (d + \log N + \frac{1}{\epsilon})^{O(1)} \)
  - Space is super-polynomial: \( 2^{(\log N)^{O(1)}} \)
Previous and Related Work

• Magen[02]: Nearest Subspace Search
  – Query time is fast: \( (d + \log N + \frac{1}{\epsilon})^{O(1)} \)
  – Space is super-polynomial: \( 2^{(\log N)^{O(1)}} \)

Dual Problem: Database is a set of points, query is a \( k \)-flat

• [AIKN] for 1-flat: for any \( t > 0 \)
  – Query time: \( O(d^3 N^{0.5+t}) \)
  – Space: \( d^2 N^{O\left(\frac{1}{\epsilon^2} + \frac{1}{t^2}\right)} \)
Previous and Related Work

• Magen[02]: Nearest Subspace Search
  – Query time is fast: \( (d + \log N + \frac{1}{\epsilon})^{O(1)} \)
  – Space is super-polynomial: \( 2^{(\log N)^{O(1)}} \)

Dual Problem: Database is a set of points, query is a \( k \)-flat

• [AIKN] for 1-flat: for any \( t > 0 \)
  – Query time: \( O(d^3 N^{0.5 + t}) \)
  – Space: \( d^2 N^{O(\frac{1}{\epsilon^2} + \frac{1}{t^2})} \)

• Very recently [MNSS] extended it for \( k \)-flats
  – Query time \( O\left(\frac{k}{n^{k+1-\rho} + t}\right) \)
  – Space: \( O\left(n^{1+\frac{k}{k+1-\rho}} + n \log^{O\left(\frac{1}{t}\right)} n\right) \)
Our Result

We give a randomized algorithm that for any sufficiently small $\epsilon$ reports a $(1 + \epsilon)$-approximate solution with high probability

- Space: $(N + d)^O\left(\frac{1}{\epsilon^2}\right)$
- Time: $\left(d + \log N + \frac{1}{\epsilon}\right)^O(1)$
Our Result

We give a randomized algorithm that for any sufficiently small $\epsilon$ reports a $(1 + \epsilon)$-approximate solution with high probability

- **Space**: $(N + d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- **Time**: $\left(d + \log N + \frac{1}{\epsilon}\right)^{O(1)}$
- Matches up to polynomials, the performance of best algorithm for ANN. No exponential dependence on $d$
Our Result

We give a randomized algorithm that for any sufficiently small $\epsilon$ reports a $(1 + \epsilon)$-approximate solution with high probability

- Space: $(N + d)^O\left(\frac{1}{\epsilon^2}\right)$
- Time: $\left(d + \log N + \frac{1}{\epsilon}\right)^O(1)$
- Matches up to polynomials, the performance of best algorithm for ANN. No exponential dependence on $d$
- The first algorithm with poly log query time and polynomial space for objects other than points
Our Result

We give a randomized algorithm that for any sufficiently small $\epsilon$ reports a $(1 + \epsilon)$-approximate solution with high probability

- Space: $(N + d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- Time: $\left(d + \log N + \frac{1}{\epsilon}\right)^{O(1)}$
- Matches up to polynomials, the performance of best algorithm for ANN. No exponential dependence on $d$
- The first algorithm with poly log query time and polynomial space for objects other than points
- Only uses reductions to ANN
Notation

- $L$ : the set of lines with size $N$
- $q$ : the query point
Notation

- $L$ : the set of lines with size $N$
- $q$ : the query point
- $B(c, r)$: ball of radius $r$ around $c$
Notation

- \( L \): the set of lines with size \( N \)
- \( q \): the query point
- \( B(c, r) \): ball of radius \( r \) around \( c \)
- \( \text{dist} \): the Euclidean distance between objects
Notation

- $L$ : the set of lines with size $N$
- $q$ : the query point
- $B(c, r)$: ball of radius $r$ around $c$
- $\text{dist}$ : the Euclidean distance between objects
- $\text{angle}$ : defined between lines
Notation

- $L$ : the set of lines with size $N$
- $q$ : the query point
- $B(c, r)$: ball of radius $r$ around $c$
- $\text{dist}$ : the Euclidean distance between objects
- $angle$ : defined between lines
- $\delta$-close: two lines $\ell$, $\ell'$ are $\delta$-close if $\sin(angle(\ell, \ell')) \leq \delta$. Similarly we define $\delta$-far/ strictly $\delta$-close/ strictly $\delta$-far
Notation

- $L$ : the set of lines with size $N$
- $q$ : the query point
- $B(c, r)$: ball of radius $r$ around $c$
- $\text{dist}$: the Euclidean distance between objects
- $\text{angle}$: defined between lines
- $\delta$-close: two lines $\ell$, $\ell'$ are $\delta$-close if $\sin(\text{angle}(\ell, \ell')) \leq \delta$. Similarly we define $\delta$-far/ strictly $\delta$-close/ strictly $\delta$-far
- $CP_{\ell_1 \rightarrow \ell_2}$ : closest point on $\ell_1$ to $\ell_2$
Unbounded Module
Net Module
Parallel Module

MODULES
Unbounded Module - Intuition

• All lines in $L$ pass through the origin $O$
Unbounded Module - Intuition

• All lines in $L$ pass through the origin $o$

• Data structure:
  – Project all lines onto any sphere $S(o, r)$ to get point set $P$
  – Build ANN data structure $ANN(P, \varepsilon)$
Unbounded Module - Intuition

• All lines in $L$ pass through the origin $o$

• Data structure:
  – Project all lines onto any sphere $S(o,r)$ to get point set $P$
  – Build ANN data structure $ANN(P, \epsilon)$

• Query Algorithm:
  – Project the query on $S(o,r)$ to get $q'$
  – Find the approximate closest point to $q'$, i.e., $p = ANN_{P}(q')$
  – Return the corresponding line of $p$
Unbounded Module

• All lines in $L$ pass through a small ball $B(o, r)$
• Query is far enough, outside of $B(o, R)$
• Use the same data structure and query algorithm
Unbounded Module

- All lines in $L$ pass through a small ball $B(o, r)$
- Query is far enough, outside of $B(o, R)$
- Use the same data structure and query algorithm

**Lemma:** if $R \geq \frac{r}{\varepsilon \delta}$, the returned line $\ell_p$ is
- Either an approximate closest line
- Or is $\delta$-close to the closest line $\ell^*$
Unbounded Module

- All lines in $L$ pass through a small ball $B(o, r)$
- Query is far enough, outside of $B(o, R)$
- Use the same data structure and query algorithm

**Lemma**: if $R \geq \frac{r}{\epsilon \delta}$, the returned line $\ell_p$ is
- Either an approximate closest line
- Or is $\delta$-close to the closest line $\ell^*$

This helps us further restrict our search to almost parallel lines to $\ell_p$
Net Module

• Intuition: sampling points from each line finely enough to get a set of points $P$, and building an $ANN(P, \epsilon)$ should suffice to find the approximate closest line.
Net Module

• Intuition: sampling points from each line finely enough to get a set of points $P$, and building an $ANN(P, \epsilon)$ should suffice to find the approximate closest line.

Lemma:
• Let $x$ be the separation parameter: distance between two adjacent samples on a line
• Then
  – Either the returned line $\ell_p$ is an approximate closest line
  – Or $\text{dist}(q, \ell_p) \leq x/\epsilon$
Parallel Module - Intuition

• All lines in $L$ are parallel
Parallel Module - Intuition

• All lines in $L$ are parallel
• Data structure:
  – Project all lines onto any hyper-plane $g$ which is perpendicular to all the lines to get point set $P$
  – Build ANN data structure $ANN(P, \epsilon)$
Parallel Module - Intuition

• All lines in $L$ are parallel
• Data structure:
  – Project all lines onto any hyper-plane $g$ which is perpendicular to all the lines to get point set $P$
  – Build ANN data structure $ANN(P, \epsilon)$
• Query algorithm:
  – Project the query on $g$ to get $q'$
  – Find the approximate closest point to $q'$, i.e., $p = ANN_P(q')$
  – Return the corresponding line to $p$
Parallel Module

- All lines in $L$ are $\delta$-close to a base line $\ell_b$
- Project the lines onto a hyper-plane $g$ which is perpendicular to $\ell_b$
- Query is close enough to $g$
- Use the same data structure and query algorithm
Parallel Module

- All lines in $L$ are $\delta$-close to a base line $\ell_b$
- Project the lines onto a hyper-plane $g$ which is perpendicular to $\ell_b$
- Query is close enough to $g$
- Use the same data structure and query algorithm

**Lemma**: if $\text{dist}(q, g) \leq \frac{D\varepsilon}{\delta}$, then
- Either the returned line $\ell_p$ is an approximate closest line
- Or $\text{dist}(q, \ell_p) \leq D$
Parallel Module

- All lines in $L$ are $\delta$-close to a base line $\ell_b$
- Project the lines onto a hyper-plane $g$ which is perpendicular to $\ell_b$
- Query is close enough to $g$
- Use the same data structure and query algorithm

**Lemma:** if $\text{dist}(q, g) \leq \frac{D\epsilon}{\delta}$, then
  - Either the returned line $\ell_p$ is an approximate closest line
  - Or $\text{dist}(q, \ell_p) \leq D$

Thus, for a set of almost parallel lines, we can use a set of parallel modules to cover a bounded region.
General Case
  • Input lines can have any configuration
  • Divergent Case
    • Input lines are $O(\epsilon)$-far from each other
  • Almost Parallel Case
    • Input lines are all $O(\epsilon)$-close to each other

ALGORITHMS
Outline of the Algorithms

• **Input**: a set of $n$ lines $S$
Outline of the Algorithms

• **Input**: a set of $n$ lines $S$
• Randomly choose a subset of $n/2$ lines $T$
Outline of the Algorithms

- **Input**: a set of $n$ lines $S$
- Randomly choose a subset of $n/2$ lines $T$
- Solve the problem over $T$ to get a line $\ell_p$
Outline of the Algorithms

- **Input**: a set of $n$ lines $S$
- Randomly choose a subset of $n/2$ lines $T$
- Solve the problem over $T$ to get a line $\ell_p$
- For $\log n$ iterations
  - Use $\ell_p$ to find a much closer line $\ell_p'$
  - Update $\ell_p$ with $\ell_p'$

Improvement step
Outline of the Algorithms

- **Input**: a set of $n$ lines $S$
- Randomly choose a subset of $n/2$ lines $T$
- Solve the problem over $T$ to get a line $\ell_p$
- For $\log n$ iterations
  - Use $\ell_p$ to find a much closer line $\ell_p'$
  - Update $\ell_p$ with $\ell_p'$

Improvement step
Outline of the Algorithms

- **Input:** a set of $n$ lines $S$
- Randomly choose a subset of $n/2$ lines $T$
- Solve the problem over $T$ to get a line $\ell_p$
- For $\log n$ iterations
  - Use $\ell_p$ to find a much closer line $\ell_p'$
  - Update $\ell_p$ with $\ell_p'$

Why?
Outline of the Algorithms

- **Input**: a set of $n$ lines $S$
- Randomly choose a subset of $n/2$ lines $T$
- Solve the problem over $T$ to get a line $\ell_p$
- For $\log n$ iterations
  - Use $\ell_p$ to find a much closer line $\ell'_p$
  - Update $\ell_p$ with $\ell'_p$

Let $\ell_1, \ldots, \ell_{\log n}$ be the $\log n$ closest lines to $q$ in the set $S$
Outline of the Algorithms

- **Input**: a set of $n$ lines $S$
- Randomly choose a subset of $n/2$ lines $T$
- Solve the problem over $T$ to get a line $\ell_p$
- For $\log n$ iterations
  - Use $\ell_p$ to find a much closer line $\ell_p'$
  - Update $\ell_p$ with $\ell_p'$

Let $\ell_1, \ldots, \ell_{\log n}$ be the $\log n$ closest lines to $q$ in the set $S$

With high probability at least one of $\{\ell_1, \ldots, \ell_{\log n}\}$ are sampled in $T$

- $\text{dist}(q, \ell_p) \leq \text{dist}(q, \ell_{\log n})(1 + \epsilon)$
- $\log n$ improvement steps suffices to find an approximate closest line
Improvement Step

Given a line \( \ell \), how to improve it, i.e., find a closer line?
Improvement Step

Given a line $\ell$, how to improve it, i.e., find a closer line?

- Data structure
- Query Processing Algorithm
General Case

• Search among all lines that are $\epsilon$-far from current line using Divergent Case
General Case

• Search among all lines that are $\epsilon$-far from current line using Divergent Case

• Search among the lines that are almost parallel to line found in previous step using Almost Parallel Case
Divergent Case

Assume any two lines are $\epsilon$-far; they diverge quickly.
Divergent Case

Assume any two lines are $\epsilon$-far; they diverge quickly.

- Let $\ell$ be the current line, and $\ell^*$ be the closest line to $q$
- Let $x = \text{dist}(q, \ell)$
- $\text{dist}(q, \ell^*) \leq x$
Divergent Case

Assume any two lines are $\epsilon$-far; they diverge quickly.

- Let $\ell$ be the current line, and $\ell^*$ be the closest line to $q$
- Let $x = \text{dist}(q, \ell)$
- $\text{dist}(q, \ell^*) \leq x$
  - All potential $\ell^*$ intersect $B(q, x)$
Divergent Case

Assume any two lines are $\varepsilon$-far; they diverge quickly.

- Let $\ell$ be the current line, and $\ell^*$ be the closest line to $q$
- Let $x = \text{dist}(q, \ell)$
- $\text{dist}(q, \ell^*) \leq x$
  - All potential $\ell^*$ intersect $B(q, x)$
  - Good news: we can build a net module inside $B(q, x)$ with separation parameter $x\varepsilon^2$ to improve over $\ell$
Divergent Case

Assume any two lines are $\epsilon$-far; they diverge quickly.

- Let $\ell$ be the current line, and $\ell^*$ be the closest line to $q$
- Let $x = \text{dist}(q, \ell)$
- $\text{dist}(q, \ell^*) \leq x$
  - All potential $\ell^*$ intersect $B(q, x)$
  - Good news: we can build a net module inside $B(q, x)$ with separation parameter $x\epsilon^2$ to improve over $\ell$
  - Bad news: we don’t know this ball in advance
Divergent Case contd.

What we know:

• $\text{dist}(\ell, \ell^*) \leq 2x$
• Let $q'$ be the projection of $q$ on $\ell$
Divergent Case contd.

What we know:

• \( \text{dist}(\ell, \ell^*) \leq 2x \)
• Let \( q' \) be the projection of \( q \) on \( \ell \)
Divergent Case contd.

What we know:

• $\text{dist}(\ell, \ell^*) \leq 2x$

• Let $q'$ be the projection of $q$ on $\ell$
  
  – $CP_{\ell \to \ell^*}$ is not farther than $\frac{x}{\epsilon}$ from $q'$
  since they are $\epsilon$-far
Divergent Case contd.

What we know:

- $\text{dist}(\ell, \ell^*) \leq 2x$
- Let $q'$ be the projection of $q$ on $\ell$
  - $CP_{\ell \rightarrow \ell^*}$ is not farther than $\frac{x}{\epsilon}$ from $q'$ since they are $\epsilon$-far
  - $B(q', O\left(\frac{x}{\epsilon}\right))$ touches all such lines
Data Structure

For each $\ell \in S$

- Sort all lines $\ell'$ according to their distance from $\ell$
Data Structure

For each $\ell \in S$

• Sort all lines $\ell'$ according to their distance from $\ell$
• For all $1 \leq i \leq n$, let $S_i$ be the $i^{th}$ closest lines
Data Structure

For each \( \ell \in S \)
- Sort all lines \( \ell' \) according to their distance from \( \ell \)
- For all \( 1 \leq i \leq n \), let \( S_i \) be the \( i^{th} \) closest lines
  - Sort all lines in \( S_i \) such as \( \ell' \) according to the position of \( CP_{\ell \rightarrow \ell'} \)
Data Structure

For each $\ell \in S$

- Sort all lines $\ell'$ according to their distance from $\ell$
- For all $1 \leq i \leq n$, let $S_i$ be the $i^{th}$ closest lines
  - Sort all lines in $S_i$ such as $\ell'$ according to the position of $CP_{\ell \rightarrow \ell'}$
  - For each interval of lines $A$ in sorted $S_i$
Data Structure

For each $\ell \in S$

- Sort all lines $\ell'$ according to their distance from $\ell$
- For all $1 \leq i \leq n$, let $S_i$ be the $i^{th}$ closest lines
  - Sort all lines in $S_i$ such as $\ell'$ according to the position of $CP_{\ell \rightarrow \ell'}$
  - For each interval of lines $A$ in sorted $S_i$
    - Find smallest ball $B_A(o_A, r_A)$ with its center on $\ell$ which intersects all lines in $A$
      $\Rightarrow (r_A \leq O(\frac{\chi}{\epsilon}))$
Data Structure

For each $\ell \in S$

- Sort all lines $\ell'$ according to their distance from $\ell$
- For all $1 \leq i \leq n$, let $S_i$ be the $i^{th}$ closest lines
  - Sort all lines in $S_i$ such as $\ell'$ according to the position of $CP_{\ell \rightarrow \ell'}$
  - For each interval of lines $A$ in sorted $S_i$
    - Find smallest ball $B_A(o_A, r_A)$ with its center on $\ell$ which intersects all lines in $A$
      $\rightarrow (r_A \leq O(\frac{\epsilon}{\epsilon}))$
    - Construct a net module inside of the ball of $B(o_A, r_A/\epsilon^2)$ with separation $r_A \epsilon^3$
      ($\# \text{samples} = O(n \frac{r_A}{(\epsilon^2 r_A \epsilon^3)}) = O(n/\epsilon^5)$)
Data Structure

For each $\ell \in S$
- Sort all lines $\ell'$ according to their distance from $\ell$
- For all $1 \leq i \leq n$, let $S_i$ be the $i^{th}$ closest lines
  - Sort all lines in $S_i$ such as $\ell'$ according to the position of $CP_{\ell \rightarrow \ell'}$
  - For each interval of lines $A$ in sorted $S_i$
    - Find smallest ball $B_A(o_A, r_A)$ with its center on $\ell$ which intersects all lines in $A$
      $\Rightarrow (r_A \leq O(\frac{x}{\epsilon}))$
    - Construct a net module inside of the ball of $B(o_A, r_A/\epsilon^2)$ with separation $r_A\epsilon^3$
      (**samples** = $O(n \cdot r_A/(\epsilon^2 r_A \epsilon^3)) = O(n/\epsilon^5))$
    - Construct an unbounded module outside of $B_A\left(o_A, \frac{1}{\epsilon^2} r_A\right)$
Query Processing Algorithm

Given query point $q$
Query Processing Algorithm

Given query point $q$

- Project $q$ on $\ell$ to get $q'$
- Use binary search to find the set $A$ of all lines $\ell'$ that are within distance $2x$ of $\ell$, and that $CP_{\ell \rightarrow \ell'}$ is within distance $2x/\epsilon$ of $q'$
Query Processing Algorithm

Given query point $q$

- Project $q$ on $\ell$ to get $q'$
- Use binary search to find the set $A$ of all lines $\ell'$ that are within distance $2x$ of $\ell$, and that $CP_{\ell \rightarrow \ell'}$ is within distance $2x/\varepsilon$ of $q'$
Given query point $q$

- Project $q$ on $\ell$ to get $q'$
- Use binary search to find the set $A$ of all lines $\ell'$ that are within distance $2x$ of $\ell$, and that $CP_{\ell \rightarrow \ell'}$ is within distance $2x/\epsilon$ of $q'$
- Let $B_A(o_A, r_A)$ be the corresponding ball
Query Processing Algorithm

Given query point $q$

- Project $q$ on $\ell$ to get $q'$
- Use binary search to find the set $A$ of all lines $\ell'$ that are within distance $2x$ of $\ell$, and that $CP_{\ell \rightarrow \ell'}$ is within distance $2x/\epsilon$ of $q'$
- Let $B_A(o_A, r_A)$ be the corresponding ball
- If $x \in B_A(o_A, \frac{r_A}{\epsilon^2})$ use net module:
  - Find approximate closest line -> done!
  - Or find a line with distance at most $r_A \epsilon^2 \leq x \epsilon$ ($r_A \leq x/\epsilon$) -> we improved
Query Processing Algorithm

Given query point $q$

- Project $q$ on $\ell$ to get $q'$
- Use binary search to find the set $A$ of all lines $\ell'$ that are within distance $2x$ of $\ell$, and that $CP_{\ell \rightarrow \ell'}$ is within distance $2x/\varepsilon$ of $q'$
- Let $B_A(o_A, r_A)$ be the corresponding ball
- If $x \in B_A(o_A, \frac{r_A}{\varepsilon^2})$ use net module:
  - Find approximate closest line -> done!
  - Or find a line with distance at most $r_A \varepsilon^2 \leq x \varepsilon$ ($r_A \leq x/\varepsilon$) -> we improved
- Otherwise use unbounded module to find the approximate closest line -> done!
Almost Parallel

All lines are $2\varepsilon$-close to each other.

For each line $\ell$

• Partition the space into slabs using perpendicular hyperplanes to $\ell$ s.t. for any pair of lines $\ell_1, \ell_2$: 
All lines are $2\varepsilon$-close to each other.

For each line $\ell$

- Partition the space into slabs using perpendicular hyperplanes to $\ell$ s.t. for any pair of lines $\ell_1, \ell_2$:
  - In each slab the relative order of $\text{dist}_{H(\ell, o)}(\ell, \ell_1)$ and $\text{dist}_{H(\ell, o)}(\ell, \ell_2)$ on the hyper-plane remains the same as we move $o$ on $\ell$ in the slab

There is a unique ordering of the lines
Almost Parallel

All lines are $2\epsilon$-close to each other.

For each line $\ell$

- Partition the space into slabs using perpendicular hyperplanes to $\ell$ s.t. for any pair of lines $\ell_1, \ell_2$:
  - In each slab the relative order of $\text{dist}_{H(\ell, o)}(\ell, \ell_1)$ and $\text{dist}_{H(\ell, o)}(\ell, \ell_2)$ on the hyper-plane remains the same as we move $o$ on $\ell$ in the slab.

There is a unique ordering of the lines

- $\text{dist}_{H(\ell, o)}(\ell_1, \ell_2)$ on the hyper-plane is monotone
Almost Parallel

All lines are $2\varepsilon$-close to each other.

For each line $\ell$

• Partition the space into slabs using perpendicular hyperplanes to $\ell$ s.t. for any pair of lines $\ell_1, \ell_2$:
  – In each slab the relative order of $\text{dist}_H(\ell, o)(\ell, \ell_1)$ and $\text{dist}_H(\ell, o)(\ell, \ell_2)$ on the hyper-plane remains the same as we move $o$ on $\ell$ in the slab

There is a unique ordering of the lines

– $\text{dist}_H(\ell, o)(\ell_1, \ell_2)$ on the hyper-plane is monotone

The minimum ball intersecting any prefix of lines have its center on the boundary of slab
Almost Parallel

All lines are $2\varepsilon$-close to each other.

For each line $\ell$

- Partition the space into slabs using perpendicular hyperplanes to $\ell$ s.t. for any pair of lines $\ell_1, \ell_2$:
  - In each slab the relative order of $\text{dist}_{H(\ell,o)}(\ell, \ell_1)$ and $\text{dist}_{H(\ell,o)}(\ell, \ell_2)$ on the hyper-plane remains the same as we move $o$ on $\ell$ in the slab

There is a unique ordering of the lines

- $\text{dist}_{H(\ell,o)}(\ell_1, \ell_2)$ on the hyper-plane is monotone

The minimum ball intersecting any prefix of lines have its center on the boundary of slab.

- $O(n^2)$ slabs suffices
Data Structure in Each Slab

- For each $i$, let $B(o, r)$ be the smallest ball touching the closest $i^{th}$ lines s.t. $o \in \ell$. We know $o$ would be on the boundary of slab.
Data Structure in Each Slab

- For each $i$, let $B(o, r)$ be the smallest ball touching the closest $i^{th}$ lines s.t. $o \in \ell$. We know $o$ would be on the boundary of slab.
Data Structure in Each Slab

- For each $i$, let $B(o, r)$ be the smallest ball touching the closest $i^{th}$ lines s.t. $o \in \ell$. We know $o$ would be on the boundary of slab.
Data Structure in Each Slab

- For each $i$, let $B(o, r)$ be the smallest ball touching the closest $i^{th}$ lines s.t. $o \in \ell$. We know $o$ would be on the boundary of slab.
- Let $\delta_0 > \cdots > \delta_t$ be all pairwise angles
- Let $R_0 = \frac{r}{\epsilon \delta_0}, \ldots, R_t = \frac{r}{\epsilon \delta_t}$
- Consider the balls $B(o, R_0), \ldots, B(o, R_t)$
Data Structure in Each Slab

- For each \( i \), let \( B(o, r) \) be the smallest ball touching the closest \( i^{th} \) lines s.t. \( o \in \ell \). We know \( o \) would be on the boundary of slab.

- Let \( \delta_0 > \cdots > \) be all pairwise angles.

- Let \( R_0 = \frac{r}{\varepsilon \delta_0}, \ldots, R_t = \frac{r}{\varepsilon \delta_t} \)

- Consider the balls \( B(o, R_0), \ldots, B(o, R_t) \)

- Build net module inside \( B(o, R_0) \)
Data Structure in Each Slab

- For each \( i \), let \( B(o, r) \) be the smallest ball touching the closest \( i^{th} \) lines s.t. \( o \in \ell \). We know \( o \) would be on the boundary of slab.
- Let \( \delta_0 > \cdots > \delta_t \) be all pairwise angles
- Let \( R_0 = \frac{r}{\epsilon \delta_0}, \ldots, R_t = \frac{r}{\epsilon \delta_t} \)
- Consider the balls \( B(o, R_0), \ldots, B(o, R_t) \)
- Build net module inside \( B(o, R_0) \)
Data Structure in Each Slab

- For each $i$, let $B(o, r)$ be the smallest ball touching the closest $i^{th}$ lines s.t. $o \in \ell$. We know $o$ would be on the boundary of slab.
- Let $\delta_0 > \cdots > \delta_t$ be all pairwise angles
- Let $R_0 = \frac{r}{\epsilon \delta_0}, \ldots, R_t = \frac{r}{\epsilon \delta_t}$
- Consider the balls $B(o, R_0), \ldots, B(o, R_t)$
- Build net module inside $B(o, R_0)$
- For each ball $B(o, R_i)$
  - Build unbounded module on it
Data Structure in Each Slab

- For each $i$, let $B(o, r)$ be the smallest ball touching the closest $i^{th}$ lines s.t. $o \in \ell$. We know $o$ would be on the boundary of slab.
- Let $\delta_0 > \cdots > \delta_t$ be all pairwise angles.
- Let $R_0 = \frac{r}{\epsilon \delta_0}, \ldots, R_t = \frac{r}{\epsilon \delta_t}$
- Consider the balls $B(o, R_0), \ldots, B(o, R_t)$
- Build net module inside $B(o, R_0)$
- For each ball $B(o, R_i)$
  - Build unbounded module on it
  - For each line $\ell_b$
    - Build a set of parallel modules with $\ell_b$ as their base line for all the lines that are $\delta_i$-close to $\ell_b$, so that they cover the space between $B(o, R_i)$ and $B(o, R_{i+1})$ with separation $R_{i+1} \epsilon$
Query Processing Algorithm

• Given $q$, find the right slab, and retrieve all candidate lines
• Using binary search find $r$
Query Processing Algorithm

• Given $q$, find the right slab, and retrieve all candidate lines
• Using binary search find $r$
• Find largest $i$ such that $q \notin B(o, R_i)$
Query Processing Algorithm

- Given \( q \), find the right slab, and retrieve all candidate lines
- Using binary search find \( r \)
- Find largest \( i \) such that \( q \notin B(o, R_i) \)
- Use the unbounded module of \( B(o, R_i) \) to find a line \( \ell' \), we know
  - Either \( \ell' \) is an approximate closest line -> done
  - It is \( \delta_{i+1} \)-close to \( \ell^* \)
Query Processing Algorithm

- Given \( q \), find the right slab, and retrieve all candidate lines.
- Using binary search find \( r \).
- Find largest \( i \) such that \( q \not\in B(o, R_i) \).
- Use the unbounded module of \( B(o, R_i) \) to find a line \( \ell' \), we know
  - Either \( \ell' \) is an approximate closest line -> done
  - It is \( \delta_{i+1} \)-close to \( \ell^* \)
- Use the parallel modules of \( \ell' \) to find an approximate closest line. -> done.
Query Processing Algorithm

- Given \( q \), find the right slab, and retrieve all candidate lines
- Using binary search find \( r \)
- Find largest \( i \) such that \( q \notin B(o, R_i) \)
- Use the unbounded module of \( B(o, R_i) \) to find a line \( \ell' \), we know
  - Either \( \ell' \) is an approximate closest line -> done
  - It is \( \delta_{i+1} \)-close to \( \ell^* \)
- Use the parallel modules of \( \ell' \) to find an approximate closest line. -> done
Summary

- Nearest Line Search Problem
Summary

• Nearest Line Search Problem
• Modules: unbounded, net, parallel
Summary

• Nearest Line Search Problem
• Modules: unbounded, net, parallel
• Use of random sampling
Summary

• Nearest Line Search Problem
• Modules: unbounded, net, parallel
• Use of random sampling
• How to improve given a line
Summary

• Nearest Line Search Problem
• Modules: unbounded, net, parallel
• Use of random sampling
• How to improve given a line
• Bounds of our algorithm
  – Polynomial Space:
    \[
    \left( \frac{dN}{\epsilon} \right)^{O(1)} \times S \left( \left( \frac{N}{\epsilon} \right)^{O(1)}, \epsilon \right) = O(N + d)^{O\left(\frac{1}{\epsilon^2}\right)}
    \]
  – Poly-logarithmic query time:
    \[
    (d \log N)^{O(1)} \times T \left( \left( \frac{N}{\epsilon} \right)^{O(1)}, \epsilon \right) = \left( d + \log N + \frac{1}{\epsilon} \right)^{O(1)}
    \]
Future Work

• The current result is not good in practice
  – Large exponents
  – Algorithm is complicated
Can we get a simpler algorithms?
Future Work

• The current result is not good in practice
  – Large exponents
  – Algorithm is complicated
  Can we get a simpler algorithms?

• Generalization to higher dimensional flats
Future Work

• The current result is not good in practice
  – Large exponents
  – Algorithm is complicated
  Can we get a simpler algorithms?

• Generalization to higher dimensional flats

• Generalization to other objects, e.g. balls
THANK YOU!