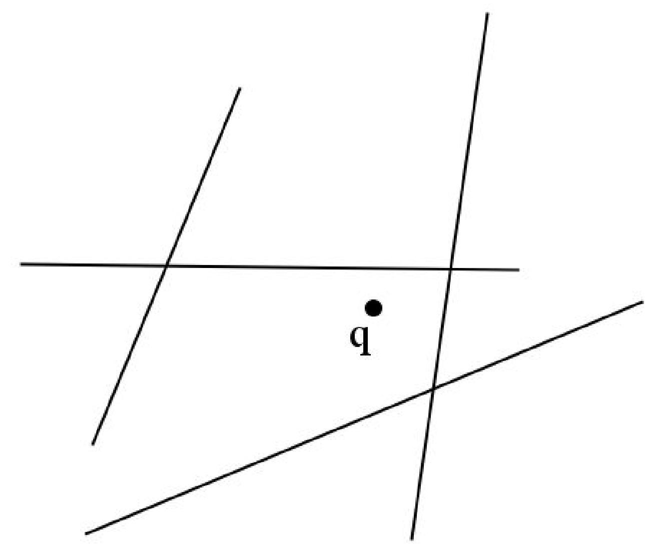
Approximate Nearest Line Search in High Dimensions

Sepideh Mahabadi

Massachusetts Institute of Technology

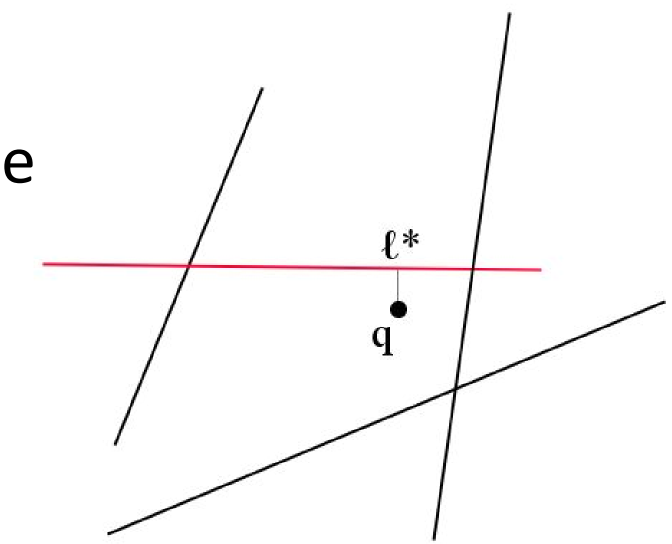
The NLS Problem

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



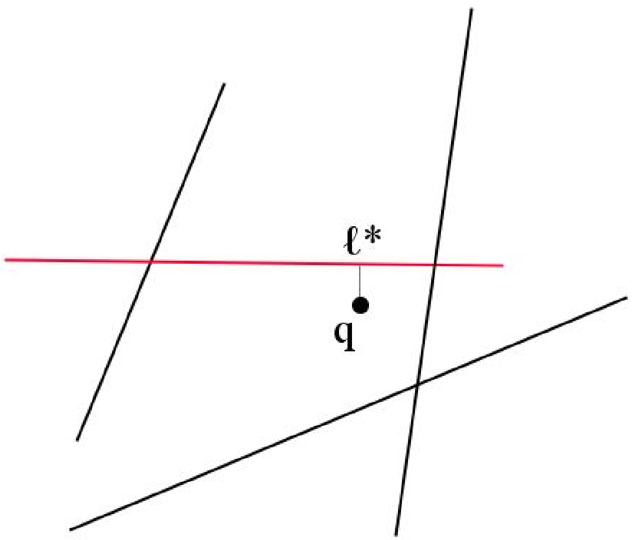
The NLS Problem

- 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 ℓ^* to q



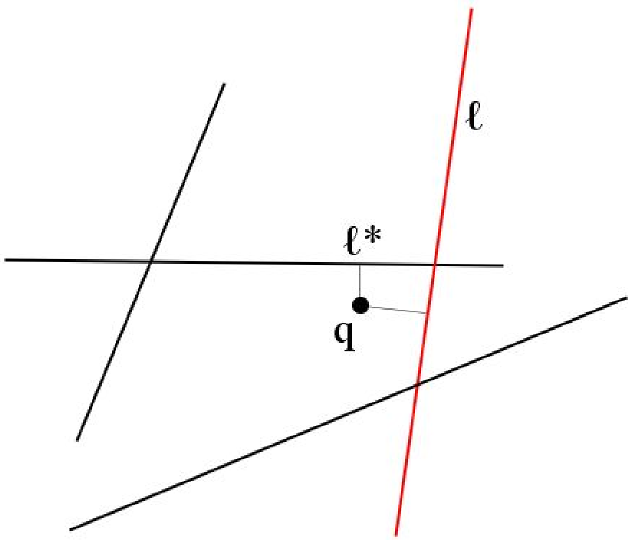
The NLS Problem

- 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 ℓ^* 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 ℓ^* to q
  - polynomial space
  - sub-linear query time



Approximation

- Finds an approximate closest line ℓ

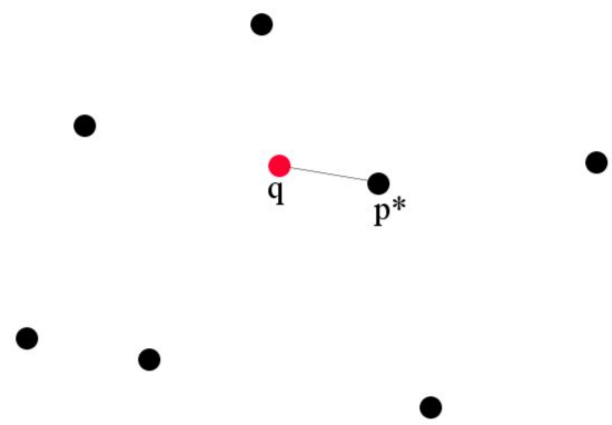
$$dist(q, \ell) \leq dist(q, \ell^*)(1 + \epsilon)$$

Nearest Neighbor Problems
Motivation
Previous Work
Our result
Notation

BACKGROUND

Nearest Neighbor Problem

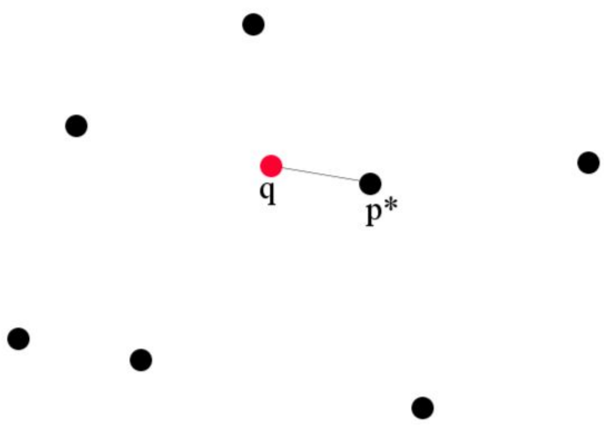
NN: Given a set of N points P, build a data structure s.t. given a query point q, finds the closest point p^* to q.



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.

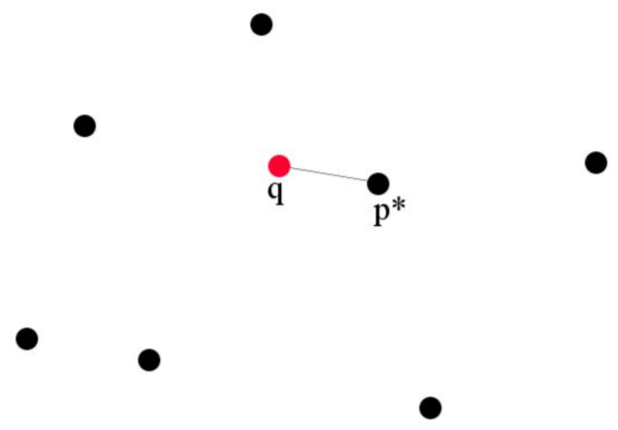
- Applications: database, information retrieval, pattern recognition, computer vision
  - Features: dimensions
  - Objects: points
  - Similarity: distance between points



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.

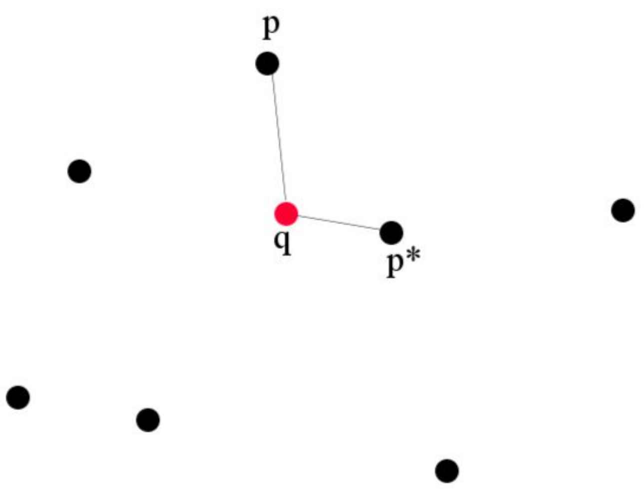
- Applications: database, information retrieval, pattern recognition, computer vision
  - Features: dimensions
  - Objects: points
  - Similarity: distance between points
- Current solutions suffer from "curse of dimensionality":
  - Either **space** or **query time** is **exponential** in d
  - Little 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.,

$$dist(q,p) \leq dist(q,p^*)(1+\epsilon)$$

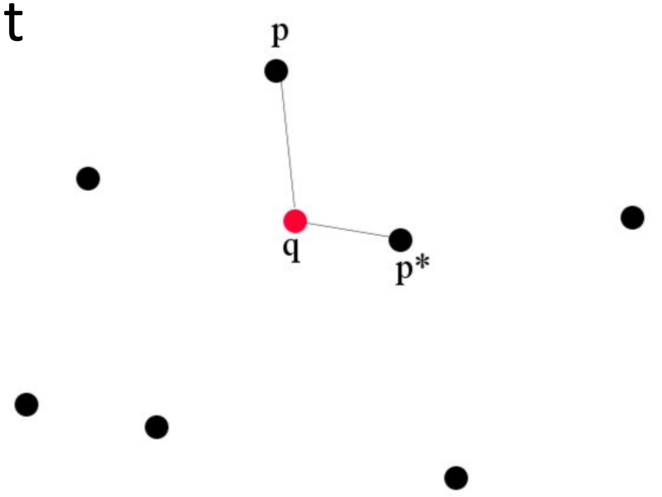


Approximate Nearest Neighbor(ANN)

- 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.,

$$dist(q,p) \leq dist(q,p^*)(1+\epsilon)$$

- 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

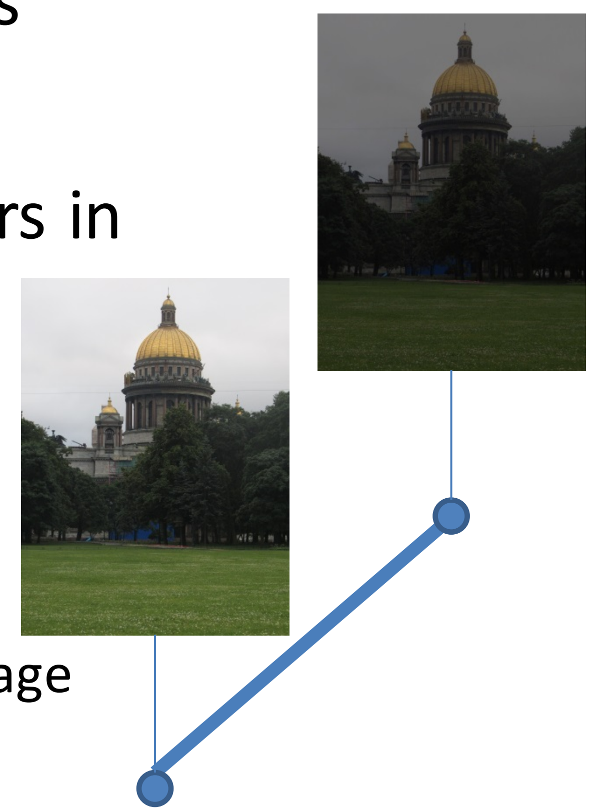
One of the simplest generalizations of ANN: data items are represented by k-flats (affine subspace) instead of points

- 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 for constant k
  - Query time is fast : $\left(d + \log N + \frac{1}{\epsilon}\right)^{O(1)}$
  - Space is super-polynomial : $2^{(\log N)^{O(1)}}$

Previous and Related Work

- Magen[02]: Nearest Subspace Search for constant k
  - Query time is fast : $\left(d + \log N + \frac{1}{\epsilon}\right)^{O(1)}$
  - 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 for constant k
  - Query time is fast : $\left(d + \log N + \frac{1}{\epsilon}\right)^{O(1)}$
  - 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: $d^2 N^{O\left(\frac{1}{\epsilon^2}+\frac{1}{t^2}\right)}$
- Very recently [MNSS] extended it for k-flats
  - Query time $O\left(n^{\frac{k}{k+1-\rho}+t}\right)$
  - Space: $O(n^{1+\frac{\sigma k}{k+1-\rho}} + n \log^{O\left(\frac{1}{t}\right)} n)$

Our Result

We give a randomized algorithm that for any sufficiently small ϵ 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 ϵ 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 ϵ 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 ϵ 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

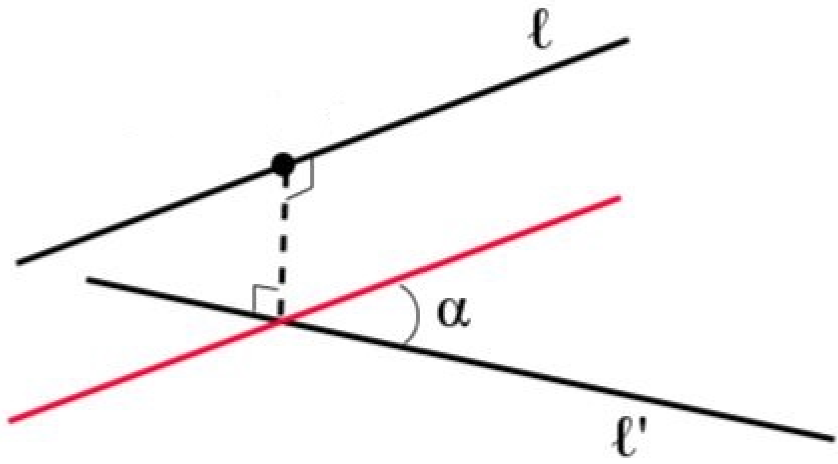
- L : the set of lines with size N
- q : the query point
- $B(c, r)$: ball of radius r around c

Notation

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

Notation

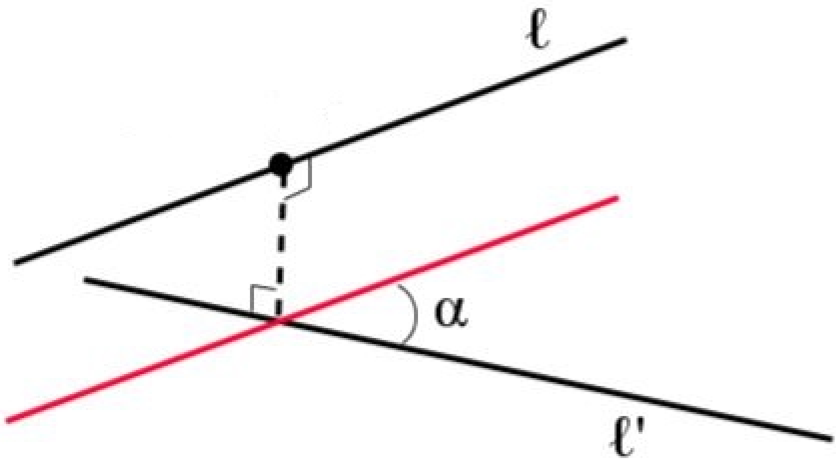
- L : the set of lines with size N
- q : the query point
- $B(c,r)$: ball of radius r around c
- $dist$: the Euclidean distance between objects
- $angle$: defined between lines



Notation

- L : the set of lines with size N
- q : the query point
- $B(c, r)$: ball of radius r around c
- $dist$: the Euclidean distance between objects
- $angle$: defined between lines
- δ-close: two lines ℓ , ℓ' are δ-close if

$$\sin(angle(\ell, \ell')) \leq \delta$$



Net Module

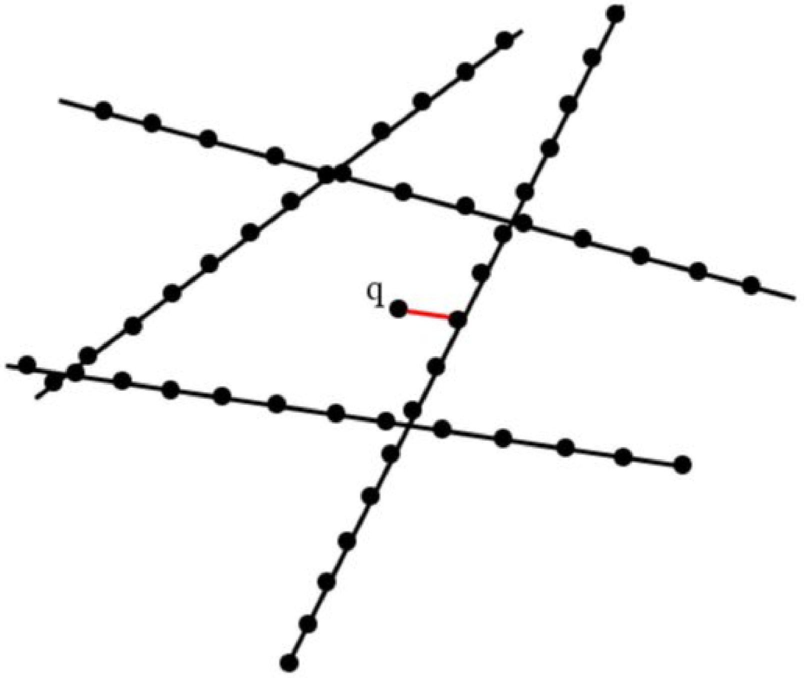
Unbounded Module

Parallel Module

MODULES

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.

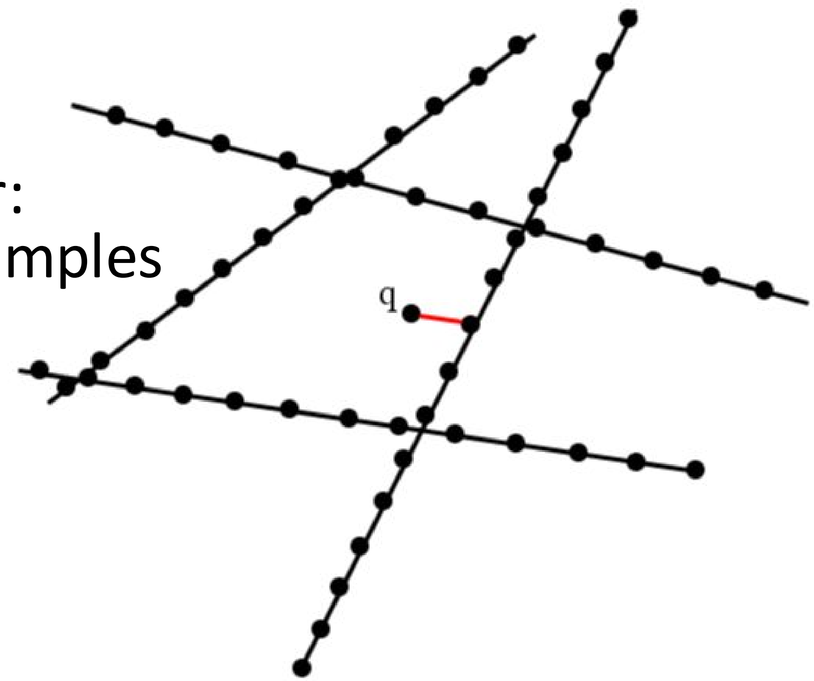


Net Module

- 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 ℓ_p is an approximate closest line
  - Or $dist(q, \ell_p) \leq x/\epsilon$

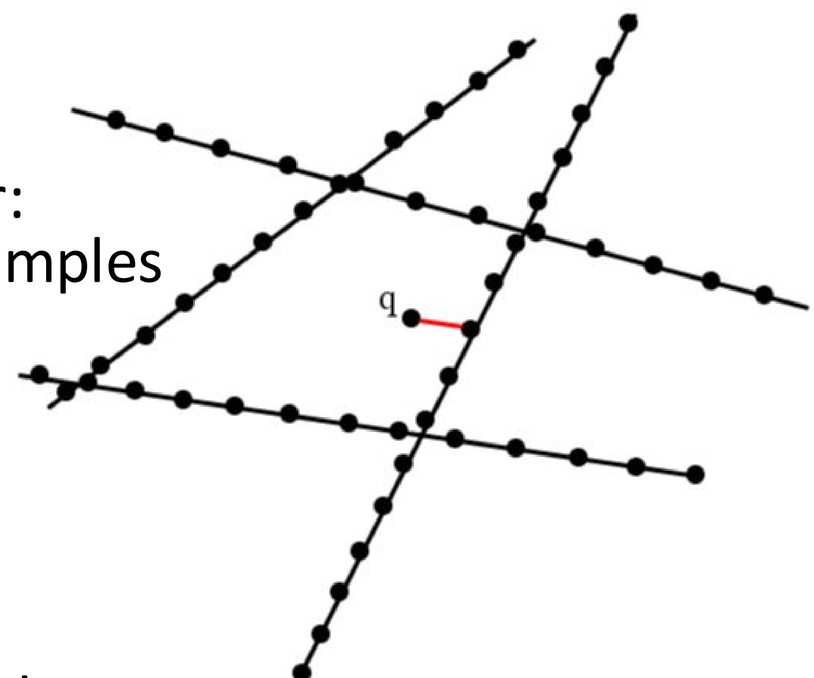


Net Module

- 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 ℓ_p is an approximate closest line
  - Or $dist(q, \ell_p) \leq x/\epsilon$

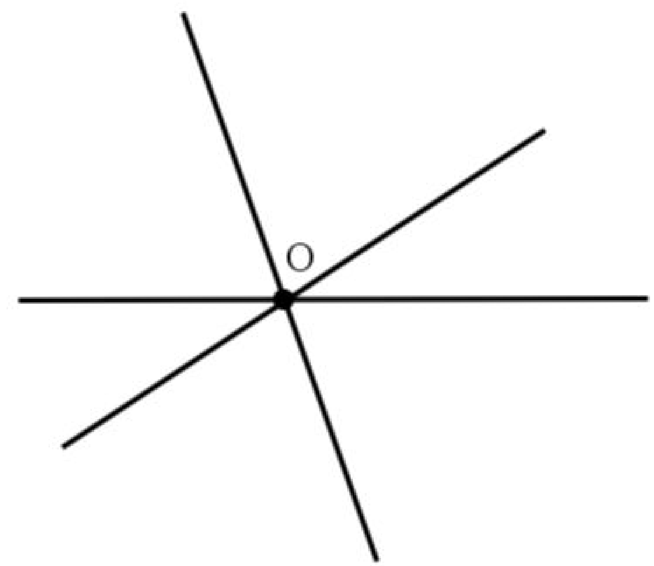


Issue:

It should be used inside a bounded region

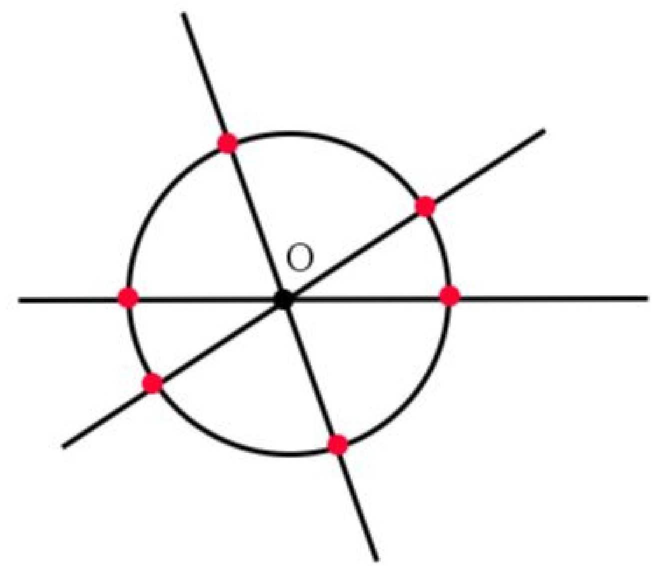
Unbounded Module - Intuition

- All lines in L pass through the origin o



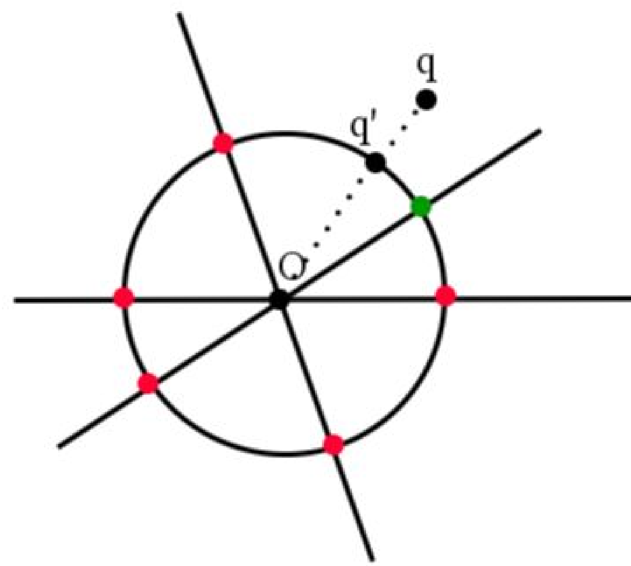
Unbounded Module - Intuition

- 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 $ANN(P,\epsilon)$



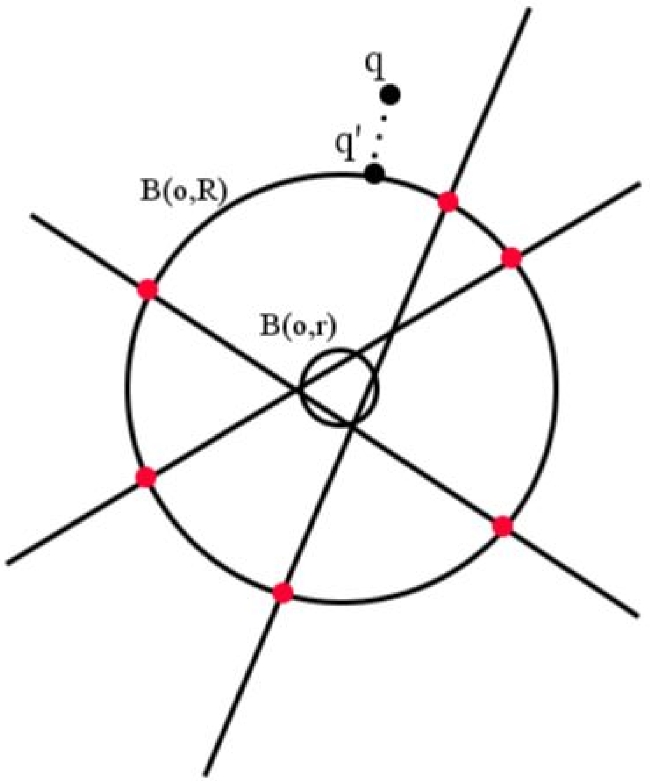
Unbounded Module - Intuition

- 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

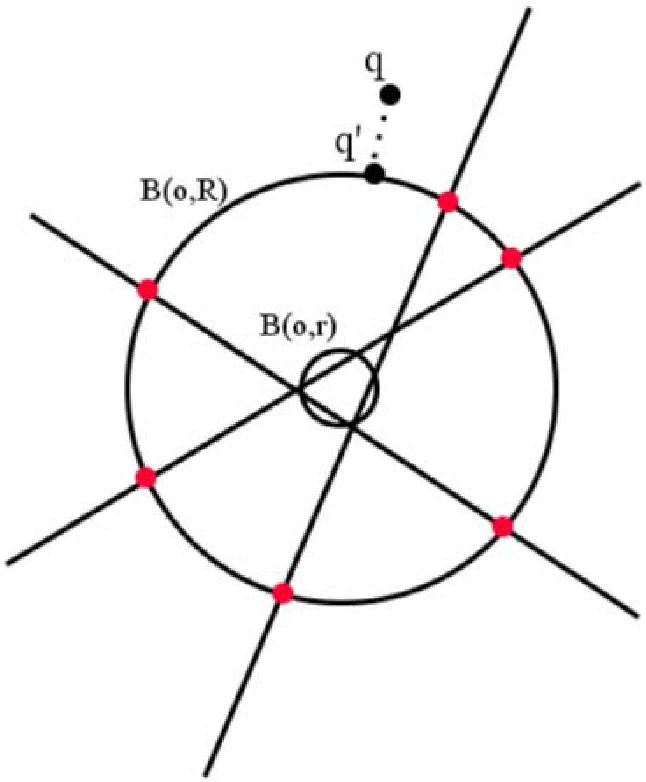


Unbounded Module

- 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

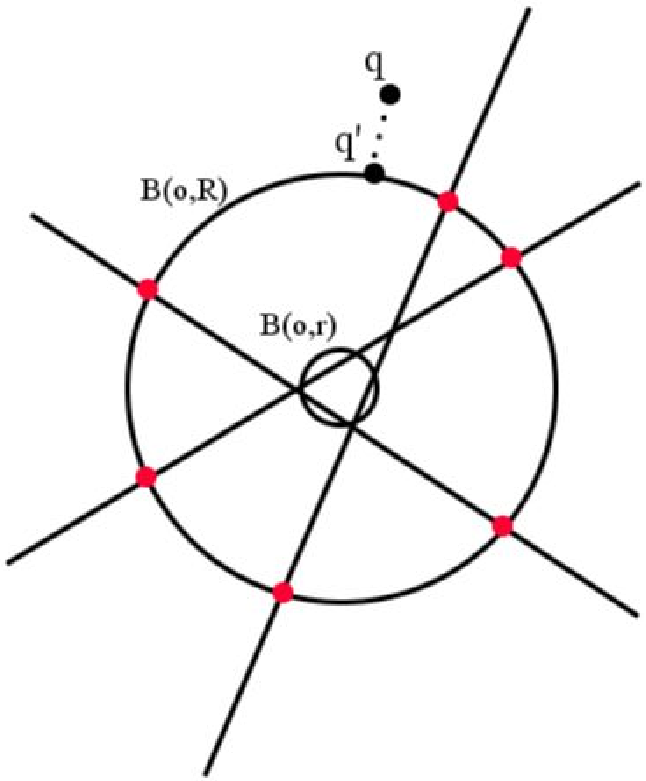
Lemma: if $R \geq \frac{r}{\epsilon\delta}$, the returned line ℓ_p is

- Either an approximate closest line
- Or is δ-close to the closest line ℓ^*



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 ℓ_p is

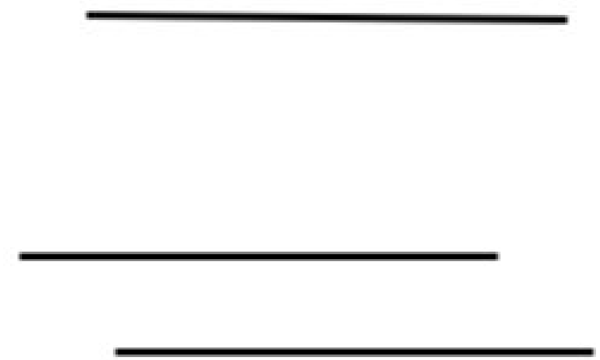
- Either an approximate closest line
- Or is δ-close to the closest line ℓ^*

This helps us in two ways

- Bound the region for the net module
- Restrict search to almost parallel lines

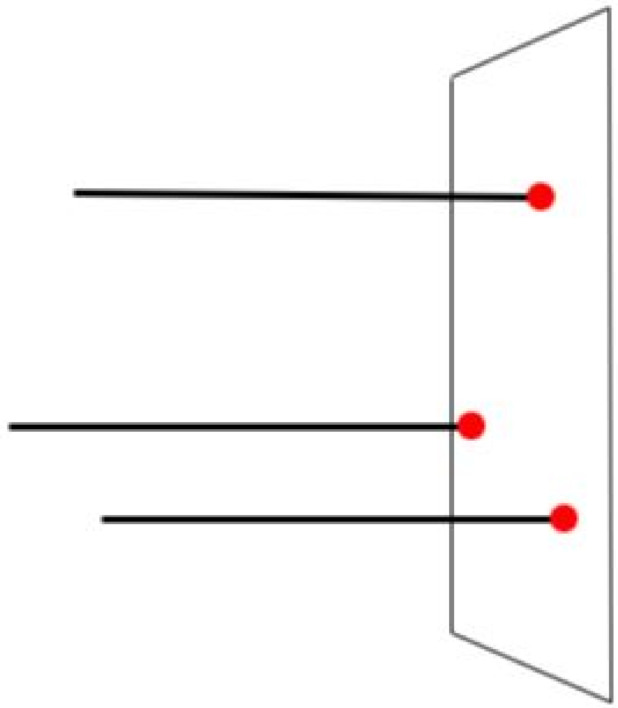
Parallel Module - Intuition

- All lines in L are parallel



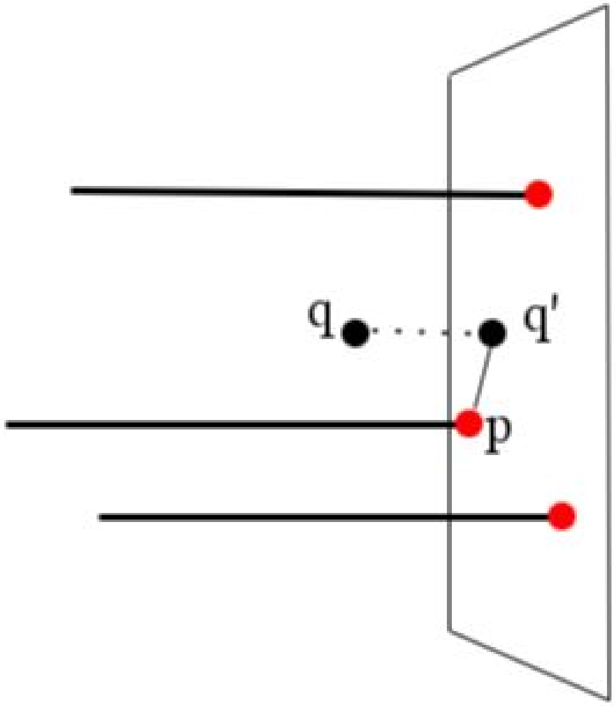
Parallel Module - Intuition

- 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 $ANN(P, \epsilon)$



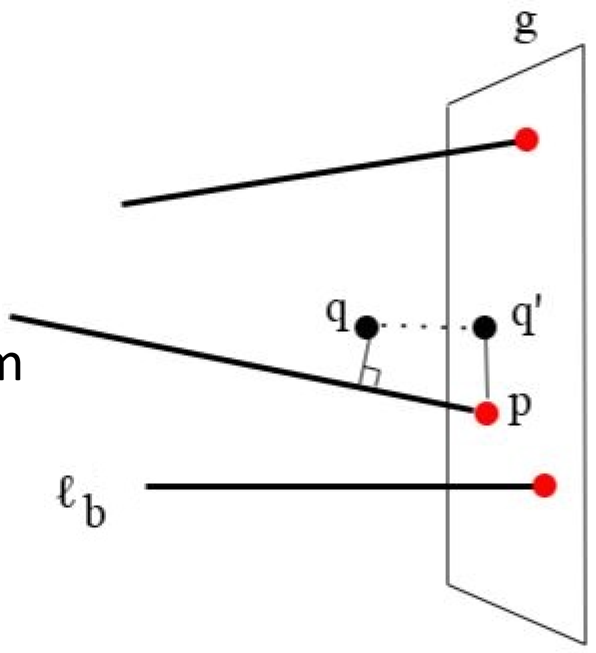
Parallel Module - Intuition

- 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



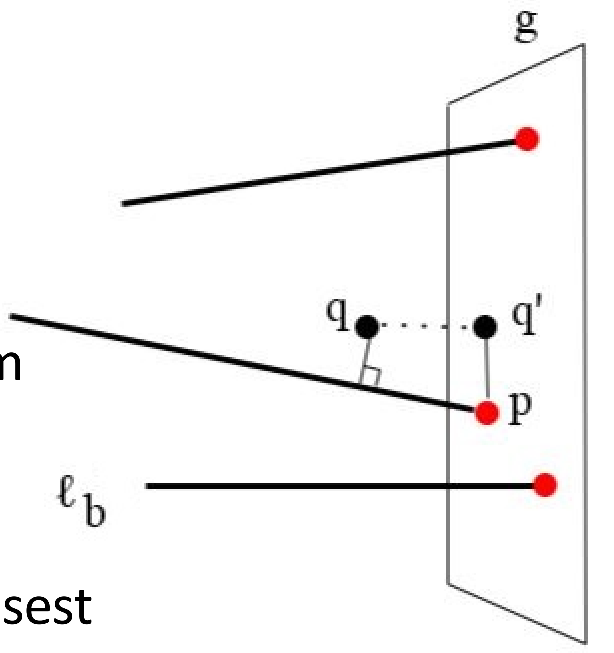
Parallel Module

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



Parallel Module

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

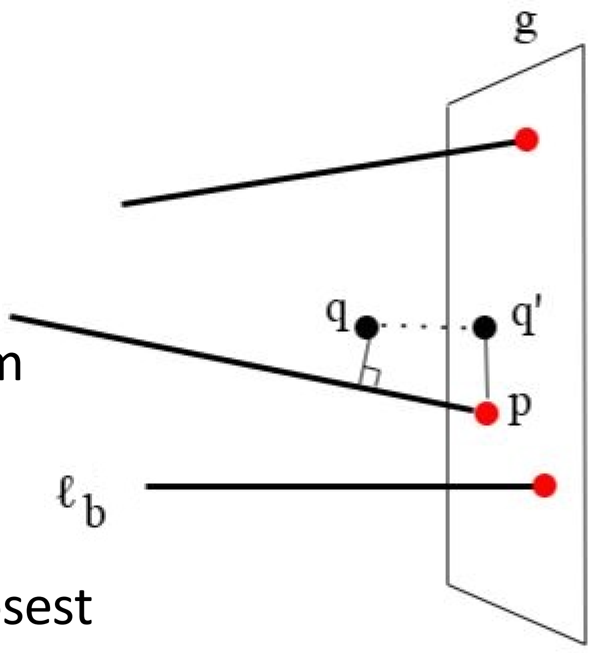


Lemma: if $dist(q, g) \leq \frac{D\epsilon}{\delta}$, then

- Either the returned line ℓ_p is an approximate closest line
- Or $dist(q, \ell_p) \leq D$

Parallel Module

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



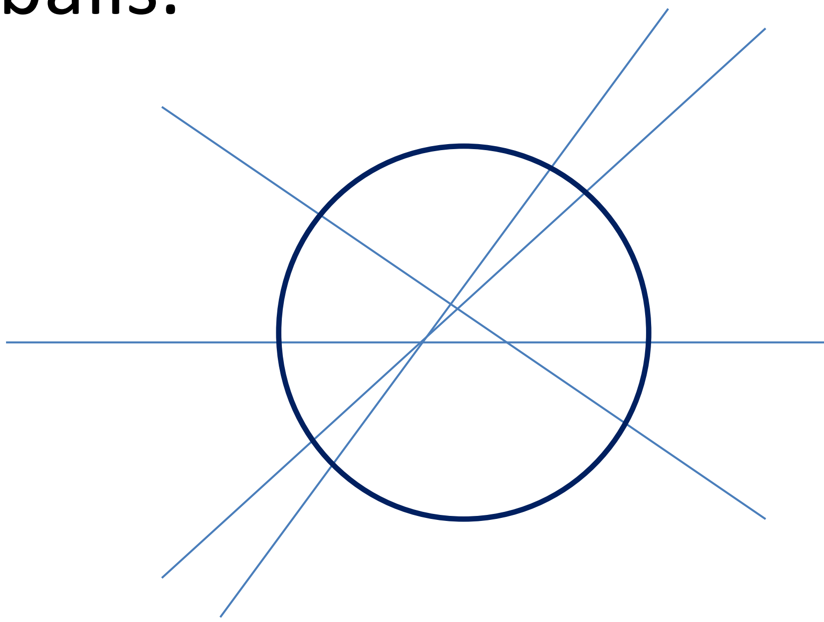
Lemma: if $dist(q, g) \leq \frac{D\epsilon}{\delta}$, then

- Either the returned line ℓ_p is an approximate closest line
- Or $dist(q, \ell_p) \leq D$

Thus, for a set of almost parallel lines, we can use a set of parallel modules to cover a bounded region.

How the Modules Work Together

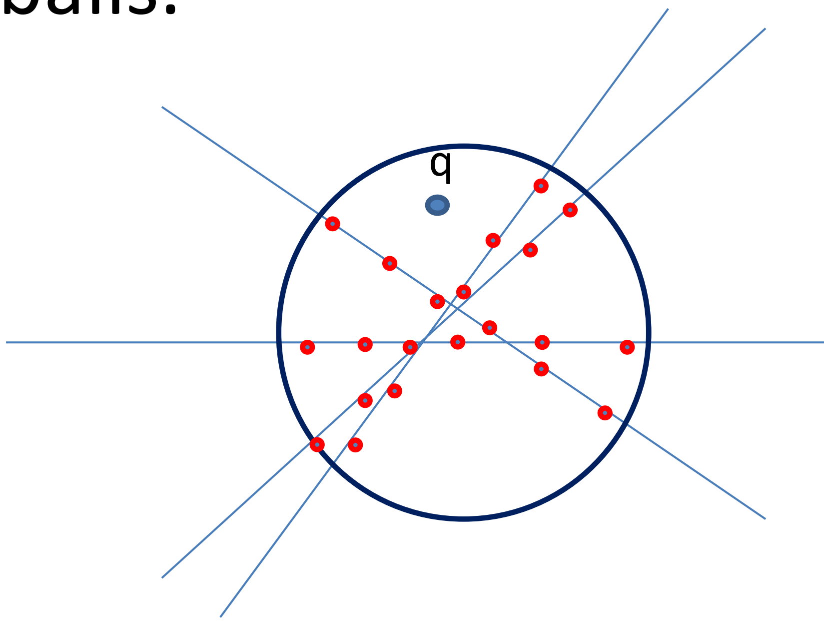
Given a set of lines, we come up with a polynomial number of balls.



How the Modules Work Together

Given a set of lines, we come up with a polynomial number of balls.

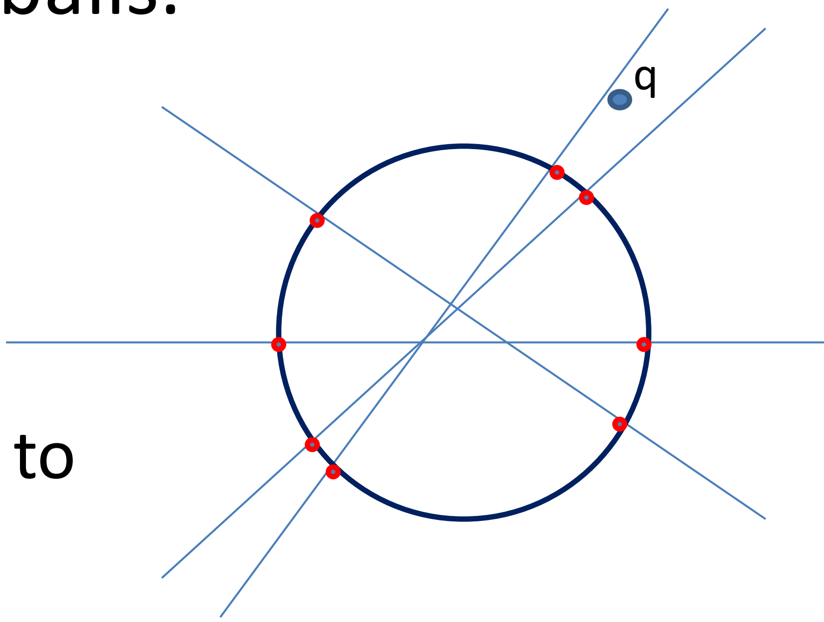
- If q is inside the ball
  - Use net module



How the Modules Work Together

Given a set of lines, we come up with a polynomial number of balls.

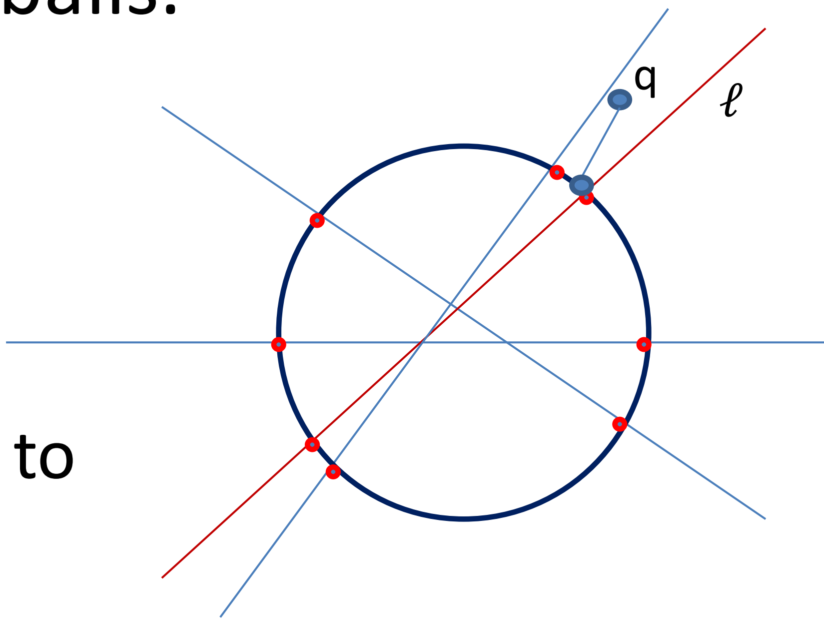
- If q is inside the ball
  - Use net module
- If q is outside the ball
  - First use unbounded module to find a line ℓ



How the Modules Work Together

Given a set of lines, we come up with a polynomial number of balls.

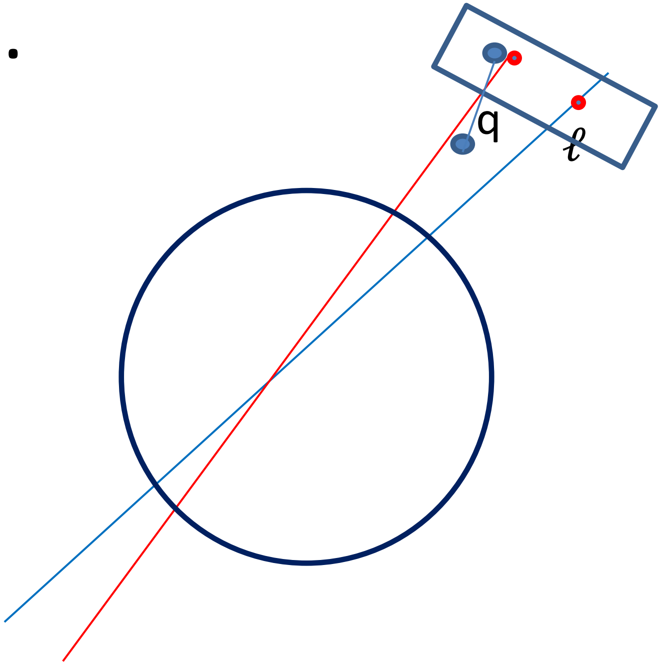
- If q is inside the ball
  - Use net module
- If q is outside the ball
  - First use unbounded module to find a line ℓ



How the Modules Work Together

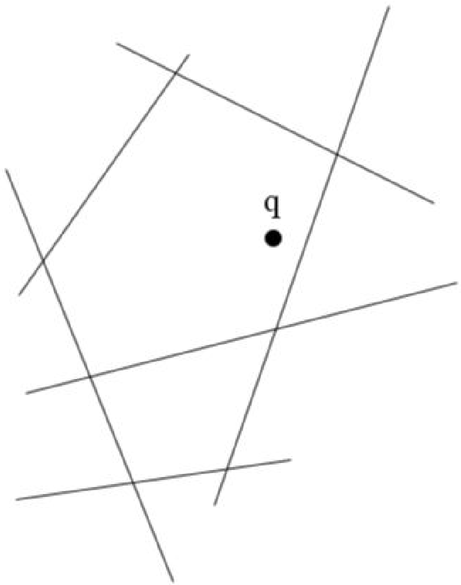
Given a set of lines, we come up with a polynomial number of balls.

- If q is inside the ball
  - Use net module
- If q is outside the ball
  - First use unbounded module to find a line ℓ
  - Then use parallel module to search among parallel lines to ℓ



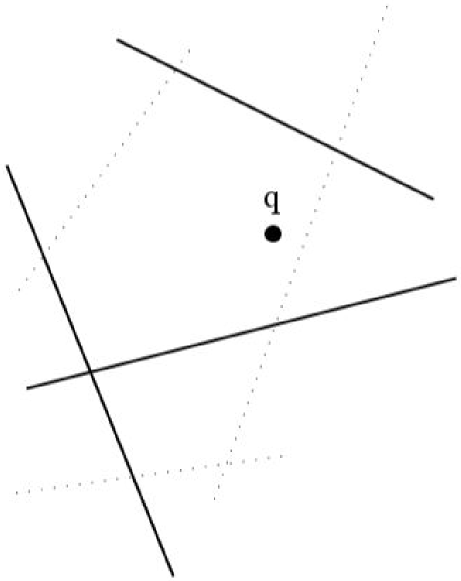
Outline of the Algorithms

- **Input**: a set of n lines S



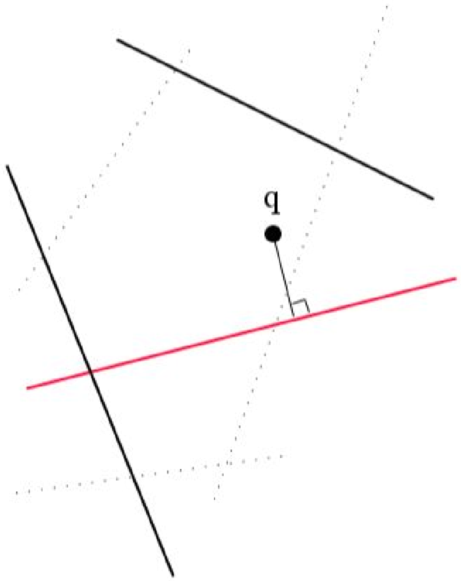
Outline of the Algorithms

- **Input**: a set of n lines S
- Randomly choose a subset of $n/2$ lines T



Outline of the Algorithms

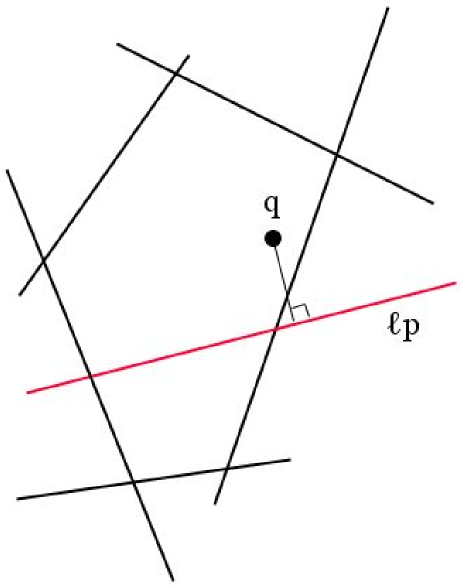
- **Input**: a set of n lines S
- Randomly choose a subset of $n/2$ lines T
- Solve the problem over T to get a line ℓ_p



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 ℓ_p
- For $\log n$ iterations
  - Use ℓ_p to find a much closer line $\ell_p{}'$
  - Update ℓ_p with ℓ_p'

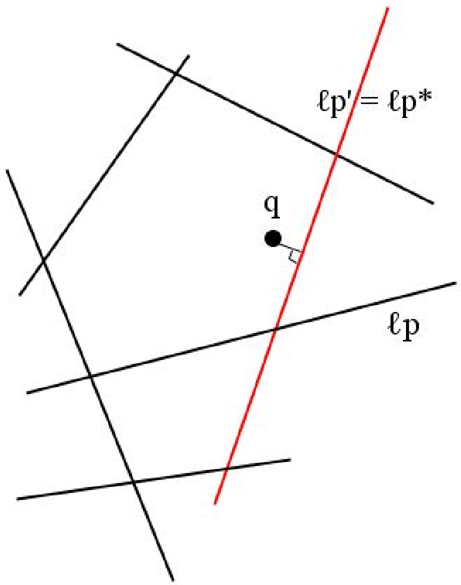
Improvement step



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 ℓ_p
- For $\log n$ iterations
  - Use ℓ_p to find a much closer line $\ell_p{}'$
  - Update ℓ_p with ℓ_p'

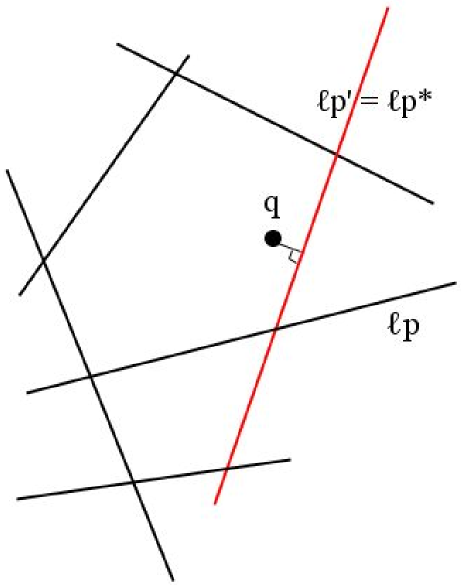
Improvement step



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 ℓ_p
- For $\log n$ iterations
  - Use ℓ_p to find a much closer line $\ell_p{}'$
  - Update ℓ_p with ℓ_p'

Improvement step

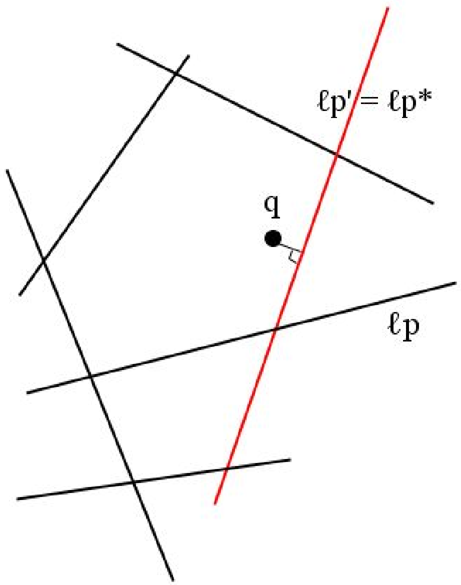


Why?

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 ℓ_p
- For $\log n$ iterations
  - Use ℓ_p to find a much closer line ${\ell_p}'$
  - Update ℓ_p with ℓ_p'

Improvement step

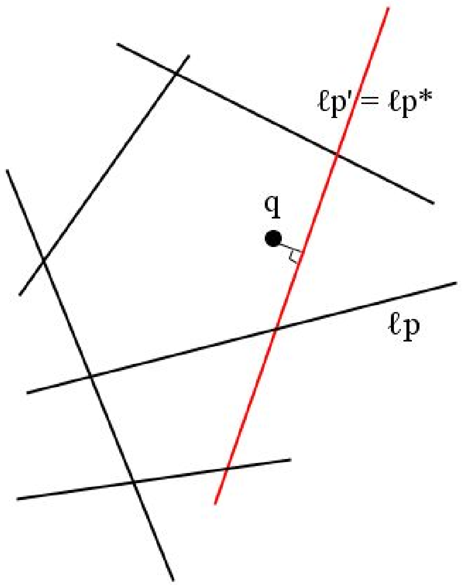


Let $s_1, \ldots, s_{\log n}$ be the $\log n$ closest lines to q in the set S

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 ℓ_p
- For $\log n$ iterations
  - Use ℓ_p to find a much closer line $\ell_p{}'$
  - Update ℓ_p with ℓ_p'

Improvement step



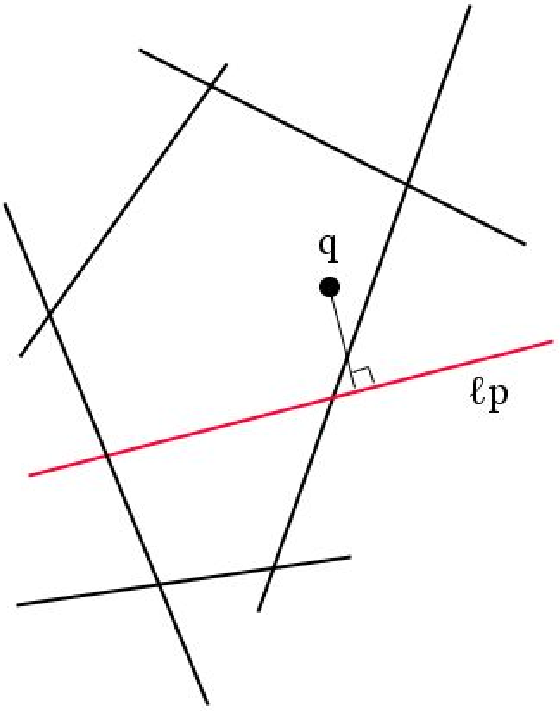
Let $s_1, \ldots, s_{\log n}$ be the $\log n$ closest lines to q in the set S

With high probability at least one of $\{s_1, \ldots, s_{\log n}\}$ is sampled in T

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

Improvement Step

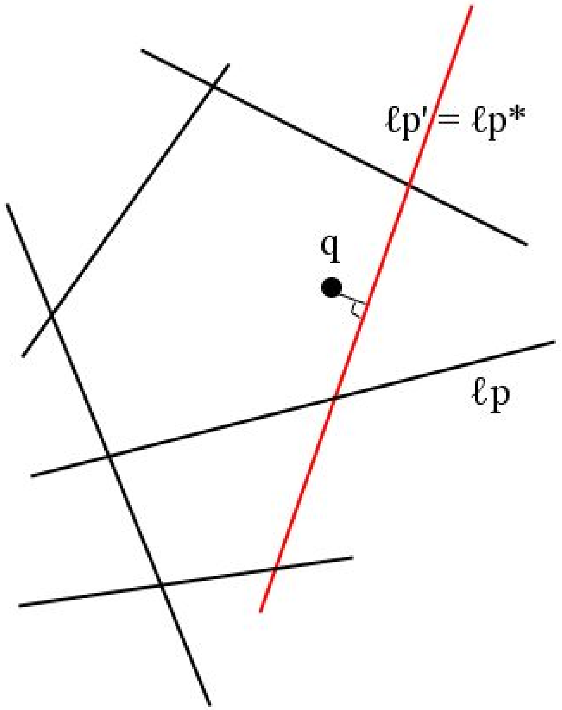
Given a line ℓ, how to improve it, i.e., find a closer line?



Improvement Step

Given a line ℓ, how to improve it, i.e., find a closer line?

- Data structure
- Query Processing Algorithm

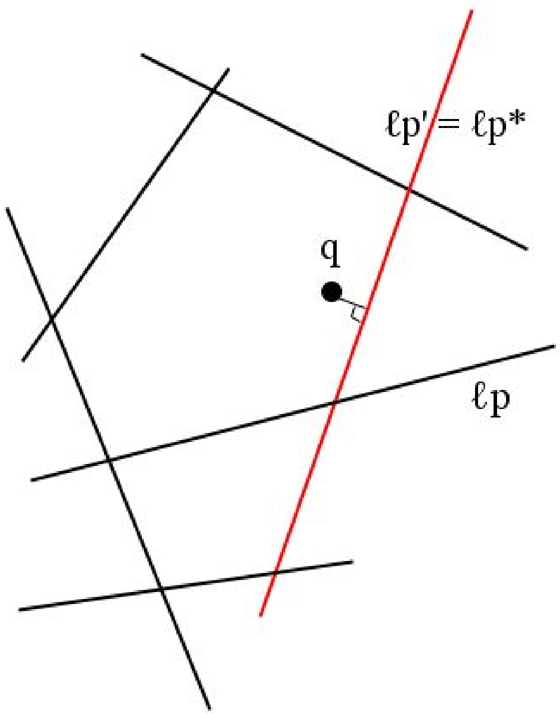


Improvement Step

Given a line ℓ, how to improve it, i.e., find a closer line?

- Data structure
- Query Processing Algorithm

Use the three modules here



Conclusion

Bounds we get for NLS problem

- Polynomial Space: $O(N+d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- Poly-logarithmic query time : $\left(d+\log N+\frac{1}{\epsilon}\right)^{O(1)}$

Conclusion

Bounds we get for NLS problem

- Polynomial Space: $O(N+d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- Poly-logarithmic query time : $\left(d+\log N+\frac{1}{\epsilon}\right)^{O(1)}$

Future Work

- The current result is not efficient in practice
  - Large exponents
  - Algorithm is complicated

Conclusion

Bounds we get for NLS problem

- Polynomial Space: $O(N+d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- Poly-logarithmic query time : $\left(d+\log N+\frac{1}{\epsilon}\right)^{O(1)}$

Future Work

- The current result is not efficient in practice
  - Large exponents
  - Algorithm is complicated
- Can we get a simpler algorithms?

Conclusion

Bounds we get for NLS problem

- Polynomial Space: $O(N+d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- Poly-logarithmic query time : $\left(d+\log N+\frac{1}{\epsilon}\right)^{O(1)}$

Future Work

- The current result is not efficient in practice
  - Large exponents
  - Algorithm is complicated
- Can we get a simpler algorithms?
- Generalization to higher dimensional flats

Conclusion

Bounds we get for NLS problem

- Polynomial Space: $O(N + d)^{O\left(\frac{1}{\epsilon^2}\right)}$
- Poly-logarithmic query time : $\left(d + \log N + \frac{1}{\epsilon}\right)^{O(1)}$

Future Work

- The current result is not efficient in practice
  - Large exponents
  - Algorithm is complicated
- Can we get a simpler algorithm?
- Generalization to higher dimensional flats
- Generalization to other objects, e.g. balls

THANK YOU!