New directions in nearest neighbor searching with applications to lattice sieving

Anja Becker, Léo Ducas, Nicolas Gama, Thijs Laarhoven

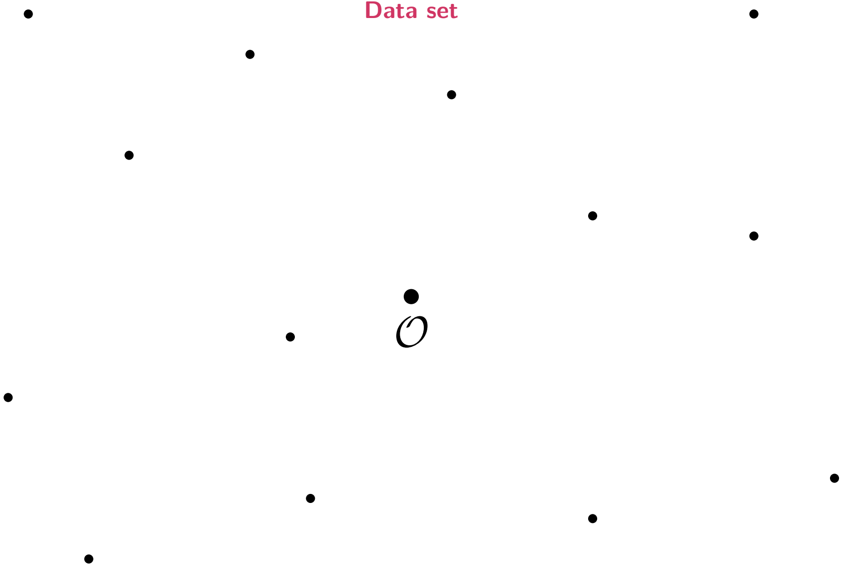
mail@thijs.com
http://www.thijs.com/

SODA 2016, Arlington (VA), USA
(January 10, 2016)

Nearest neighbor searching

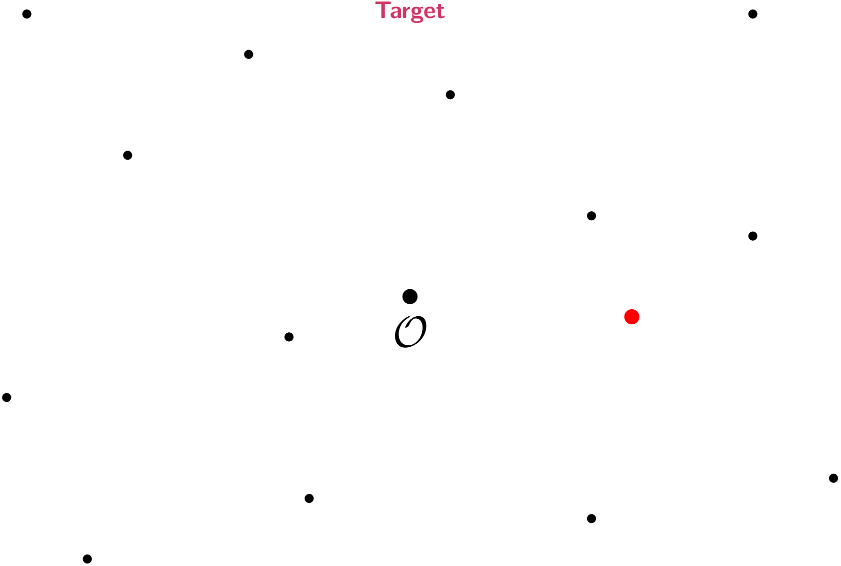
Nearest neighbor searching

Data set



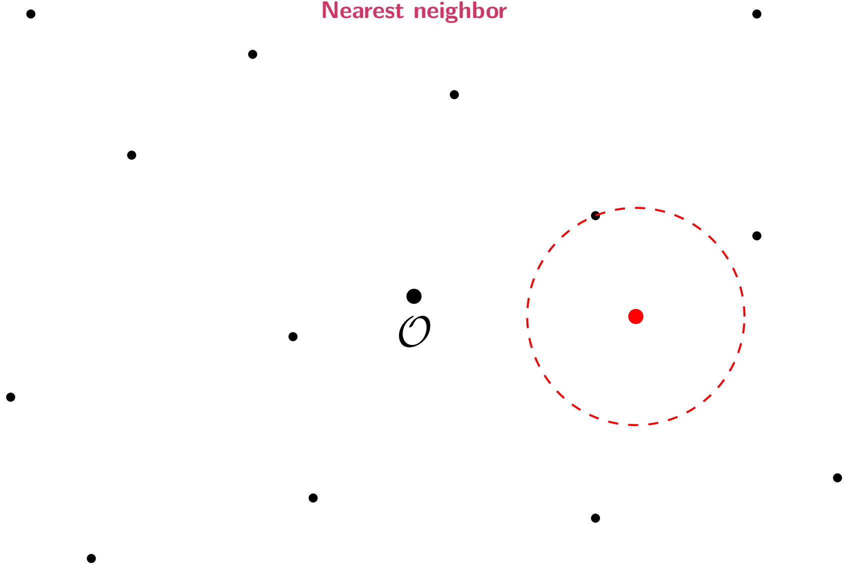
Nearest neighbor searching

Target



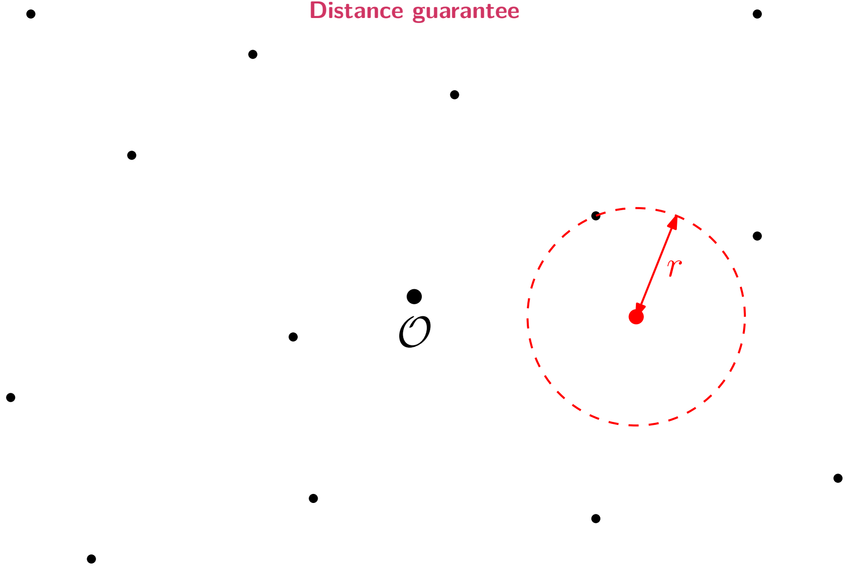
Nearest neighbor searching

Nearest neighbor



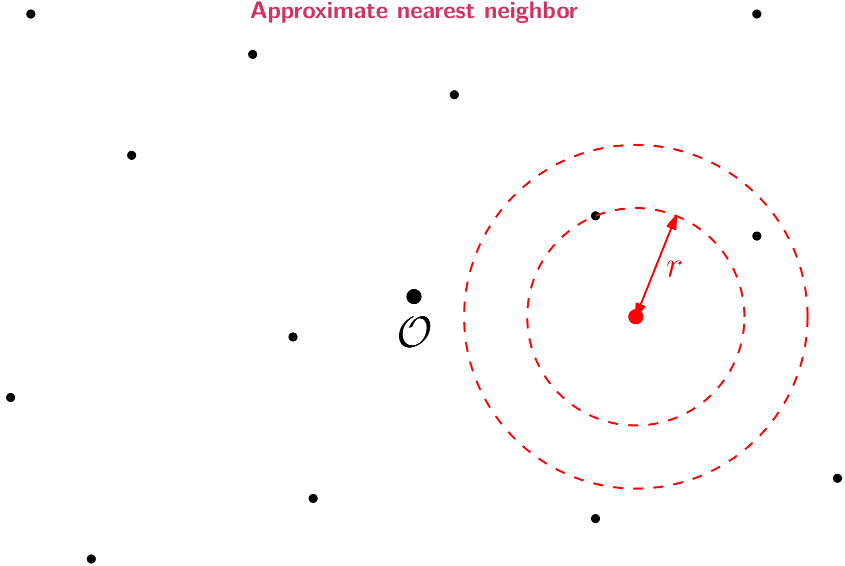
Nearest neighbor searching

Distance guarantee



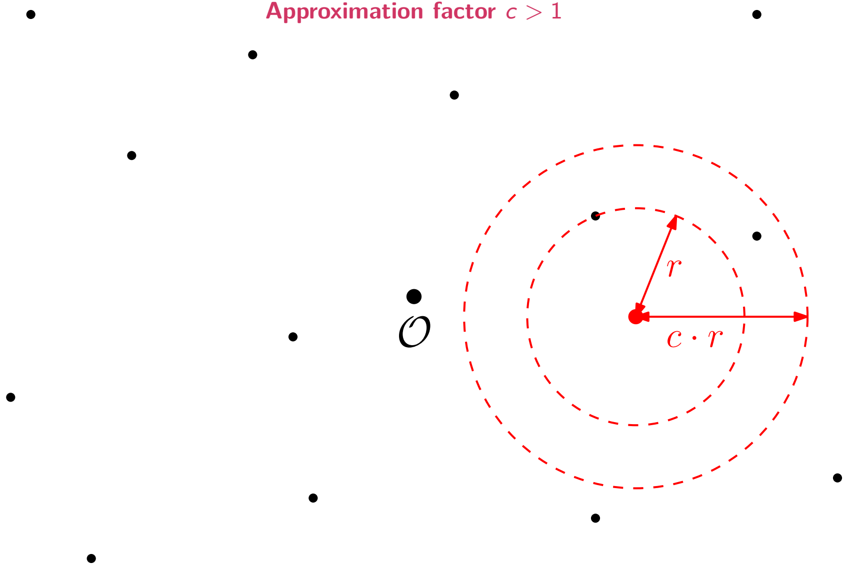
Nearest neighbor searching

Approximate nearest neighbor



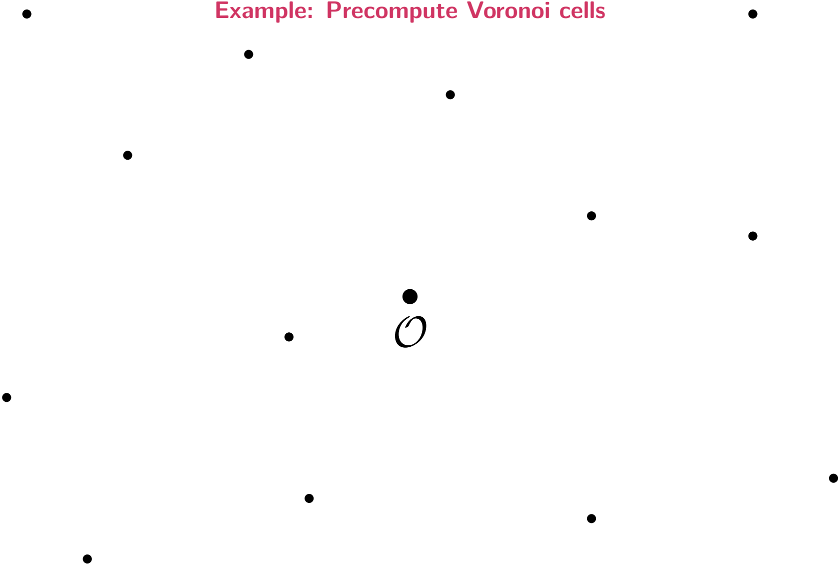
Nearest neighbor searching

Approximation factor $c > 1$



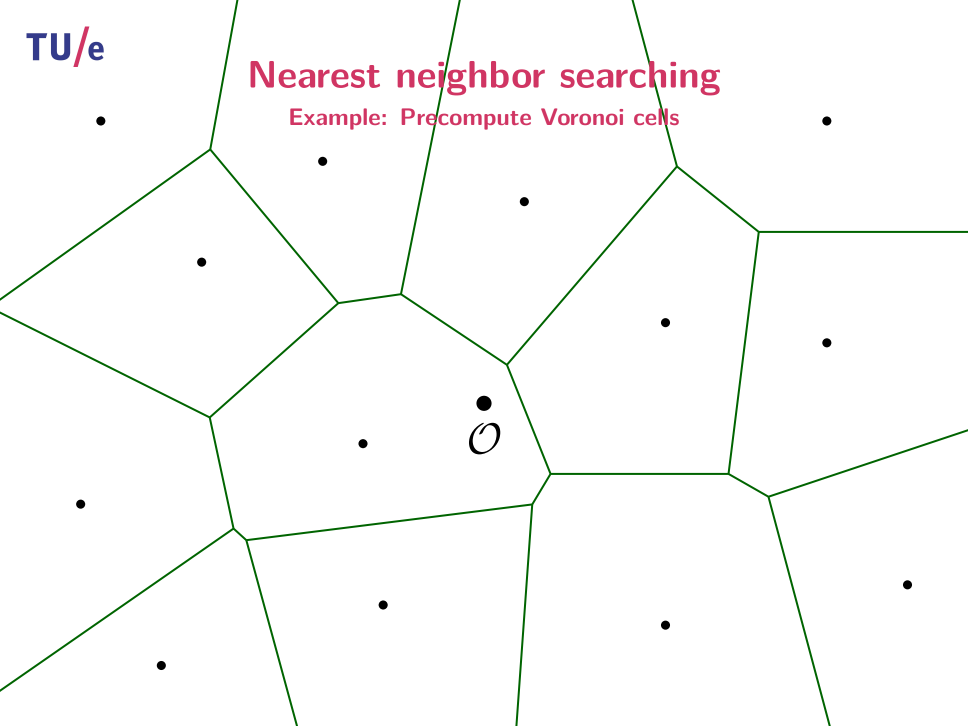
Nearest neighbor searching

Example: Precompute Voronoi cells



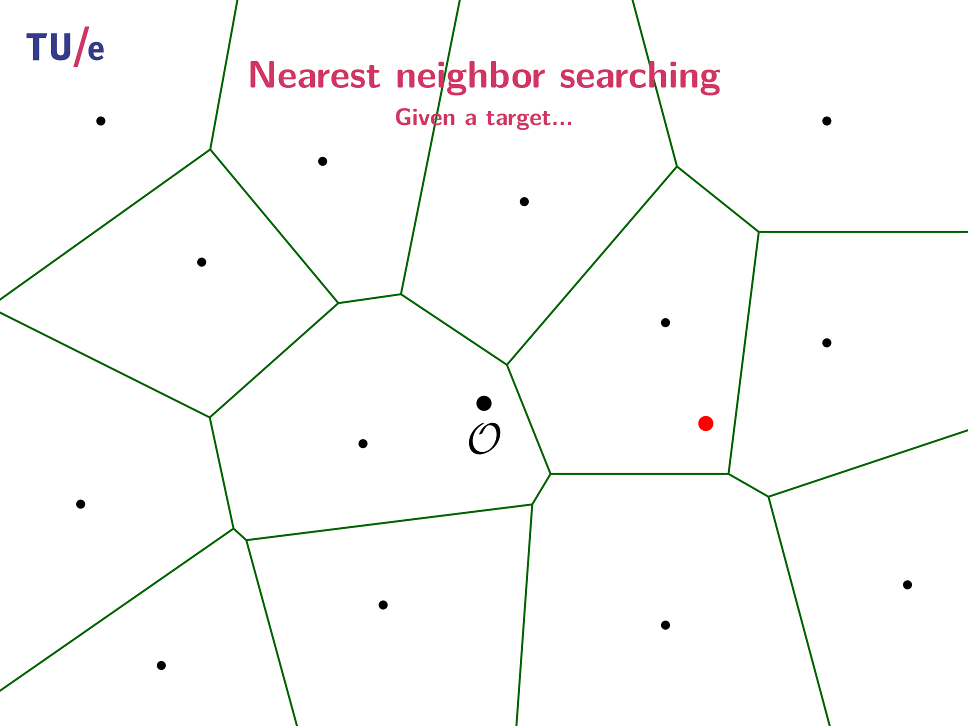
Nearest neighbor searching

Example: Precompute Voronoi cells



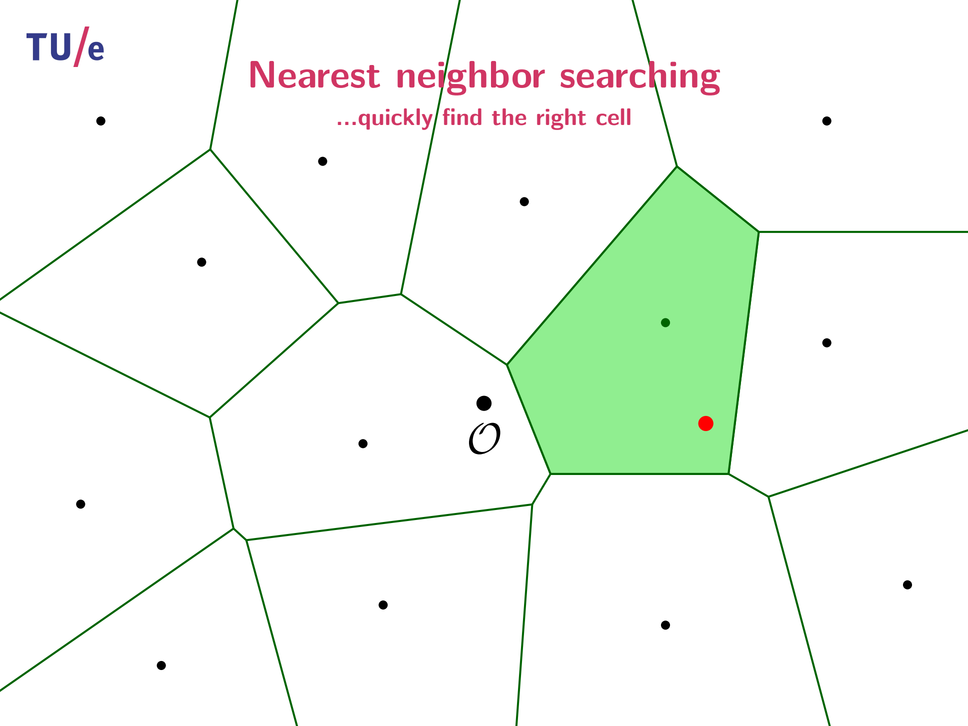
Nearest neighbor searching

Given a target...



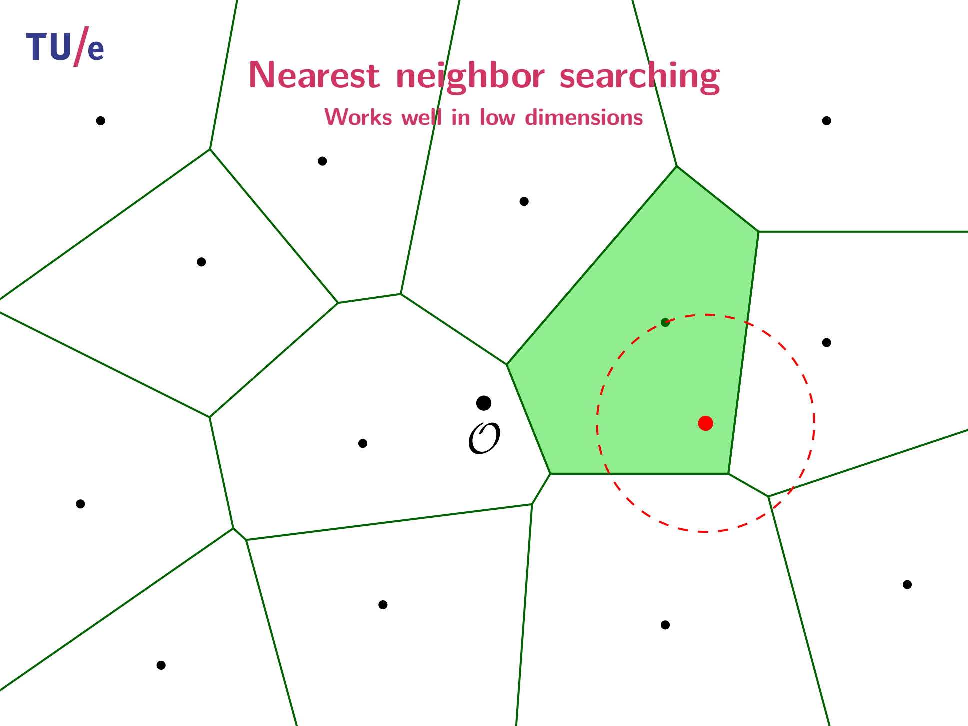
Nearest neighbor searching

...quickly find the right cell



Nearest neighbor searching

Works well in low dimensions



Nearest neighbor searching

Problem setting

- High dimensions d

Nearest neighbor searching

Problem setting

- High dimensions d
- Large data set of size $n = 2^{\Omega(d / \log d)}$

Nearest neighbor searching

Problem setting

- High dimensions d
- Large data set of size $n = 2^{\Omega(d/\log d)}$
- Assumption: Data set lies on the sphere
  - Eucl./angular NNS in $\mathbb{R}^d$ reduce to Eucl. NNS on the sphere [AR'15]

Nearest neighbor searching

Problem setting

- High dimensions d
- Large data set of size $n = 2^{\Omega(d/\log d)}$
- Assumption: Data set lies on the sphere
  - Eucl./angular NNS in $\mathbb{R}^d$ reduce to Eucl. NNS on the sphere [AR'15]
- "Random" case: $c \cdot r = \sqrt{2}$

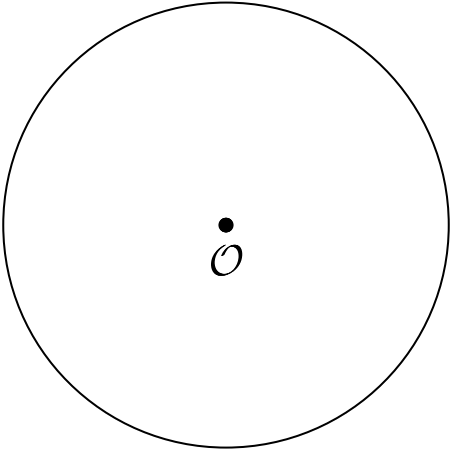
Nearest neighbor searching

Problem setting

- High dimensions d
- Large data set of size $n = 2^{\Omega(d/\log d)}$
- Assumption: Data set lies on the sphere
  - Eucl./angular NNS in $\mathbb{R}^d$ reduce to Eucl. NNS on the sphere [AR'15]
- "Random" case: $c \cdot r = \sqrt{2}$
- Goal: Query time $O(n^\rho)$ with $\rho < 1$

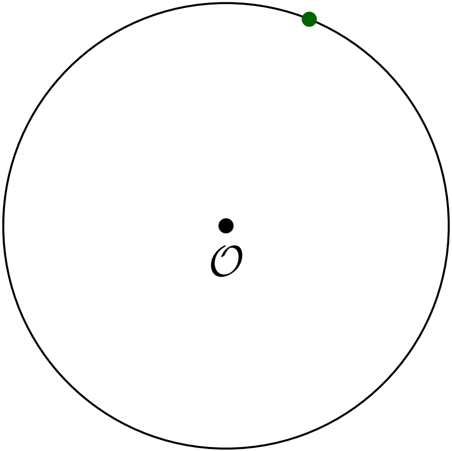
Hyperplane LSH

[Charikar, STOC'02]



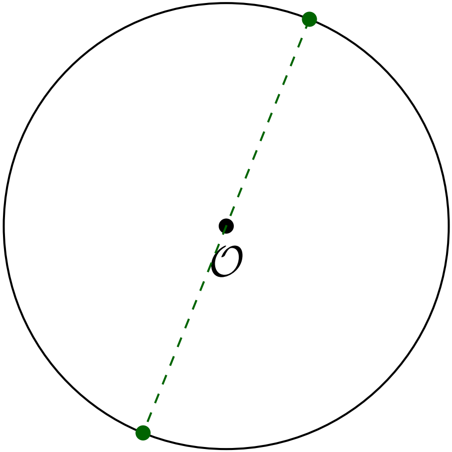
Hyperplane LSH

Random point



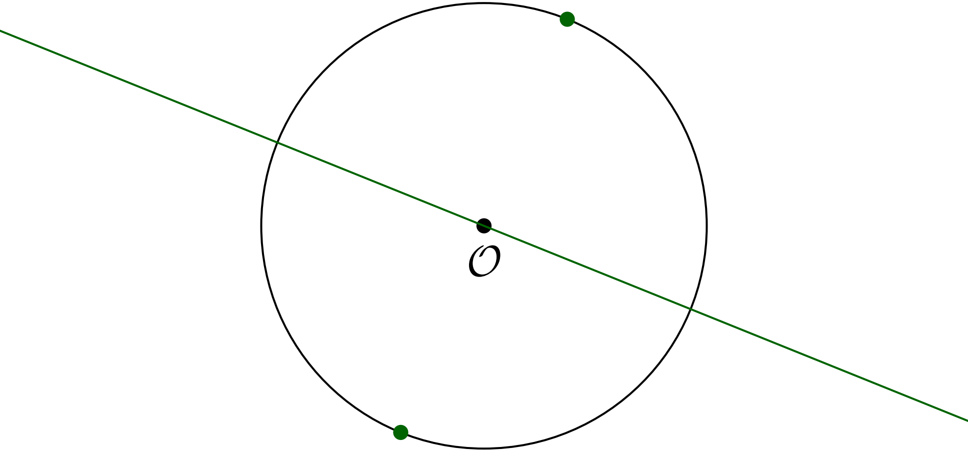
Hyperplane LSH

Opposite point



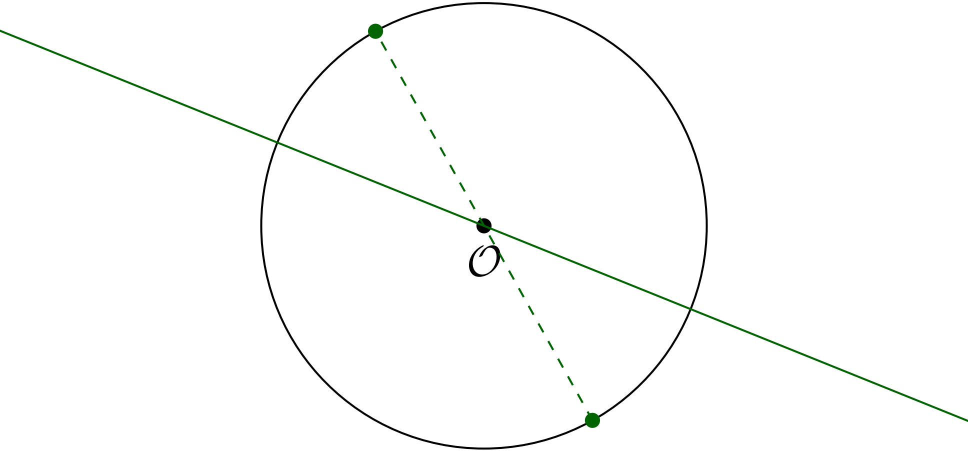
Hyperplane LSH

Two Voronoi cells



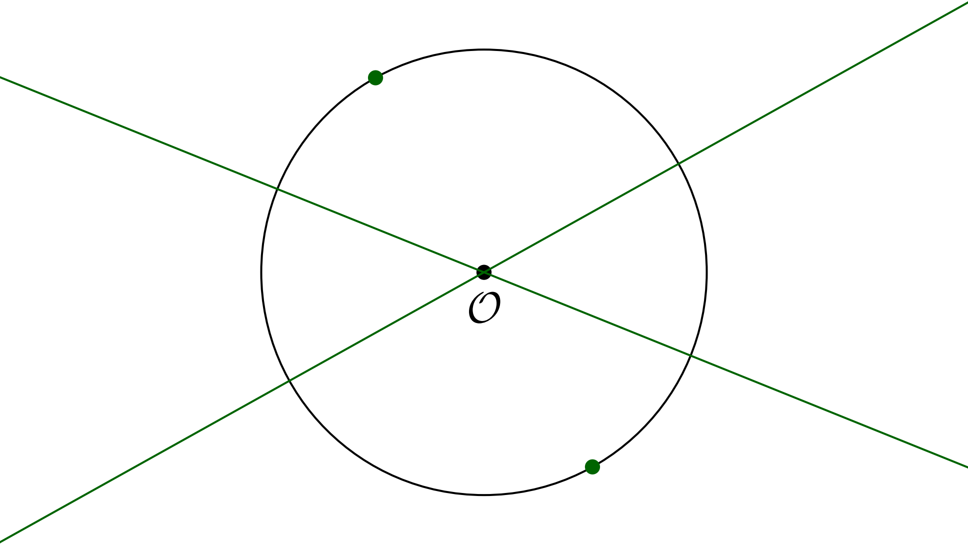
Hyperplane LSH

Another pair of points



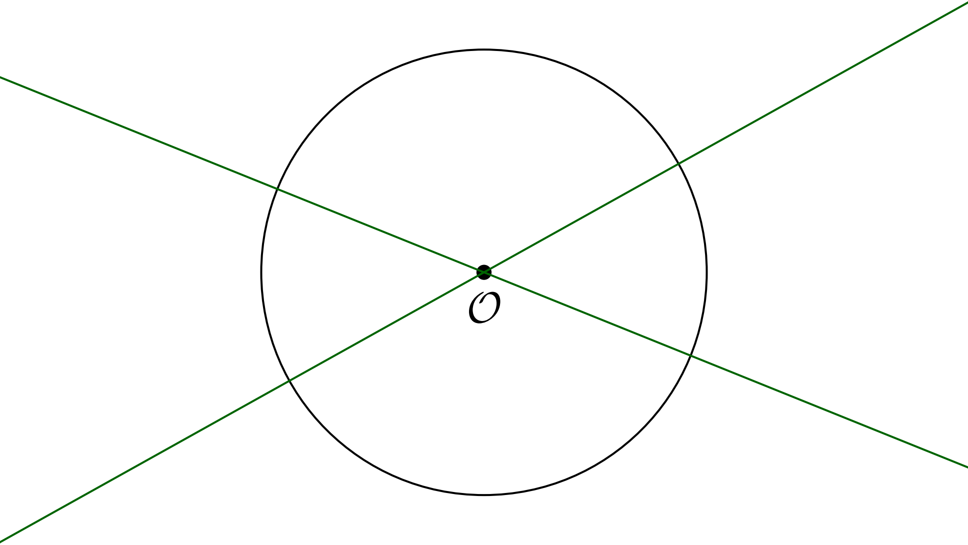
Hyperplane LSH

Another hyperplane



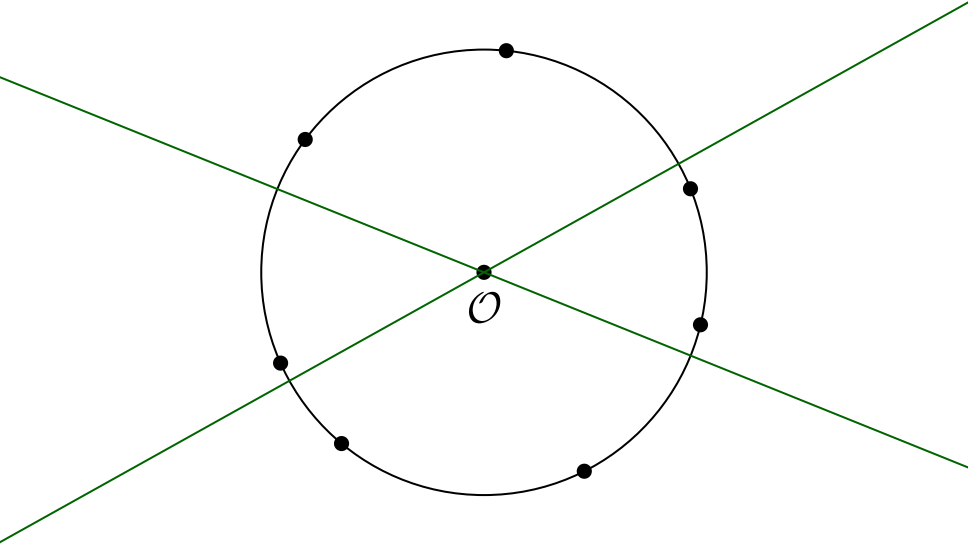
Hyperplane LSH

Defines partition



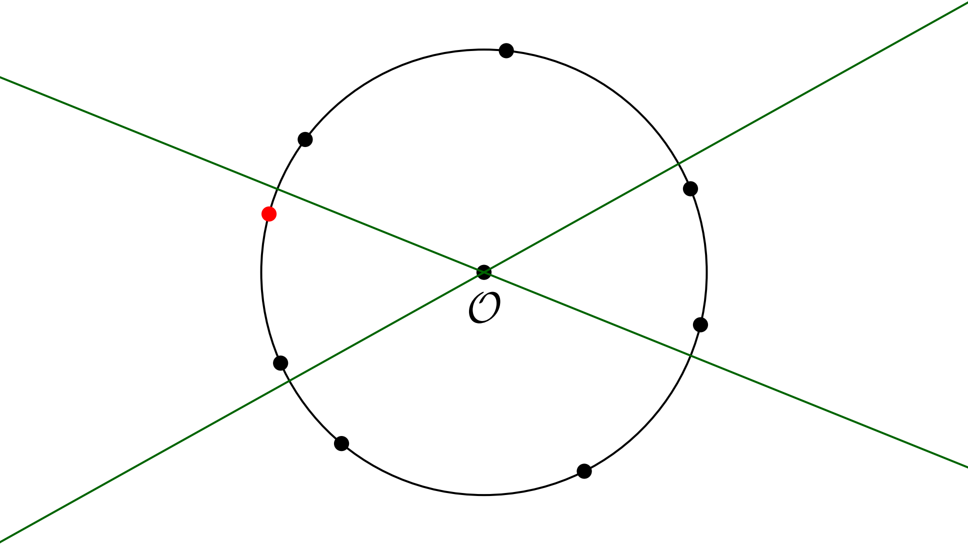
Hyperplane LSH

Preprocessing



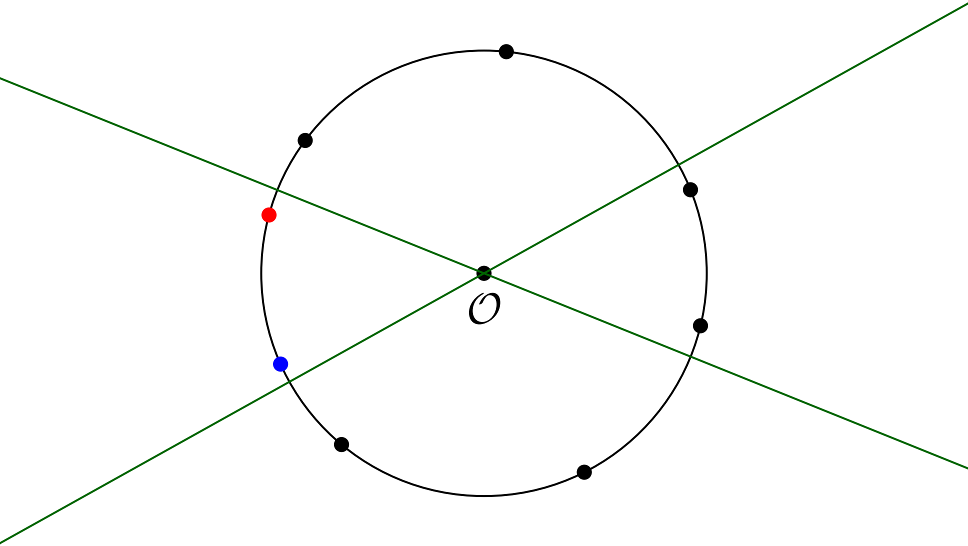
Hyperplane LSH

Query



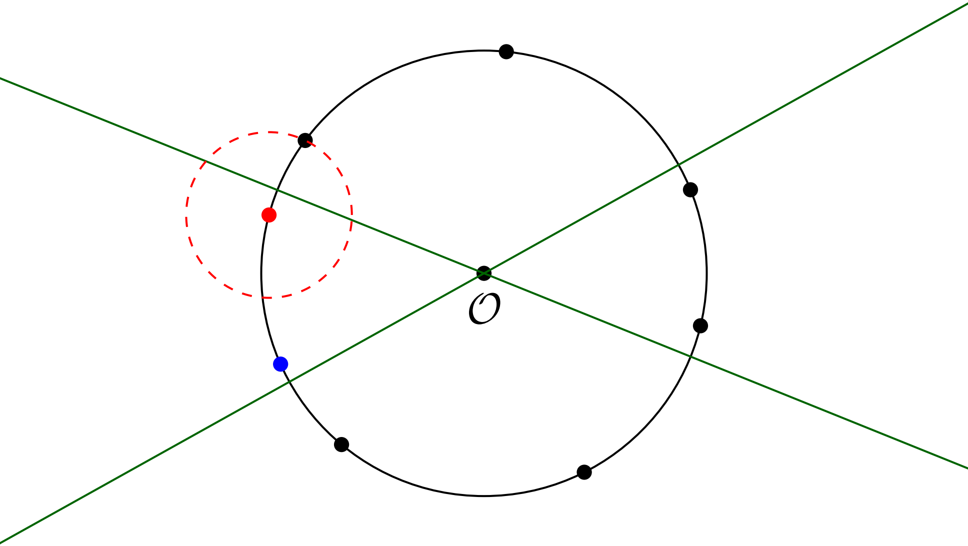
Hyperplane LSH

Collisions



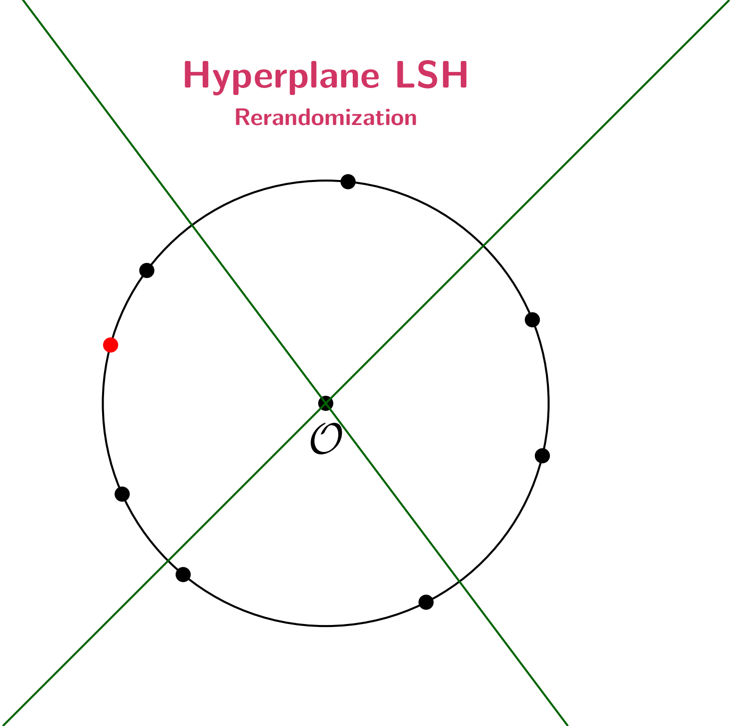
Hyperplane LSH

Failure



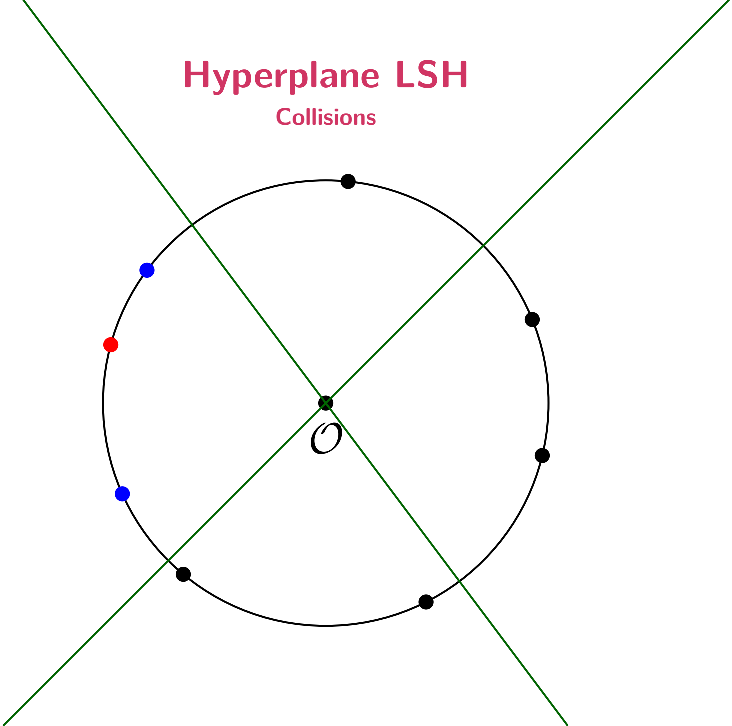
Hyperplane LSH

Rerandomization



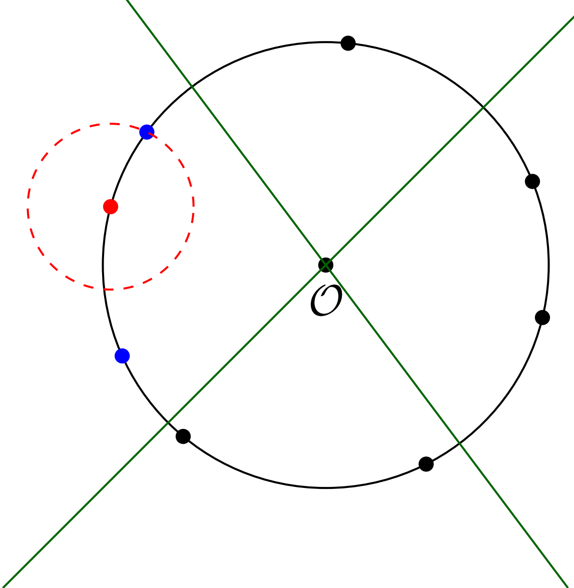
Hyperplane LSH

Collisions



Hyperplane LSH

Success



Hyperplane LSH

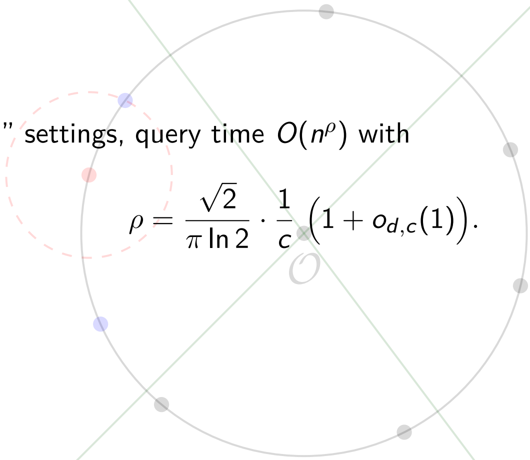
Overview

Hyperplane LSH

Overview

For "random" settings, query time $O(n^\rho)$ with

$$\rho = \frac{\sqrt{2}}{\pi \ln 2} \cdot \frac{1}{c} \Big(1 + o_{d,c}(1)\Big).$$



Hyperplane LSH

Overview

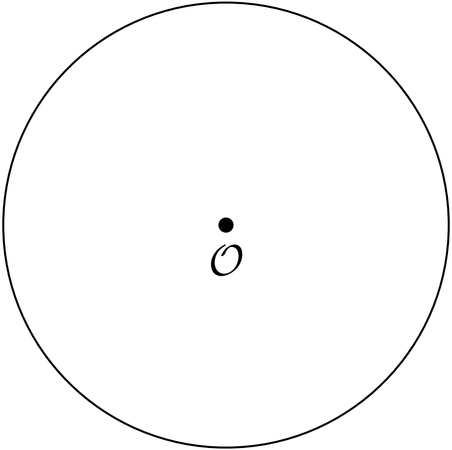
For "random" settings, query time $O(n^{\rho})$ with

$$\rho = \frac{\sqrt{2}}{\pi \ln 2} \cdot \frac{1}{c} \Big(1 + o_{d,c}(1)\Big).$$

Efficient but suboptimal as $\rho \propto \frac{1}{c^2}$ is achievable

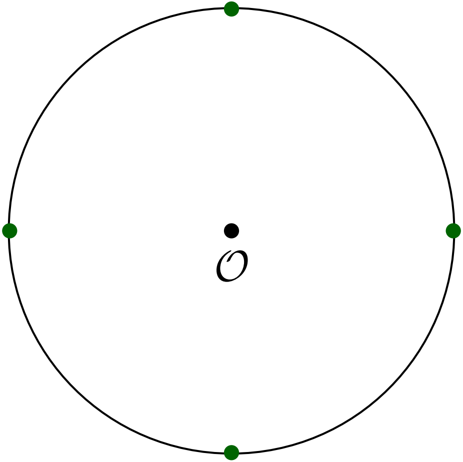
Cross-Polytope LSH

[Terasawa–Tanaka, WADS'07]
[Andoni et al., NIPS'15]



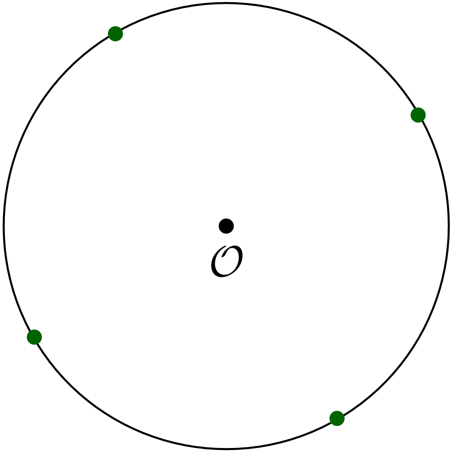
Cross-Polytope LSH

Vertices of cross-polytope (simplex)



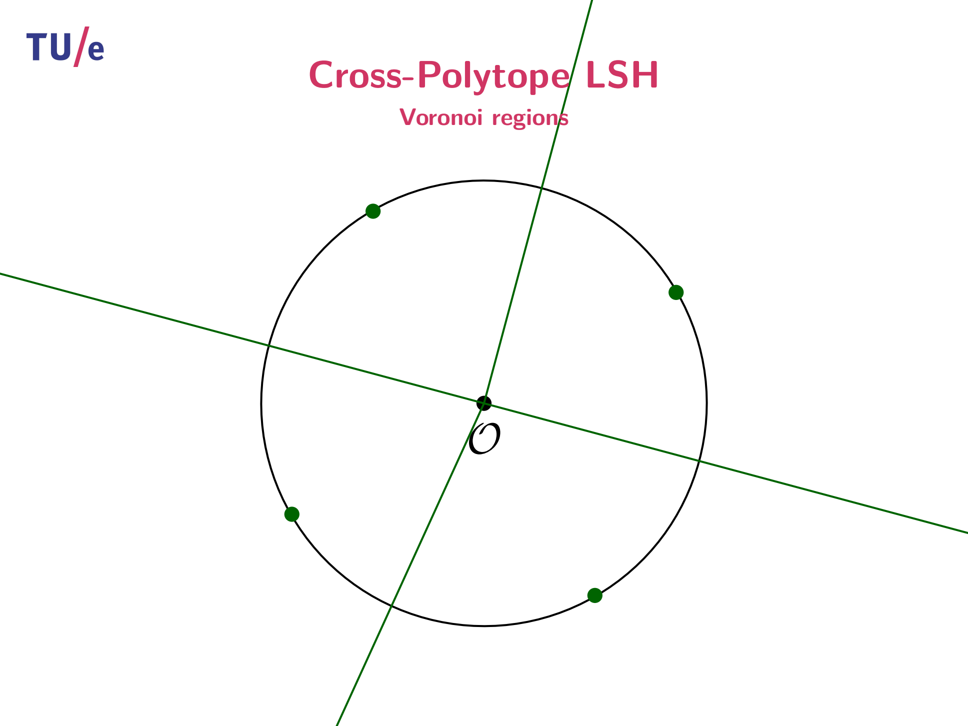
Cross-Polytope LSH

Random rotation



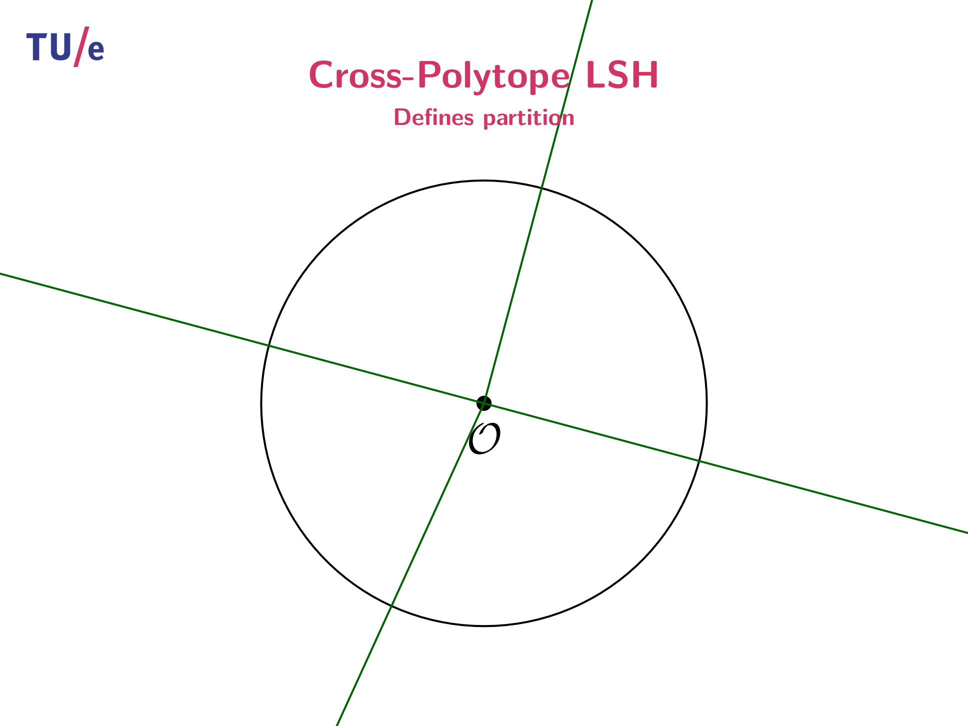
Cross-Polytope LSH

Voronoi regions



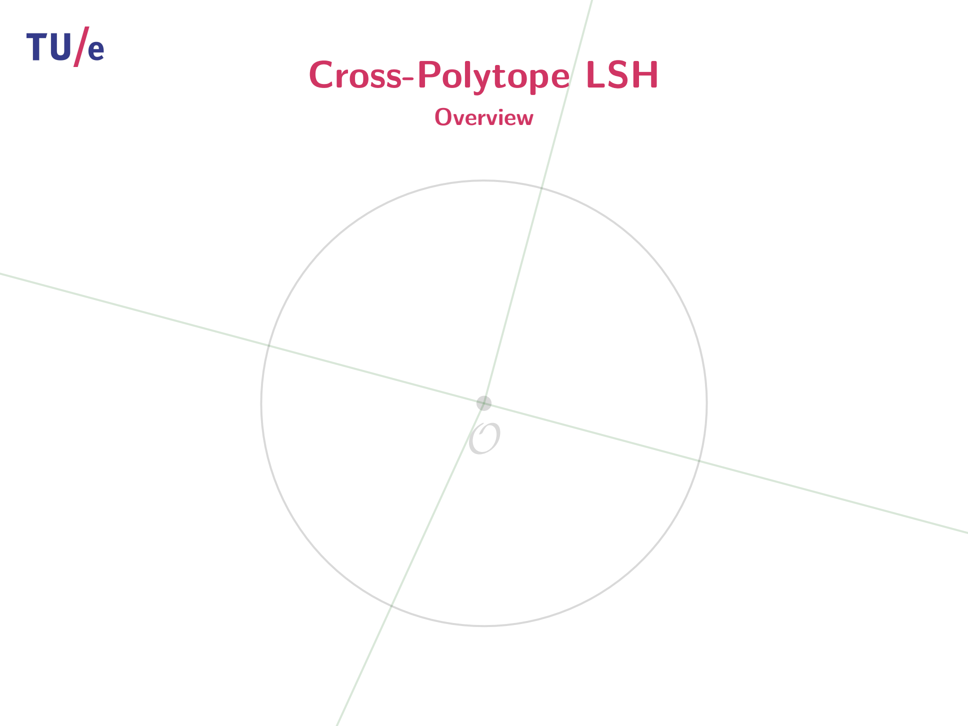
Cross-Polytope LSH

Defines partition



Cross-Polytope LSH

Overview



Cross-Polytope LSH

Overview

For “random” settings, query time $O(n^\rho)$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big)$$

Cross-Polytope LSH

Overview

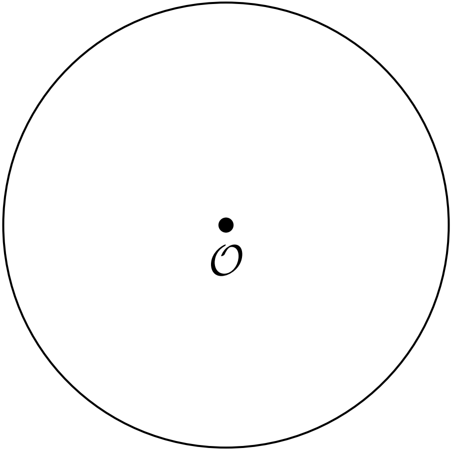
For “random” settings, query time $O(n^\rho)$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big)$$

Essentially optimal for large c and $n = 2^{o(d)}$ [Dub’10, AR’15]

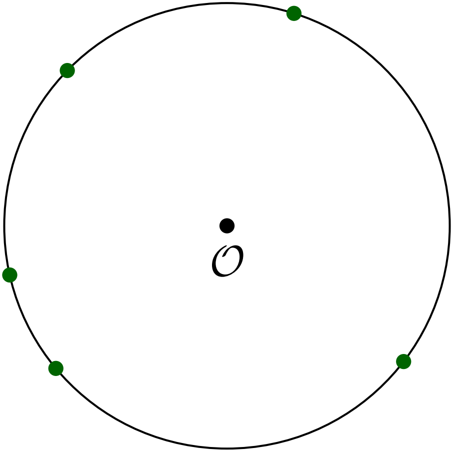
Spherical/Voronoi LSH

[Andoni et al., SODA'14]
[Andoni–Razenshteyn, STOC'15]



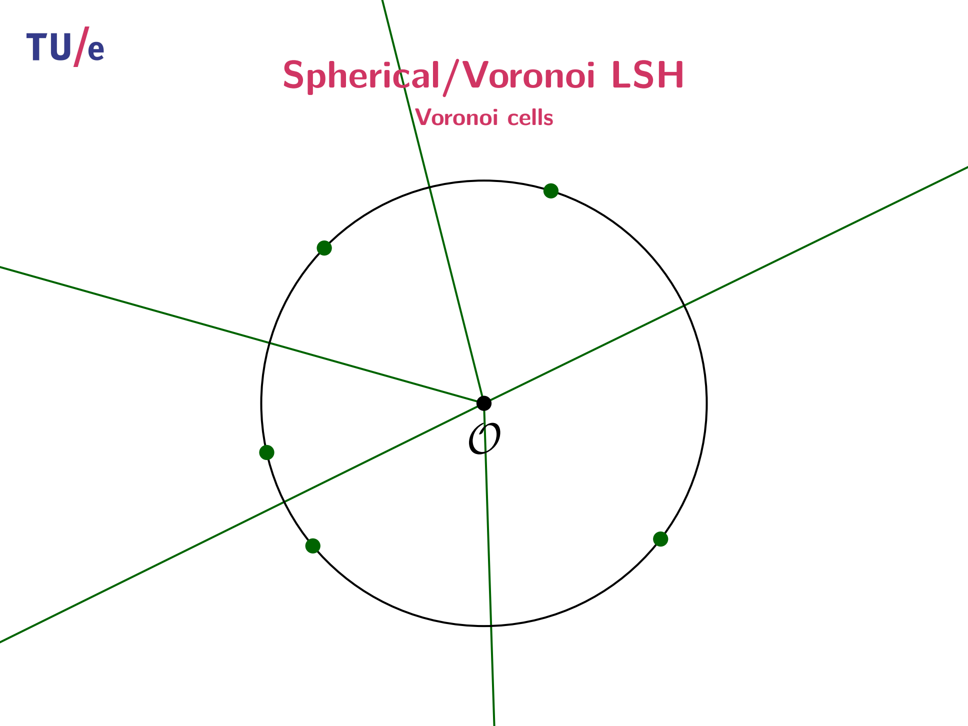
Spherical/Voronoi LSH

Random points



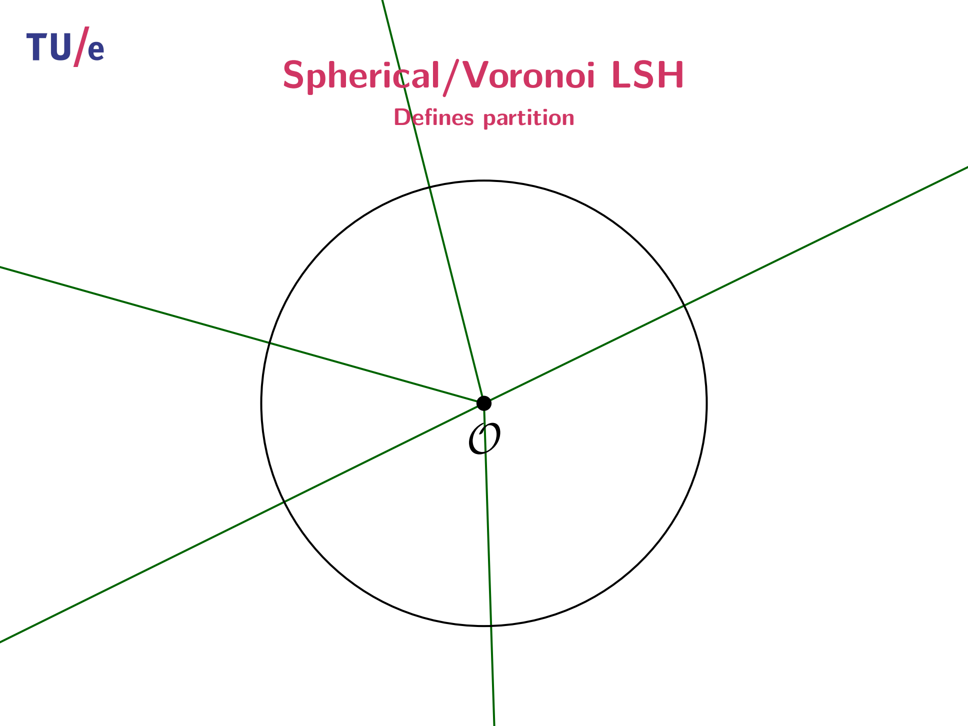
Spherical/Voronoi LSH

Voronoi cells



Spherical/Voronoi LSH

Defines partition



Spherical/Voronoi LSH

Overview

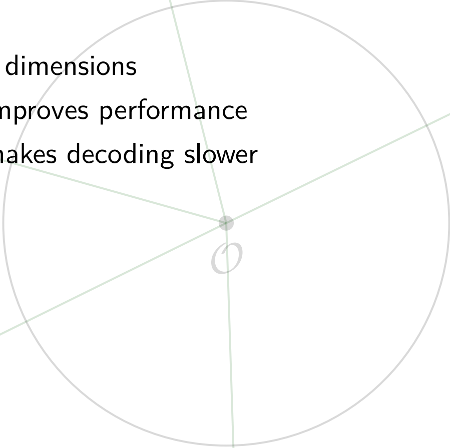
TU/e

Spherical/Voronoi LSH

Overview

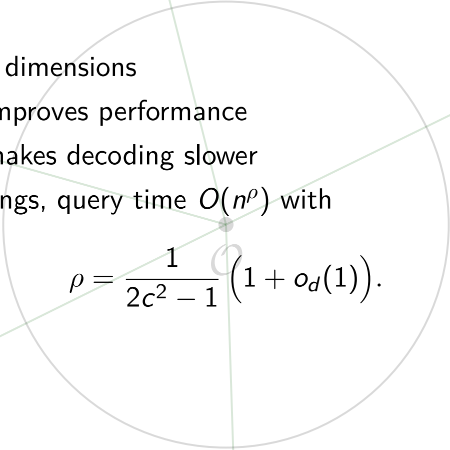
$2^{O(\sqrt{d})}$ points in d dimensions

- More points improves performance
- More points makes decoding slower



Spherical/Voronoi LSH

Overview



$2^{O(\sqrt{d})}$ points in d dimensions

- More points improves performance
- More points makes decoding slower

For "random" settings, query time $O(n^\rho)$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big).$$

TU/e

Spherical/Voronoi LSH

Overview

$2^{O(\sqrt{d})}$ points in d dimensions

- More points improves performance
- More points makes decoding slower

For "random" settings, query time $O(n^{\rho})$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big).$$

Essentially optimal for large c and $n = 2^{o(d)}$

TU/e

LSH overview

- Hyperplane LSH: 2 Voronoi cells
  - Efficient decoding
  - Suboptimal for large d, c
- Cross-Polytope LSH: $2d$ Voronoi cells
  - Reasonably efficient decoding
  - Optimal for large c and $n = 2^{o(d)}$
- Spherical/Voronoi LSH: $2^{O(\sqrt{d})}$ Voronoi cells
  - Slow decoding
  - Optimal for large c and $n = 2^{o(d)}$

LSH overview

- Hyperplane LSH: 2 Voronoi cells
  - Efficient decoding
  - Suboptimal for large d, c
- Cross-Polytope LSH: $2d$ Voronoi cells
  - Reasonably efficient decoding
  - Optimal for large c and $n = 2^{o(d)}$
- Spherical/Voronoi LSH: $2^{O(\sqrt{d})}$ Voronoi cells
  - Slow decoding
  - Optimal for large c and $n = 2^{o(d)}$

1. Can we use even more Voronoi cells?

LSH overview

- Hyperplane LSH: 2 Voronoi cells
  - Efficient decoding
  - Suboptimal for large d, c
- Cross-Polytope LSH: $2d$ Voronoi cells
  - Reasonably efficient decoding
  - Optimal for large c and $n = 2^{o(d)}$
- Spherical/Voronoi LSH: $2^{O(\sqrt{d})}$ Voronoi cells
  - Slow decoding
  - Optimal for large c and $n = 2^{o(d)}$

1. Can we use even more Voronoi cells?
2. Can decoding be made faster?

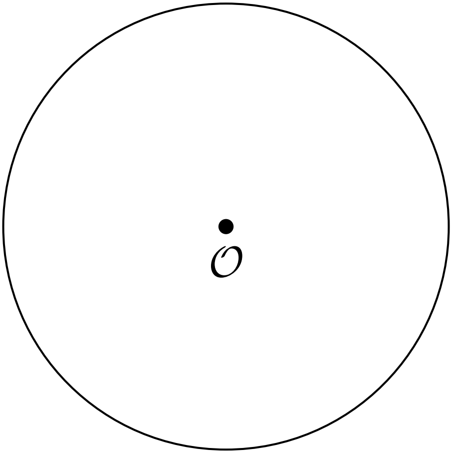
LSH overview

- Hyperplane LSH: 2 Voronoi cells
  - Efficient decoding
  - Suboptimal for large d, c
- Cross-Polytope LSH: $2d$ Voronoi cells
  - Reasonably efficient decoding
  - Optimal for large c and $n = 2^{o(d)}$
- Spherical/Voronoi LSH: $2^{O(\sqrt{d})}$ Voronoi cells
  - Slow decoding
  - Optimal for large c and $n = 2^{o(d)}$

1. Can we use even more Voronoi cells?
2. Can decoding be made faster?
3. What about $n = 2^{\Omega(d)}$?

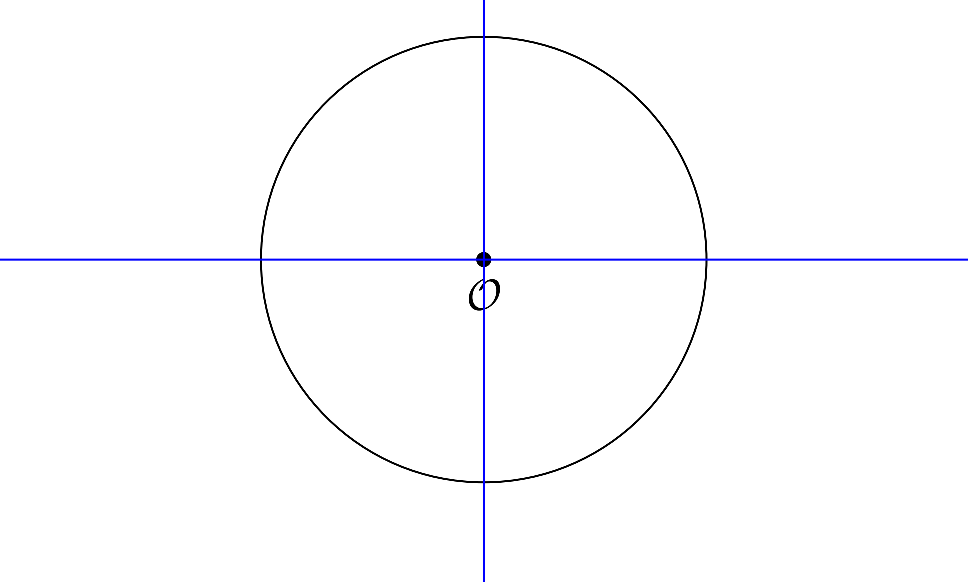
Contribution

Overview



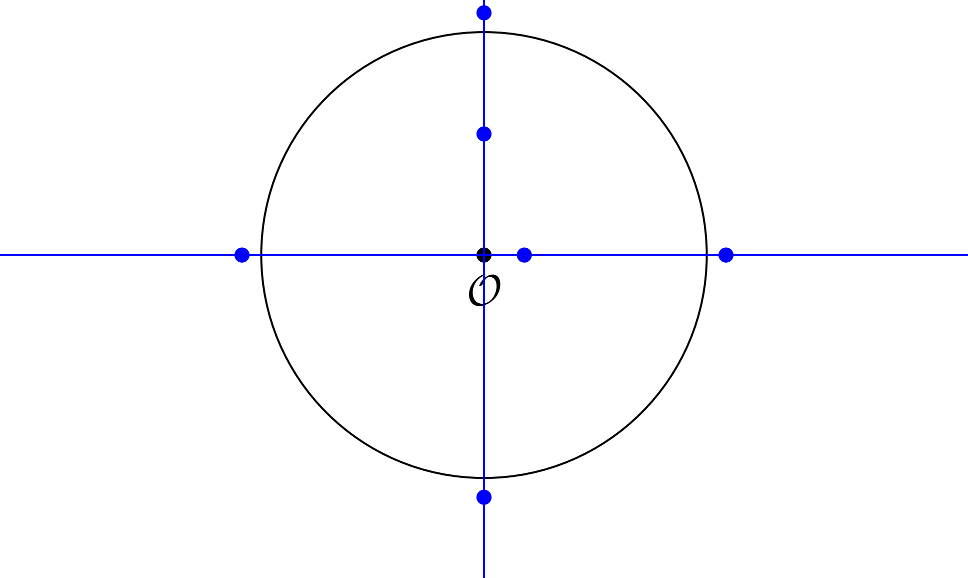
Contribution

Partition dimensions into blocks



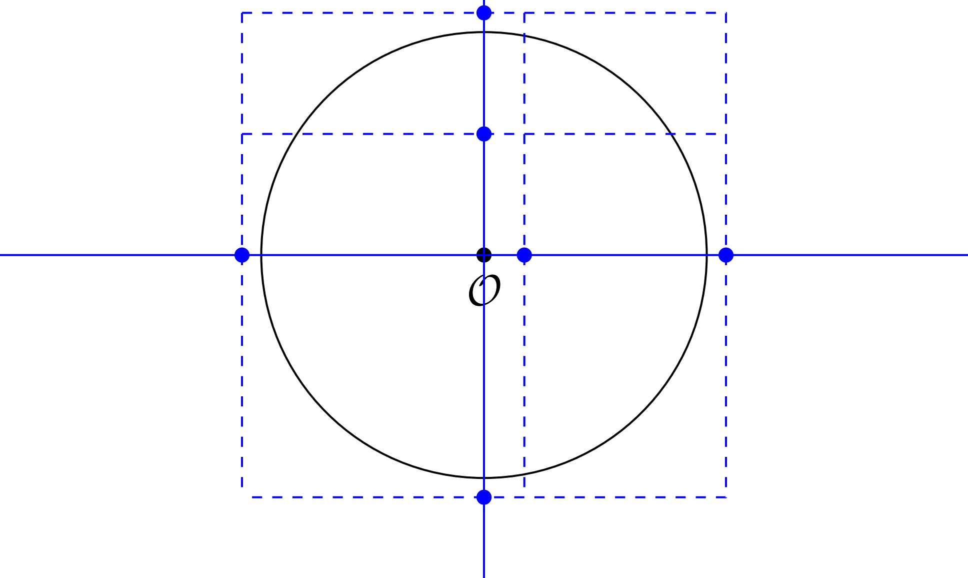
Contribution

Random subcodes



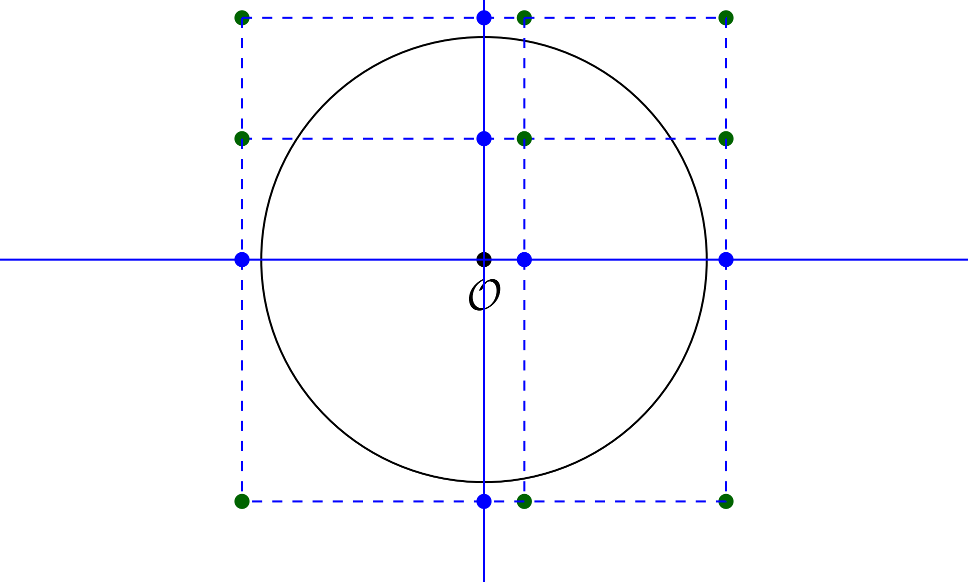
Contribution

Construct concatenated code



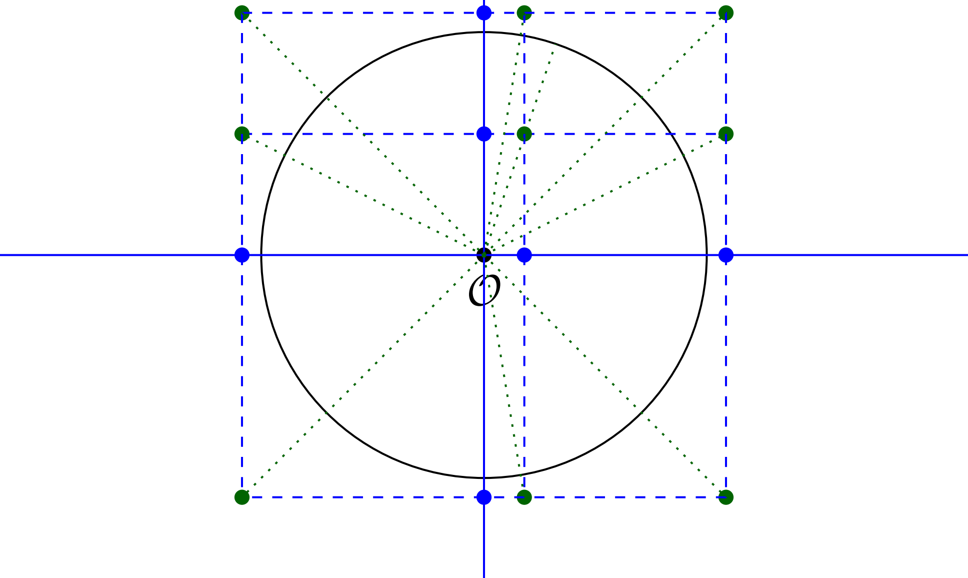
Contribution

Construct concatenated code



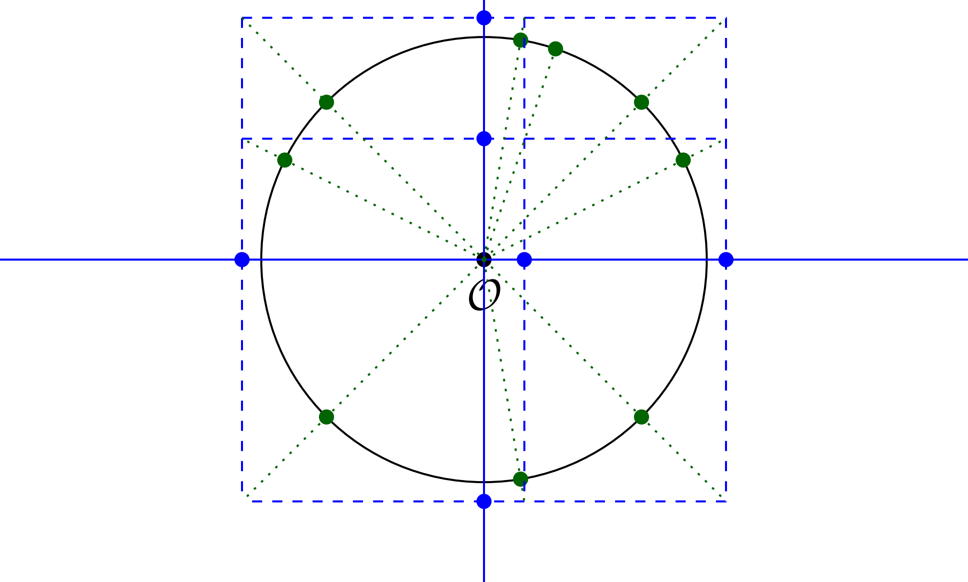
Contribution

Normalize (only for example)



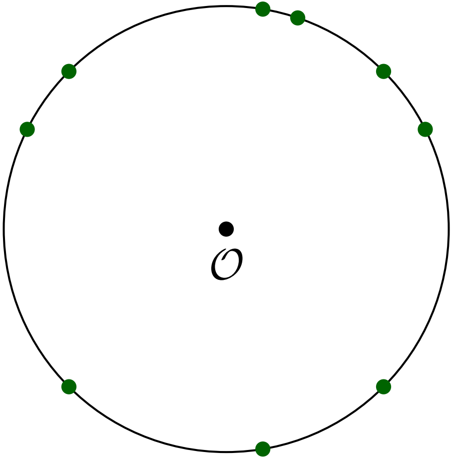
Contribution

Normalize (only for example)



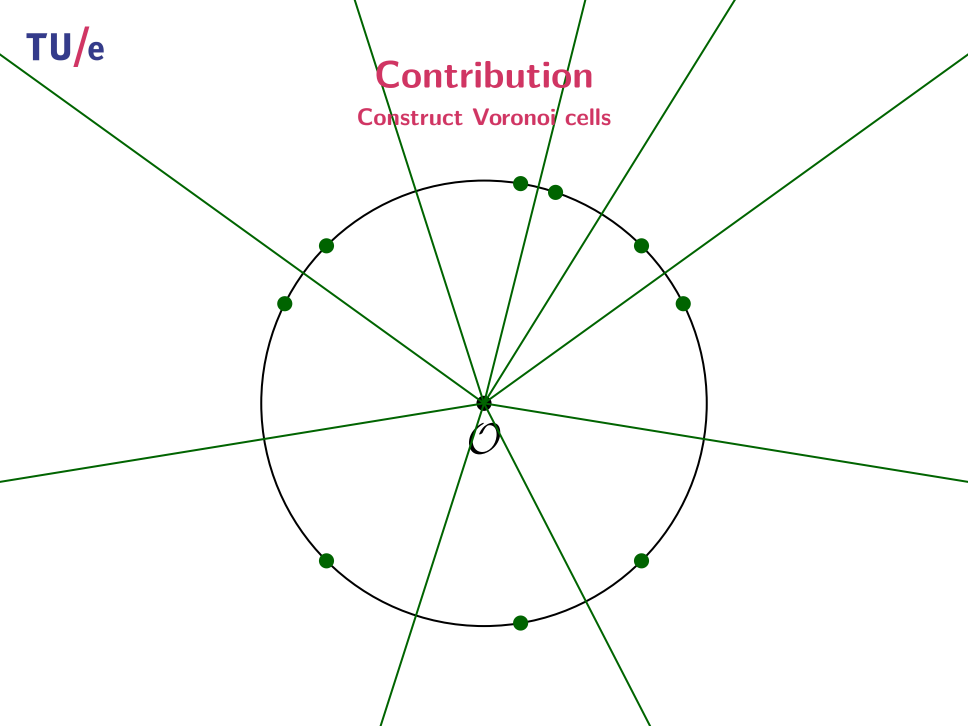
Contribution

Normalize (only for example)



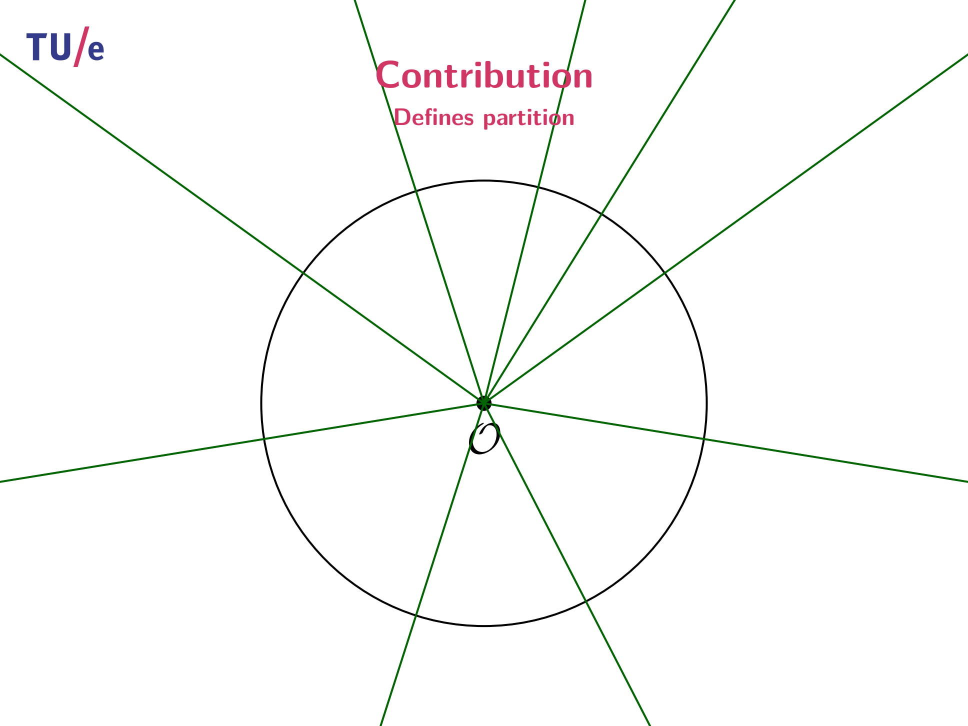
Contribution

Construct Voronoi cells



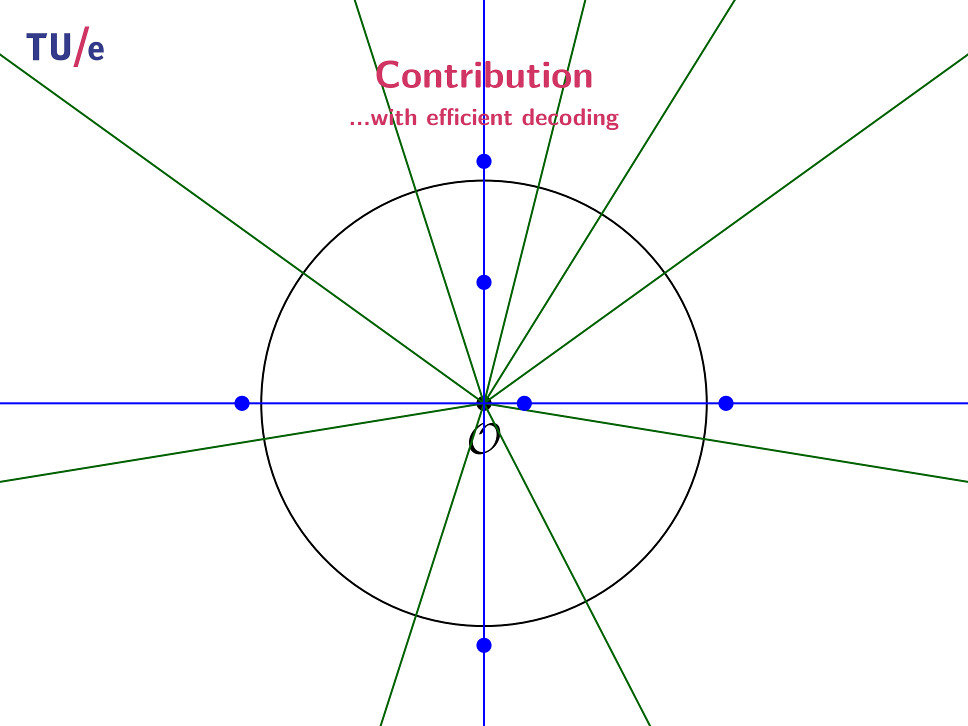
Contribution

Defines partition



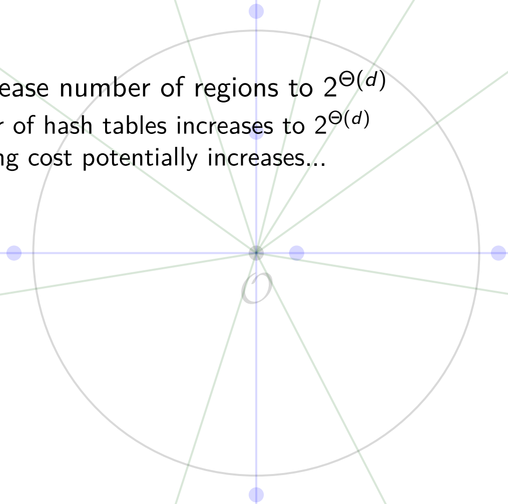
Contribution

...with efficient decoding



Techniques

- Idea 1: Increase number of regions to $2^{\Theta(d)}$
  - Number of hash tables increases to $2^{\Theta(d)}$
  - Decoding cost potentially increases...



Techniques

- Idea 1: Increase number of regions to $2^{\Theta(d)}$
  - Number of hash tables increases to $2^{\Theta(d)}$
  - Decoding cost potentially increases...
- Idea 2: Use structured codes for random regions
  - Spherical/Voronoi LSH with dependent random points
  - Allows for efficient list-decoding

Techniques

- Idea 1: Increase number of regions to $2^{\Theta(d)}$
  - Number of hash tables increases to $2^{\Theta(d)}$
  - Decoding cost potentially increases...
- Idea 2: Use structured codes for random regions
  - Spherical/Voronoi LSH with dependent random points
  - Allows for efficient list-decoding
- Idea 3: Replace partitions with filters
  - Relaxation: filters need not partition the space
  - Might not be needed to achieve improvement

Results

For random sparse settings ($n = 2^{o(d)}$), query time $O(n^\rho)$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big).$$

Results

For random sparse settings ($n = 2^{o(d)}$), query time $O(n^{\rho})$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big).$$

For random dense settings ($n = 2^{\kappa d}$ with small κ), we obtain

$$\rho = \frac{1 - \kappa}{2c^2 - 1}\Big(1 + o_{d,\kappa}(1)\Big).$$

Results

For random sparse settings ($n = 2^{o(d)}$), query time $O(n^\rho)$ with

$$\rho = \frac{1}{2c^2 - 1}\Big(1 + o_d(1)\Big).$$

For random dense settings ($n = 2^{\kappa d}$ with small κ), we obtain

$$\rho = \frac{1 - \kappa}{2c^2 - 1}\Big(1 + o_{d,\kappa}(1)\Big).$$

For random dense settings ($n = 2^{\kappa d}$ with large κ), we obtain

$$\rho = \frac{-1}{2\kappa}\log\left(1 - \frac{1}{2c^2 - 1}\right)\Big(1 + o_d(1)\Big).$$

Conclusions

Main result: Use even more regions using list-decodable codes

- For $n = 2^{o(d)}$, non-asymptotic improvement
- For $n = 2^{\Theta(d)}$, asymptotic improvement
- Corollary: Lower bounds for $n = 2^{o(d)}$ do not hold for $n = 2^{\Theta(d)}$

Conclusions

Main result: Use even more regions using list-decodable codes

- For $n = 2^{o(d)}$, non-asymptotic improvement
- For $n = 2^{\Theta(d)}$, asymptotic improvement
- Corollary: Lower bounds for $n = 2^{o(d)}$ do not hold for $n = 2^{\Theta(d)}$

Concrete improvements in cryptanalysis

- Reduce exponent for lattice sieving algorithms
- Reduce exponent for decoding binary codes [MO'15]

Conclusions

Main result: Use even more regions using list-decodable codes

- For $n = 2^{o(d)}$, non-asymptotic improvement
- For $n = 2^{\Theta(d)}$, asymptotic improvement
- Corollary: Lower bounds for $n = 2^{o(d)}$ do not hold for $n = 2^{\Theta(d)}$

Concrete improvements in cryptanalysis

- Reduce exponent for lattice sieving algorithms
- Reduce exponent for decoding binary codes [MO'15]

Questions?