Similarity Search

Largely based on a paper joint with

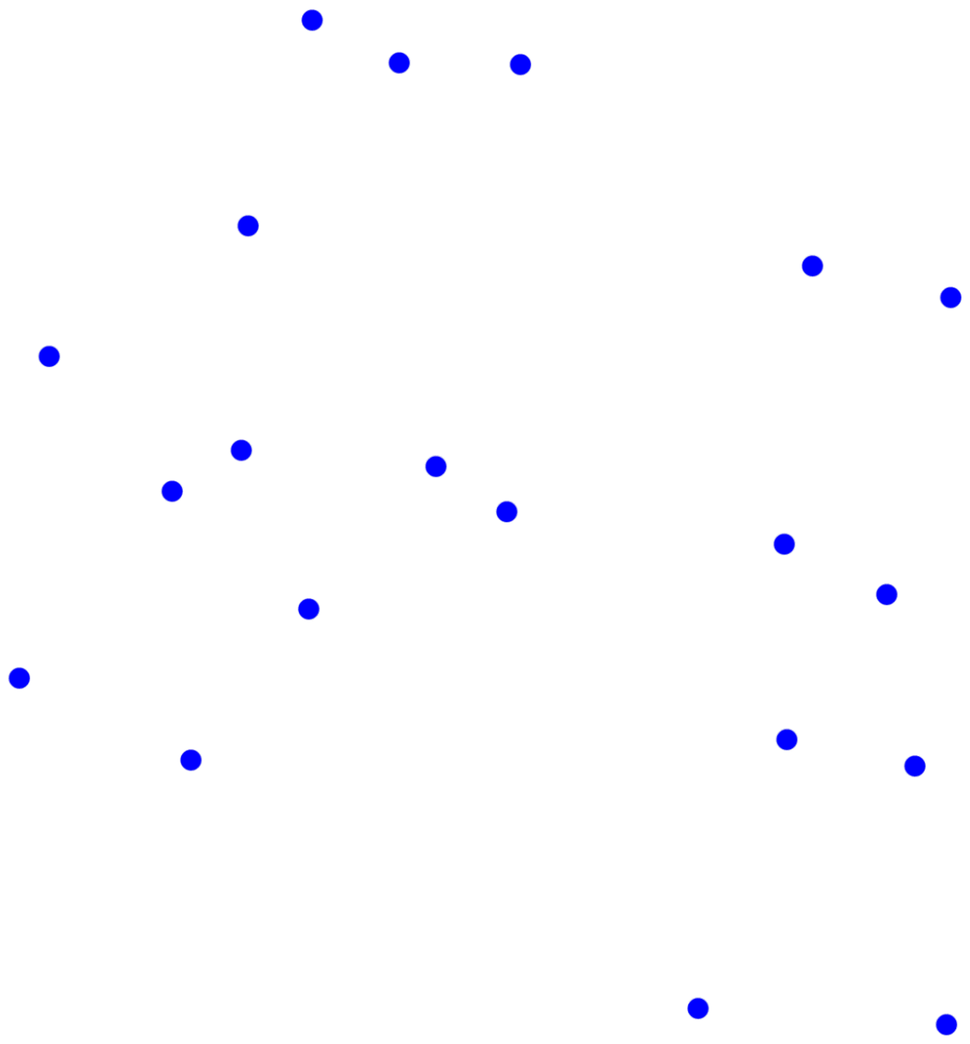
Alexandr Andoni (Columbia), Piotr Indyk (MIT),

Thijs Laarhoven (TU Eindhoven) and Ludwig Schmidt (MIT)

Near Neighbor Search

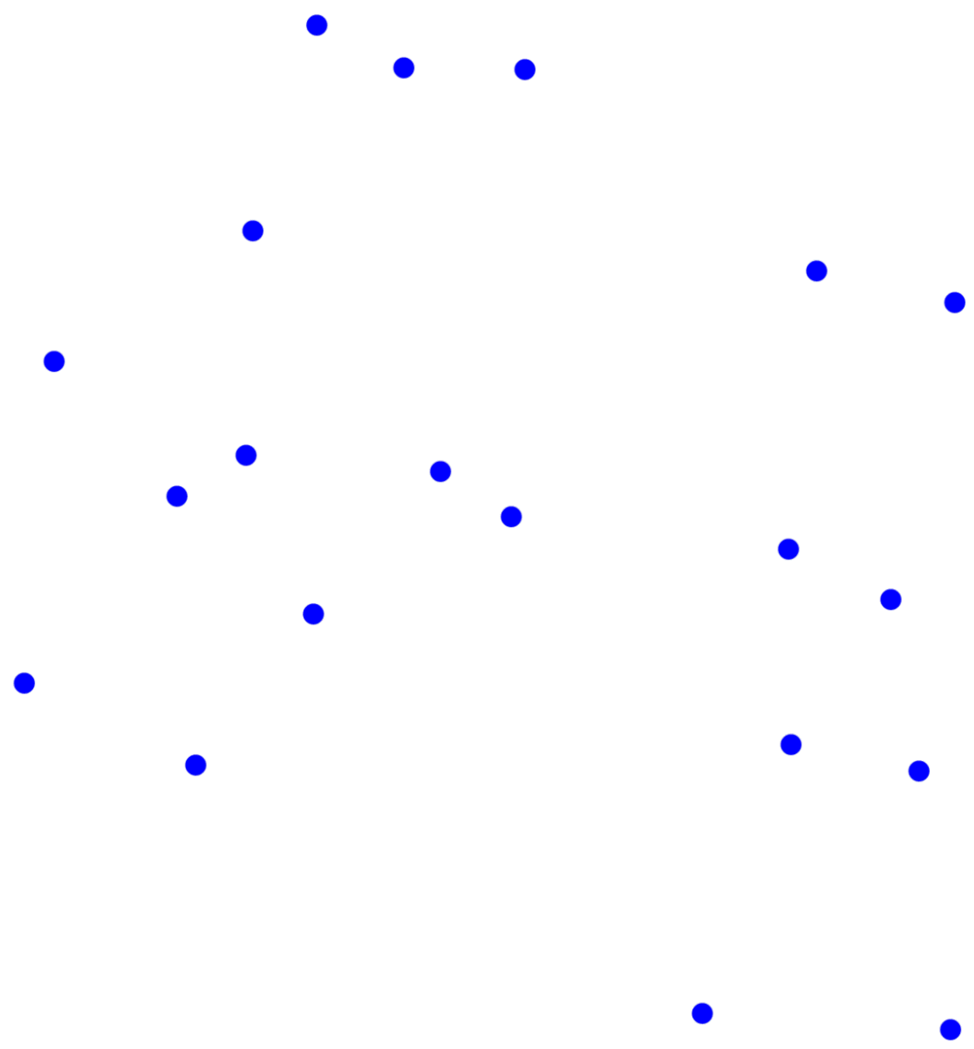
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, $\mathbf{r > 0}$



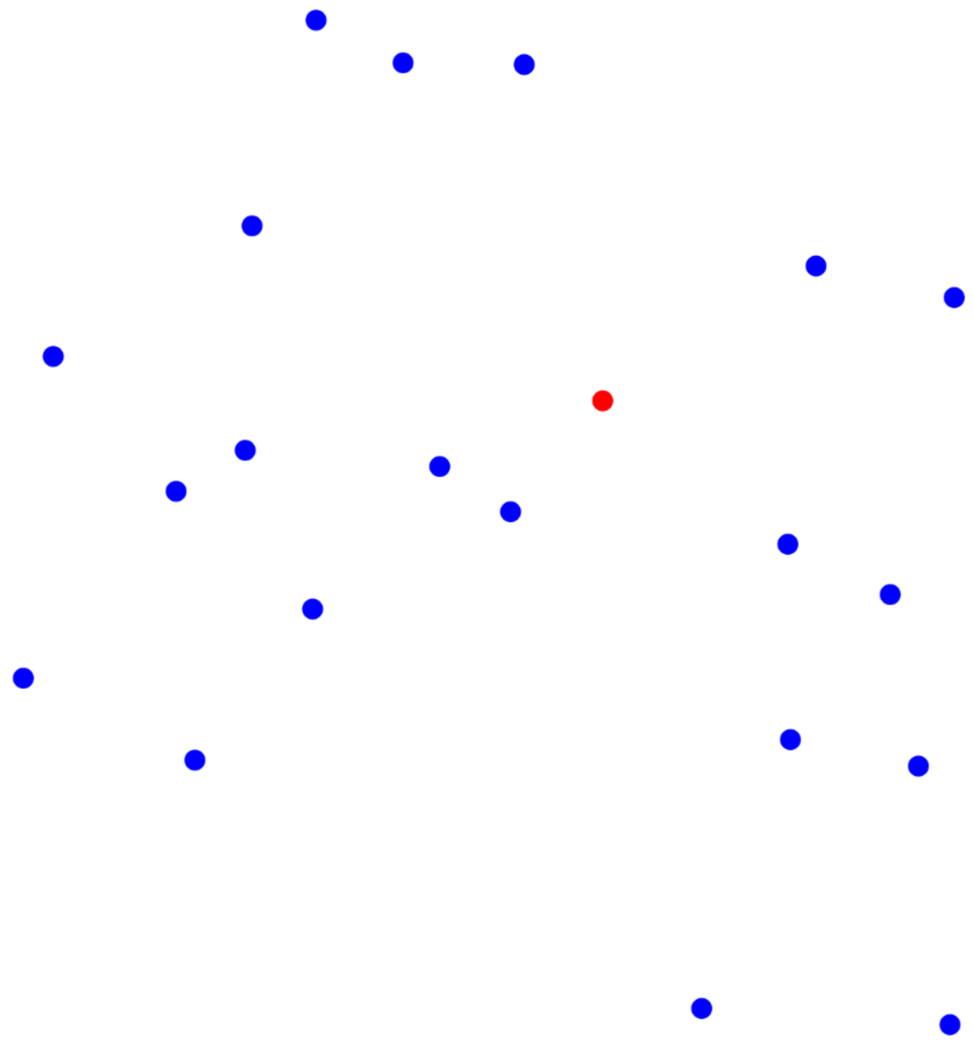
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, $\mathbf{r > 0}$
- **Goal:** a data point within **r** from a query



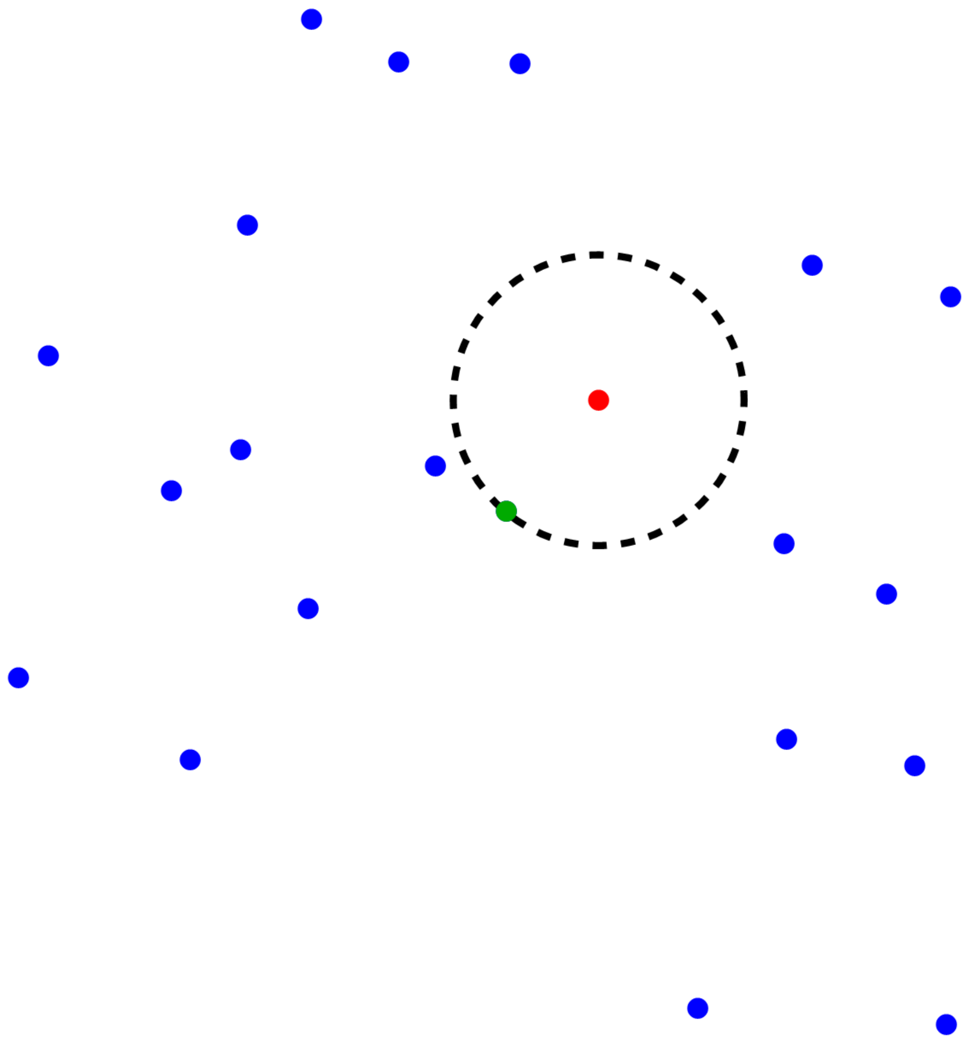
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query



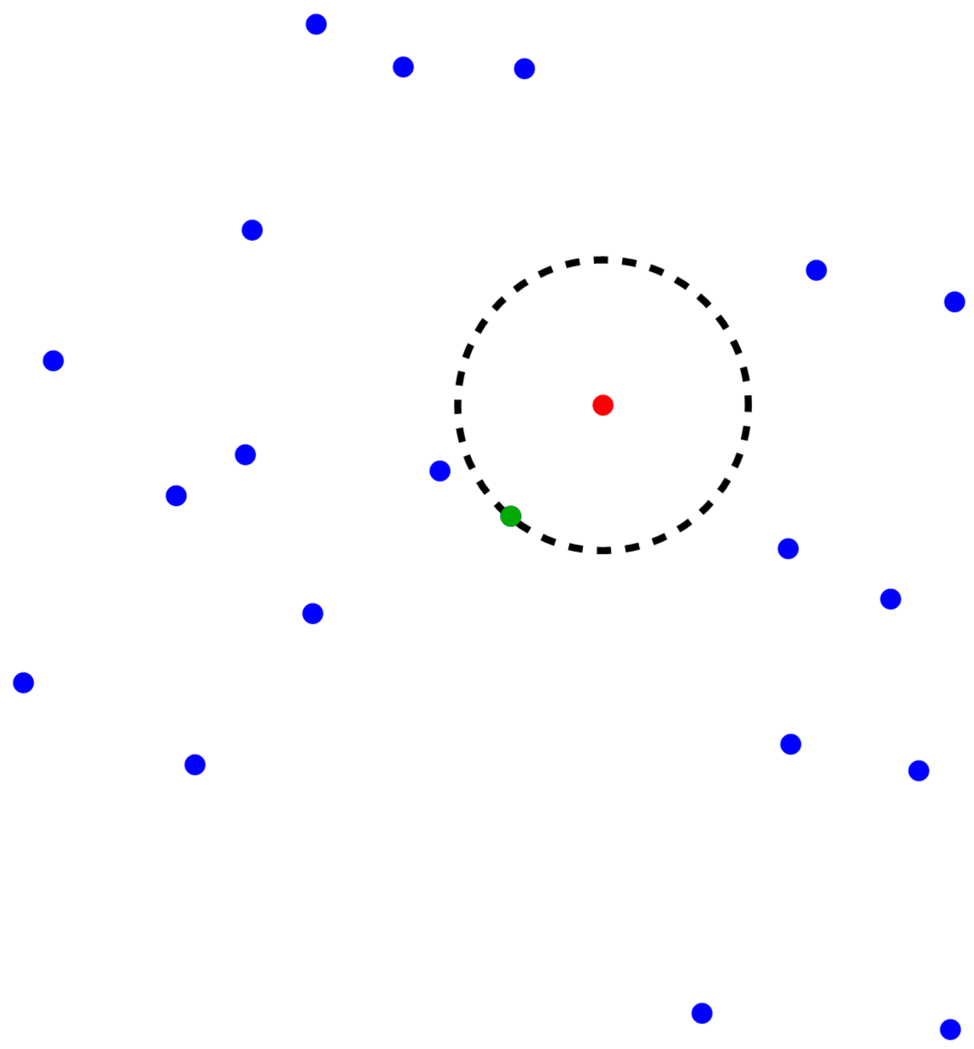
Near Neighbor Search

- **Dataset:** $\mathbf{n}$ points in $\mathbf{R}^{\mathbf{d}}$, $\mathbf{r > 0}$
- **Goal:** a data point within $\mathbf{r}$ from a query



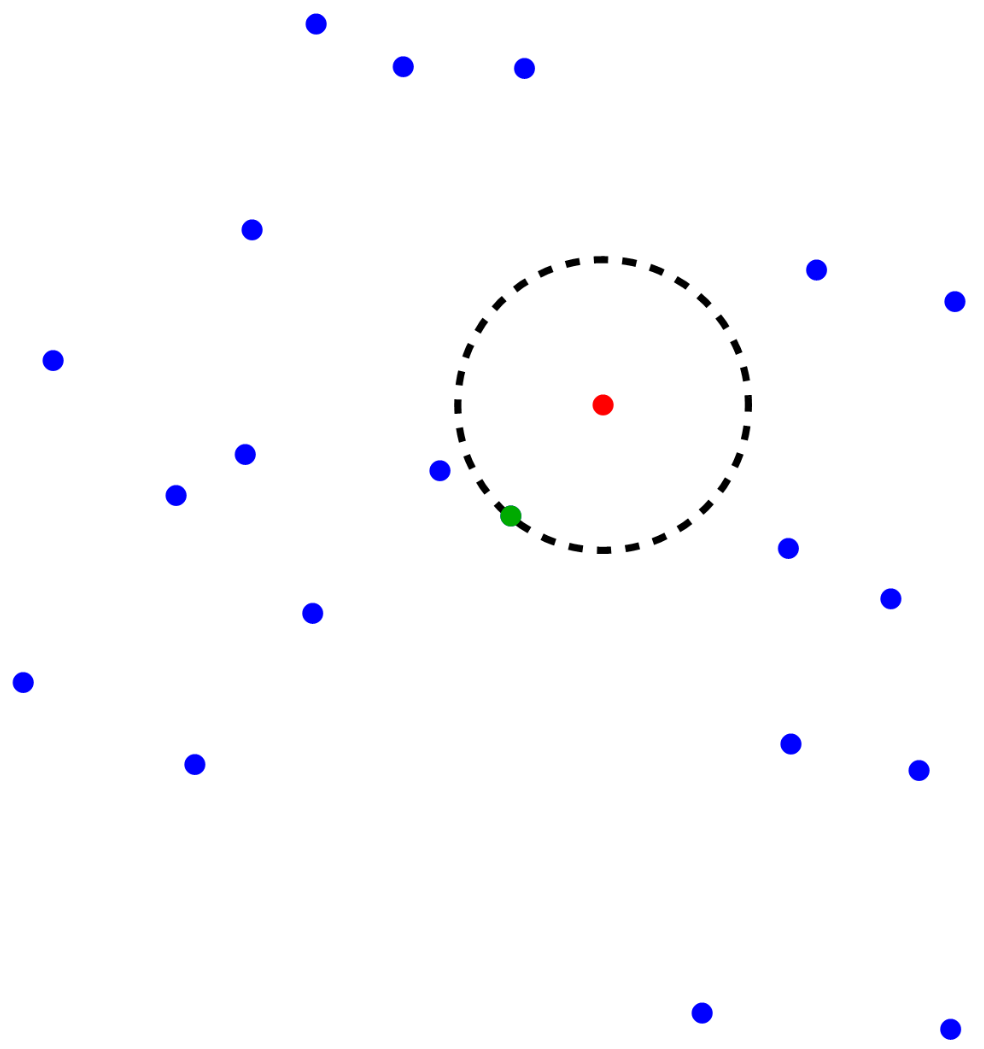
Near Neighbor Search

- **Dataset:** $\mathbf{n}$ points in $\mathbf{R}^{\mathbf{d}}$, $\mathbf{r > 0}$
- **Goal:** a data point within $\mathbf{r}$ from a query
- Space, query time



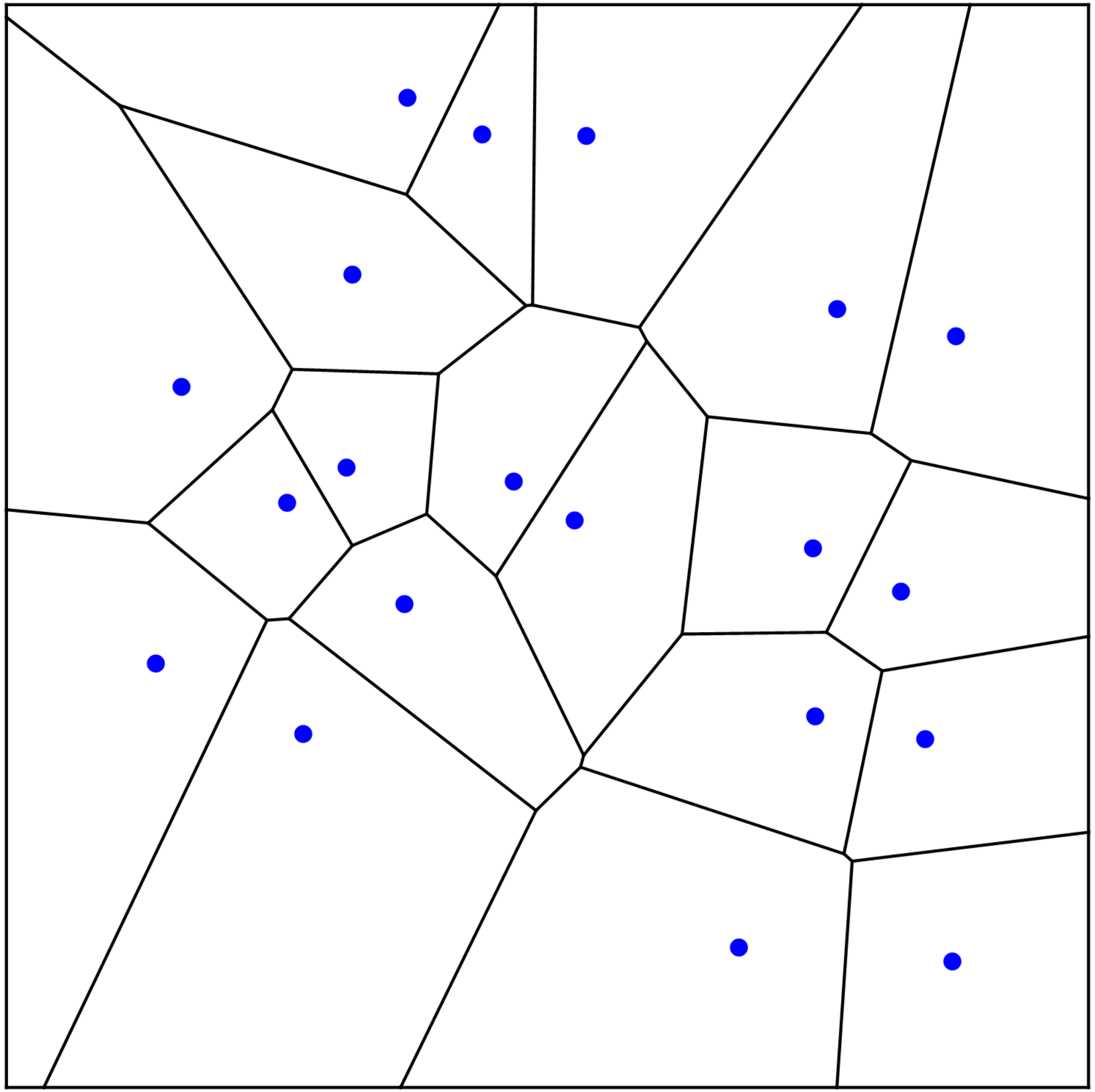
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



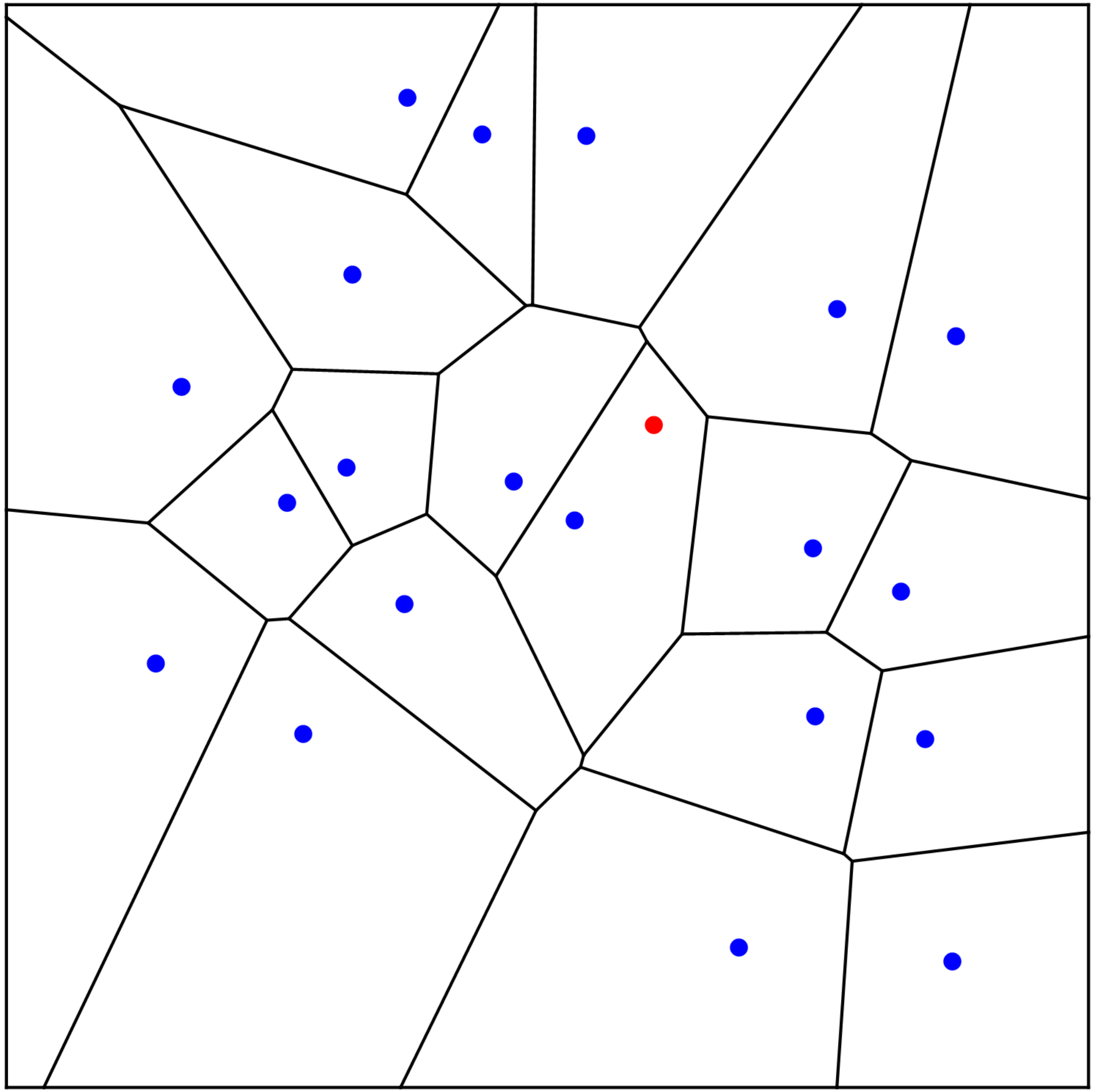
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



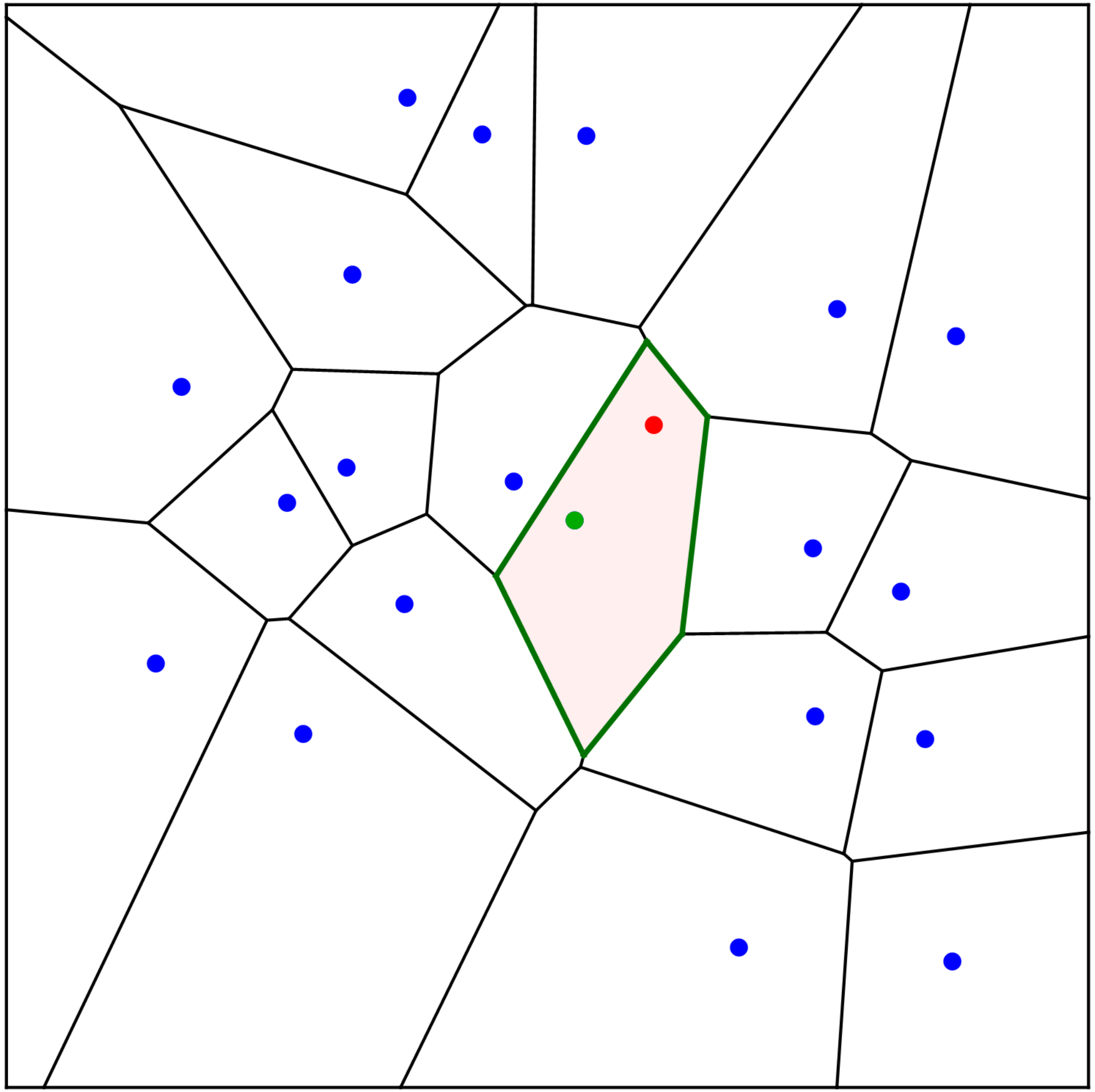
Near Neighbor Search

- **Dataset:** **n** points in **R^d**, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



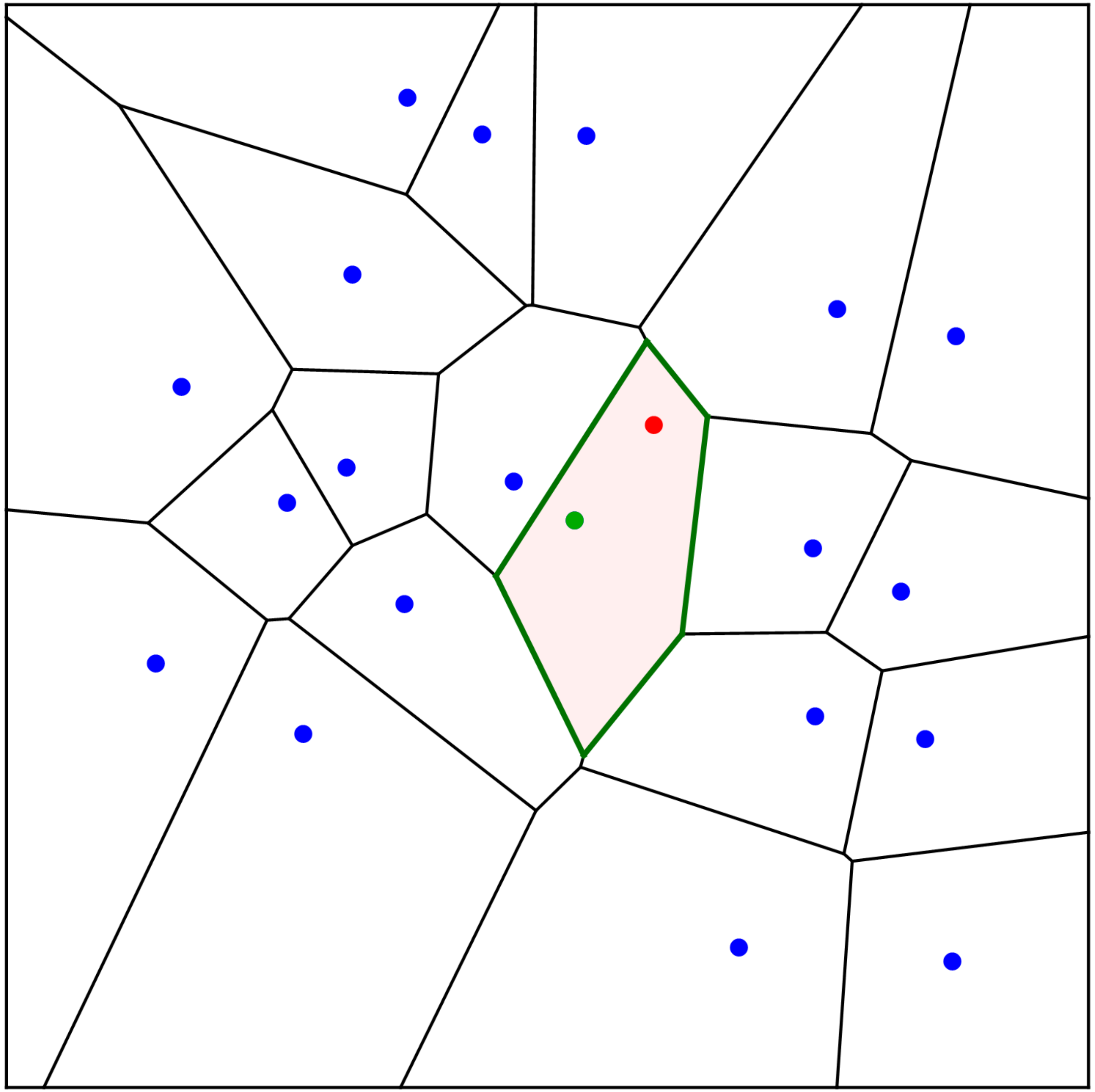
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time



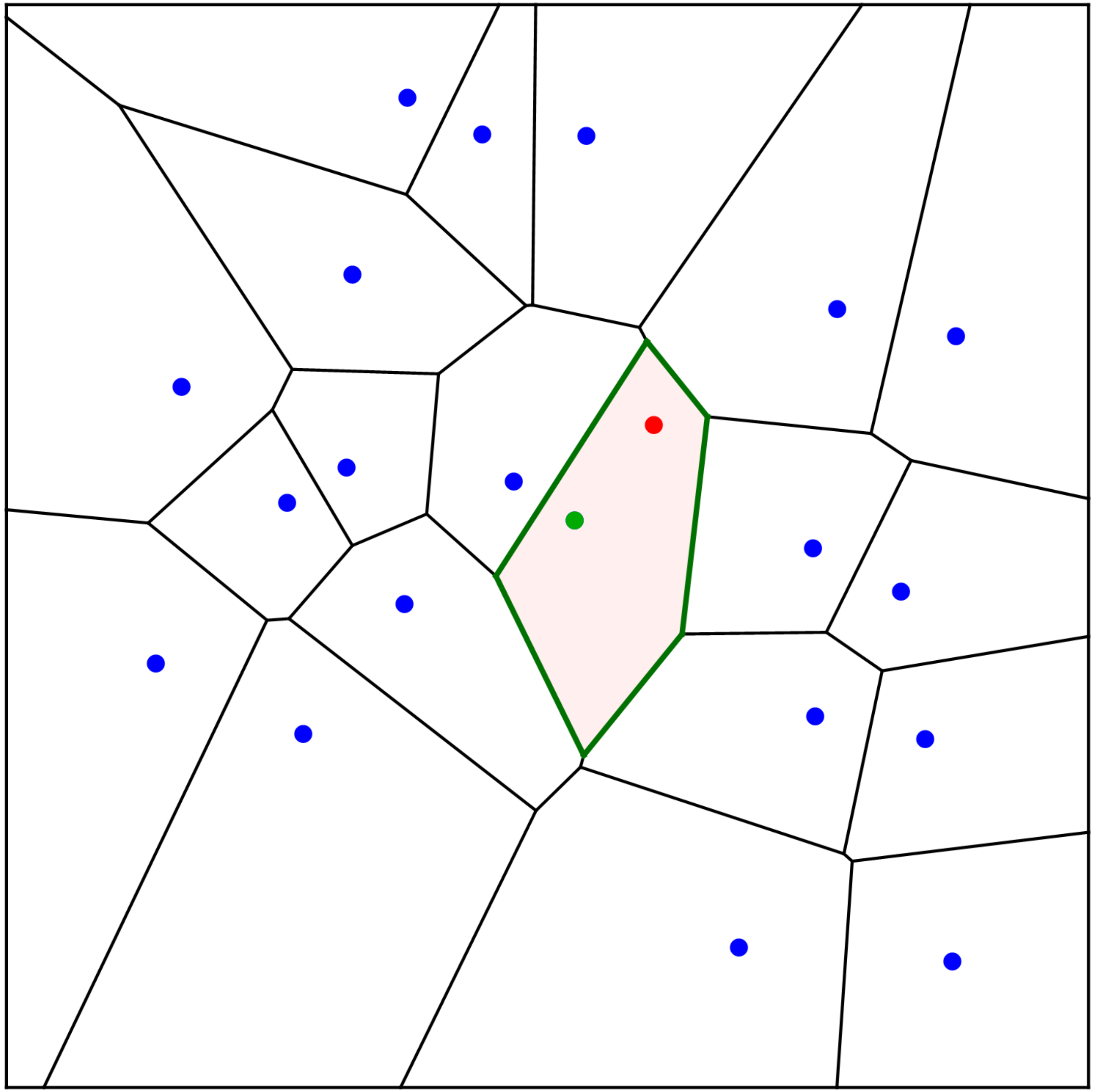
Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time
- Infeasible for large **d**:
  - Space exponential in the dimension



Near Neighbor Search

- **Dataset:** **n** points in $\mathbf{R}^d$, **r > 0**
- **Goal:** a data point within **r** from a query
- Space, query time
- **d = 2**, Euclidean distance
  - **O(n)** space
  - **O(log n)** time
- Infeasible for large **d**:
  - Space exponential in the dimension
- Most of the applications are in high dimensions



Detour: closest pair in *low* dimensions

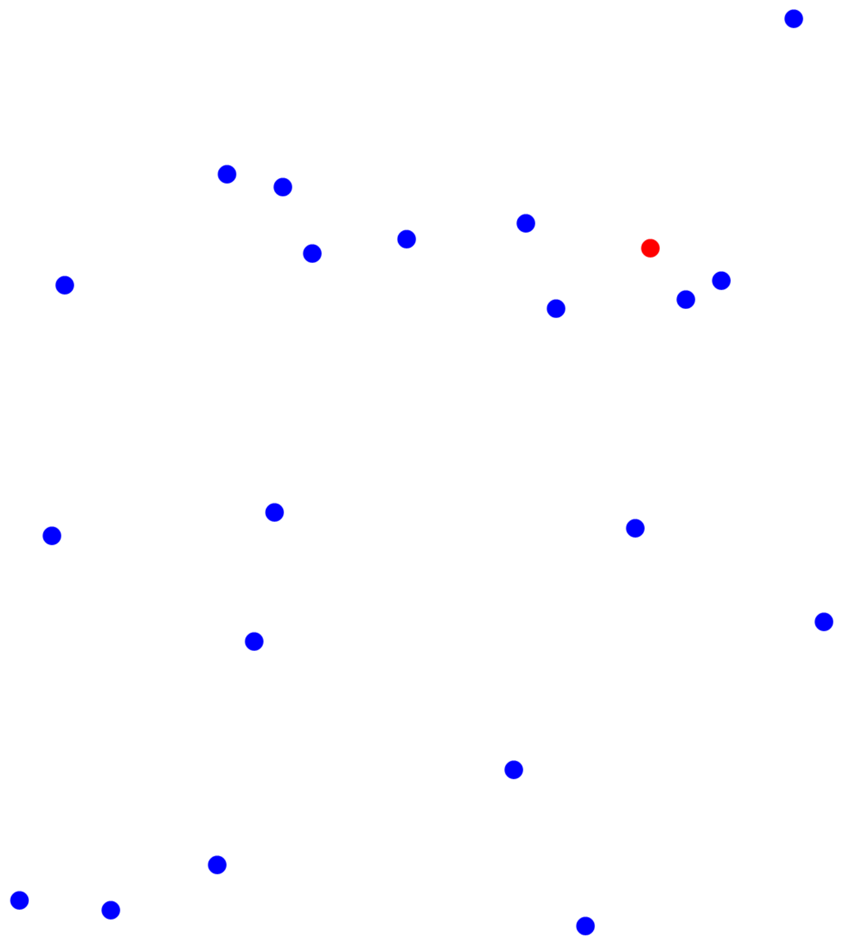
Approximate Near Neighbor Search (ANN)

Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$

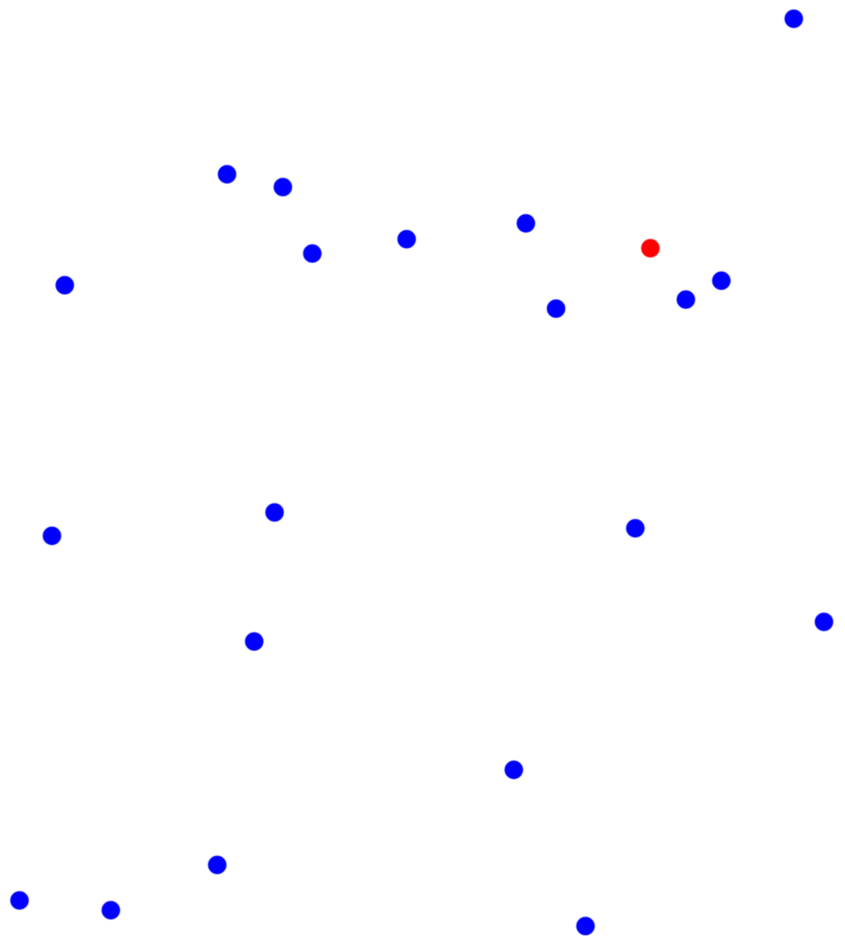
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$



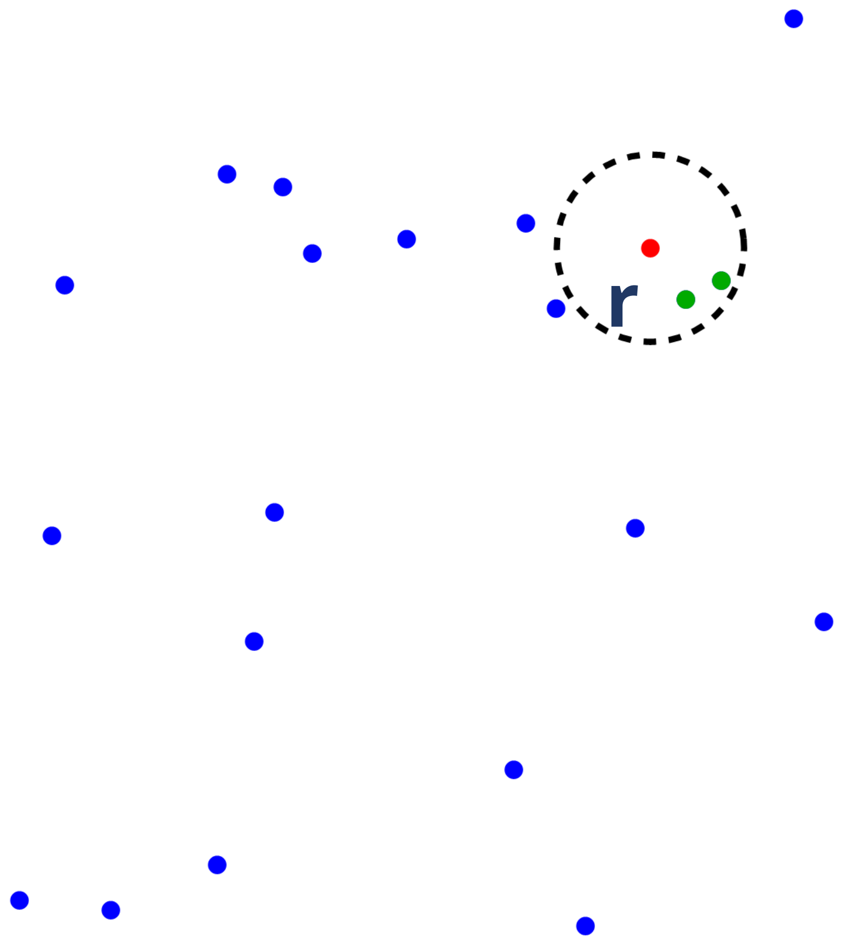
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$
- **Query:** a point within **r** from a data point



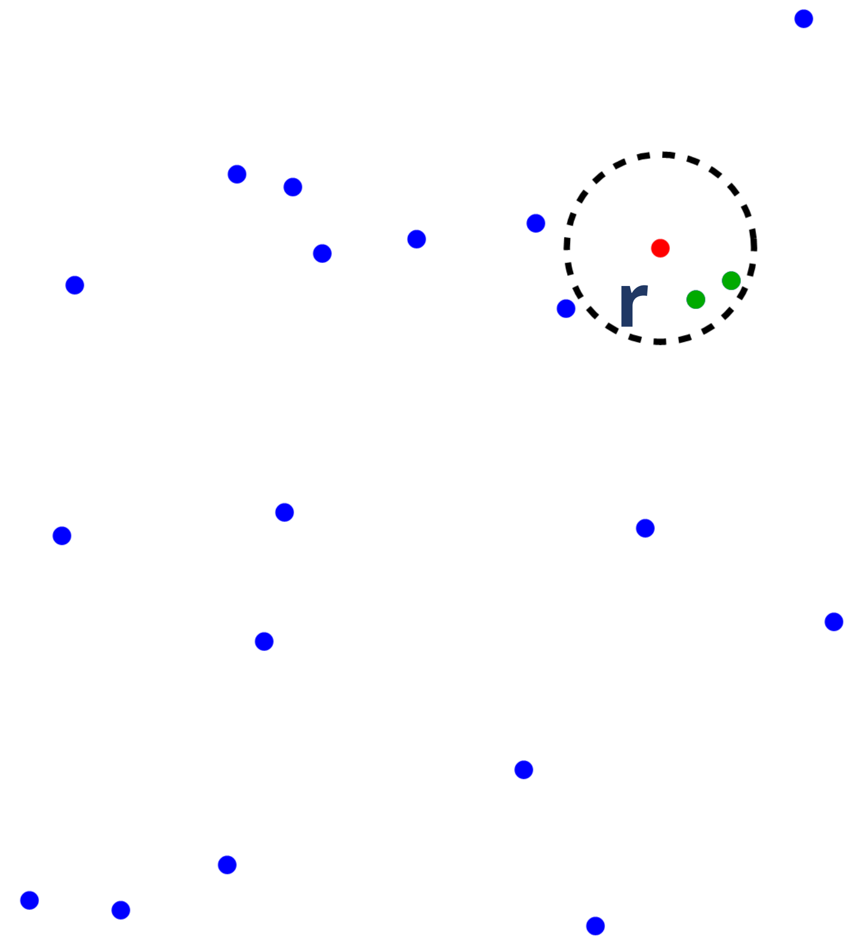
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$
- **Query:** a point within **r** from a data point



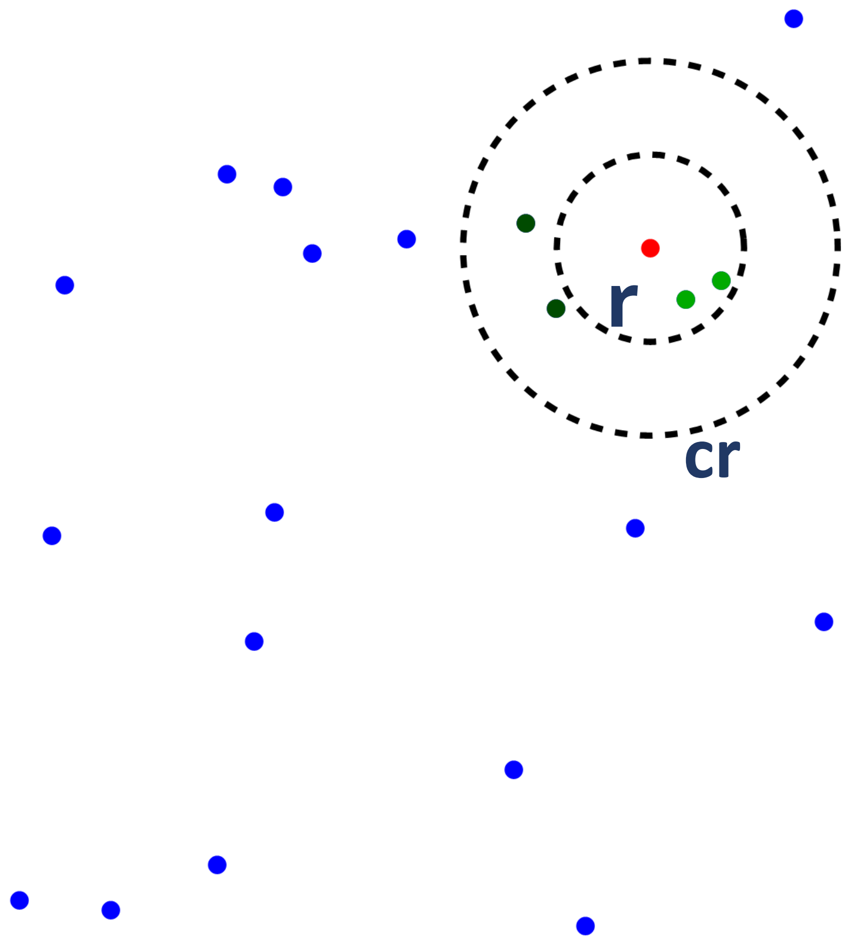
Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$
- **Query:** a point within **r** from a data point
- **Want:** a data point within **cr** from the query



Approximate Near Neighbor Search (ANN)

- **Given:**
  - **n** points in $\mathbf{R}^d$
  - distance threshold $\mathbf{r > 0}$
  - approximation $\mathbf{c > 1}$
- **Query:** a point within **r** from a data point
- **Want:** a data point within **cr** from the query



Applications

Applications

- Similarity search for: images, audio, video, texts, biological data etc

Applications

- Similarity search for: images, audio, video, texts, biological data etc
- Cryptanalysis (the Shortest Vector Problem in lattices) **[Laarhoven 2015]**

Applications

- Similarity search for: images, audio, video, texts, biological data etc
- Cryptanalysis (the Shortest Vector Problem in lattices) **[Laarhoven 2015]**
- Optimization: Coordinate Descent **[Dhillon, Ravikumar, Tewari 2011]**, Stochastic Gradient Descent **[Hofmann, Lucchi, McWilliams 2015]** etc

Spherical case

Spherical case

- Focus of this talk:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

Spherical case

- Focus of this talk:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

- Why interesting?

Spherical case

- Focus of this talk:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

- Why interesting?
- **In theory:** can reduce general case to the spherical case **[Andoni, R 2015]**

Spherical case

- Focus of this talk:

  all points and queries lie on a **unit sphere** in $\mathbf{R}^d$

- Why interesting?
- **In theory:** can reduce general case to the spherical case **[Andoni, R 2015]**
- **In practice:**
  - Cosine similarity is widely used
  - Oftentimes, can *pretend* that the dataset lies on a sphere

Spherical *random* case

Spherical *random* case

- **Dataset:** **n** random points on a sphere

Spherical *random* case

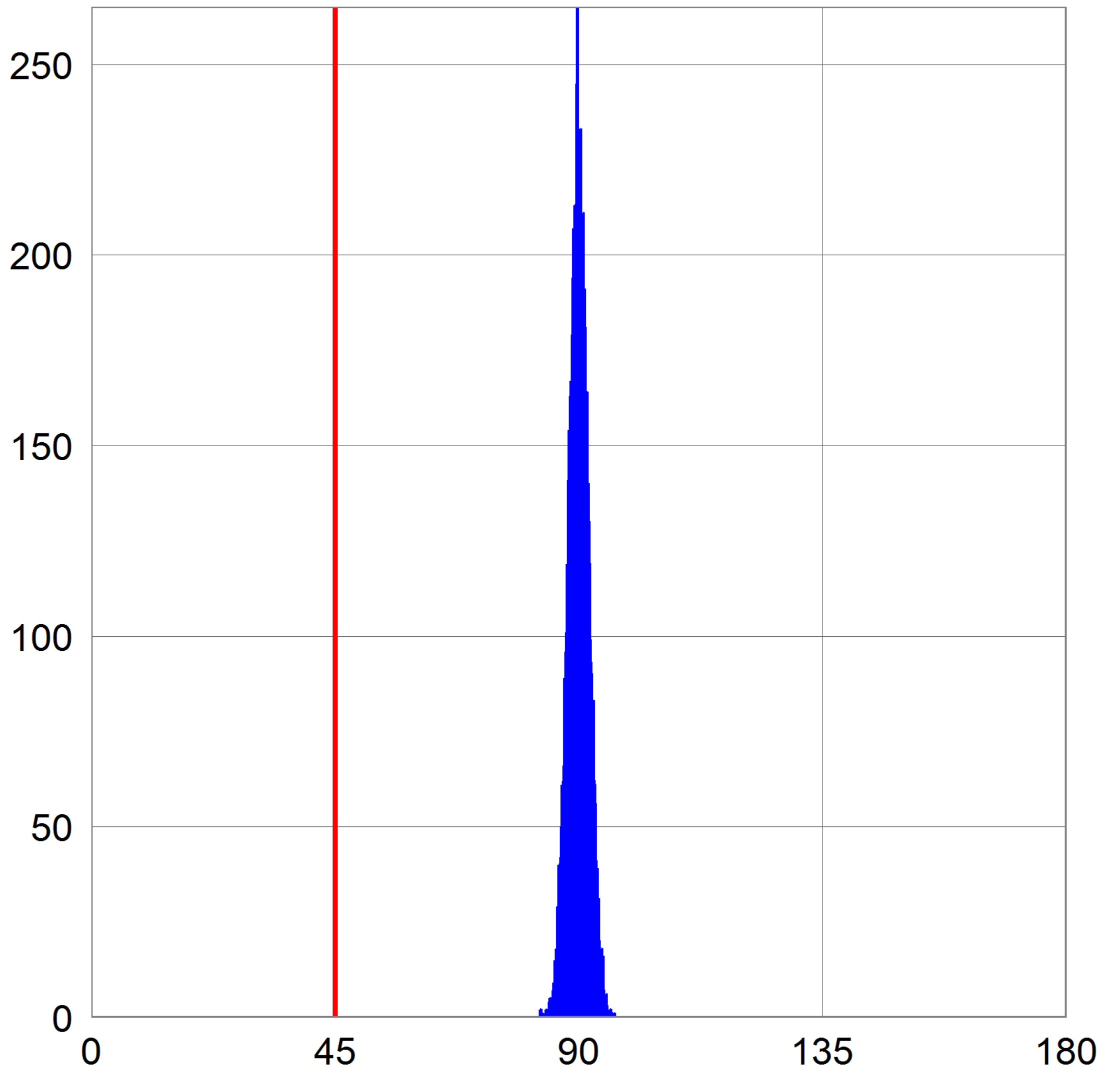
- **Dataset:** **n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point

Spherical *random* case

- **Dataset:** **n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point
- Distribution of angles: near neighbor within **45** degrees, other data points at ~**90** degrees!

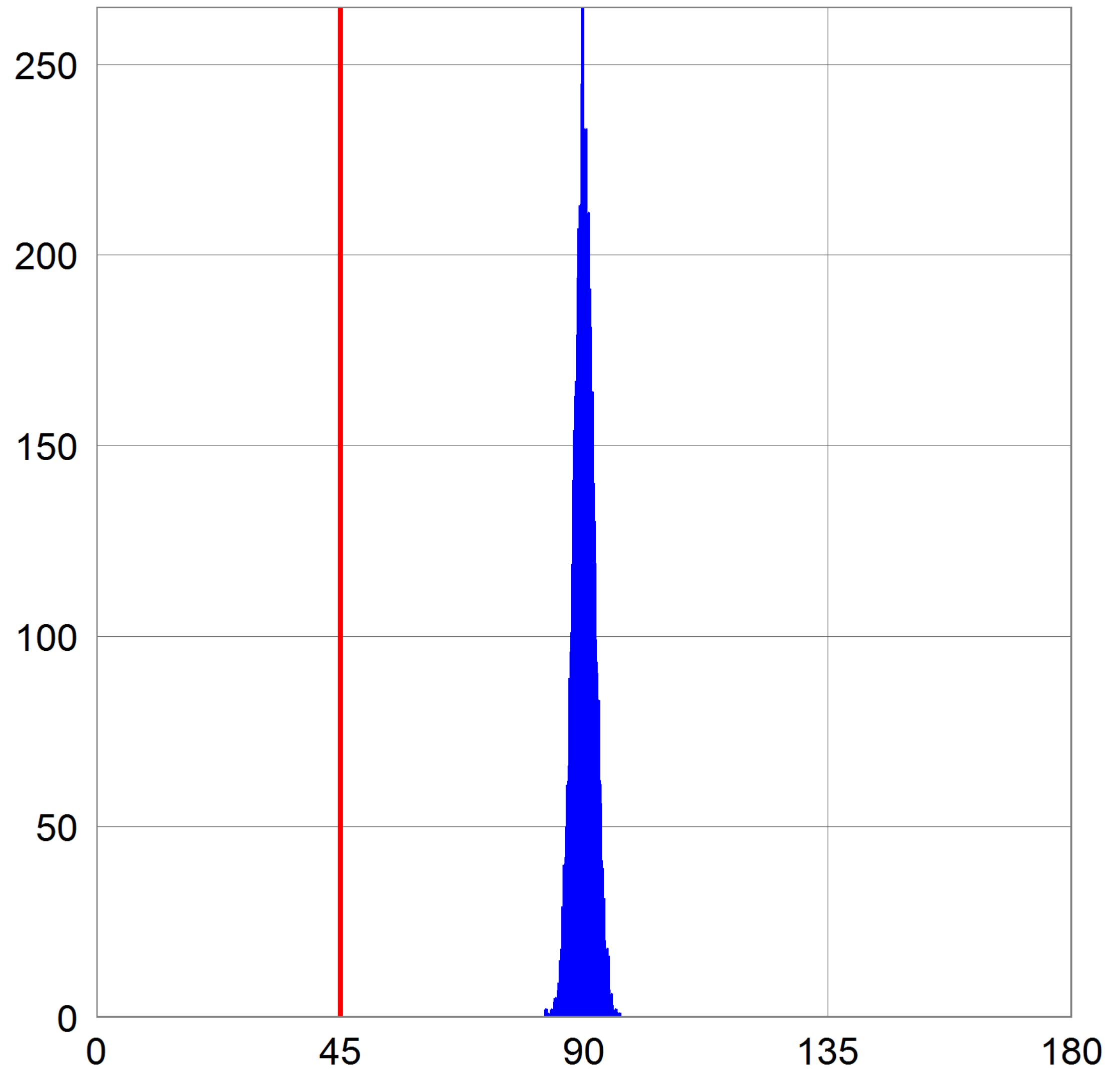
Spherical *random* case

- **Dataset: n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point
- Distribution of angles: near neighbor within **45** degrees, other data points at ~**90** degrees!



Spherical *random* case

- **Dataset:** **n** random points on a sphere
- **Query:** a random query within **45** degrees from a data point
- Distribution of angles: near neighbor within **45** degrees, other data points at ~**90** degrees!
- Instructive case to think about
  - **[Andoni, R 2015]:** a (delicate) reduction from general to random
  - Concentration of angles around **90** degrees happens in practice



Locality-Sensitive Hashing (LSH)

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often
- A random partition **R** is **(r, cr, p_1, p_2)-sensitive** if for every **p, q**:
  - If $\|p - q\| \le r$, then $\Pr_R[R(p) = R(q)] \ge p_1$
  - If $\|p - q\| \ge cr$, then $\Pr_R[R(p) = R(q)] \le p_2$

Locality-Sensitive Hashing (LSH)

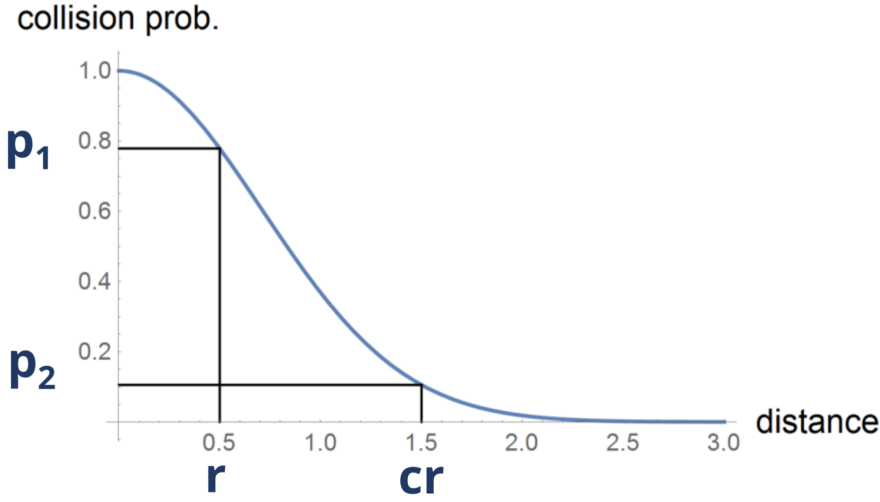
- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often
- A random partition **R** is **(r, cr, p_1, p_2)-sensitive** if for every **p, q**:
  - If $\|p - q\| \le r$, then $\Pr_R[R(p) = R(q)] \ge p_1$
  - If $\|p - q\| \ge cr$, then $\Pr_R[R(p) = R(q)] \le p_2$

From the definition of ANN

Locality-Sensitive Hashing (LSH)

- Introduced in **[Indyk, Motwani 1998]**
- **Main idea:** *random* partitions of $\mathbf{R}^d$ s.t. closer pairs of points collide more often
- A random partition $\mathbf{R}$ is **(r, cr, p_1, p_2)-sensitive** if for every $\mathbf{p}, \mathbf{q}$:
    - If $\|\mathbf{p} - \mathbf{q}\| \le r$, then $\Pr_R[R(p) = R(q)] \ge p_1$
    - If $\|\mathbf{p} - \mathbf{q}\| \ge cr$, then $\Pr_R[R(p) = R(q)] \le p_2$

From the definition of ANN



Hyperplane LSH

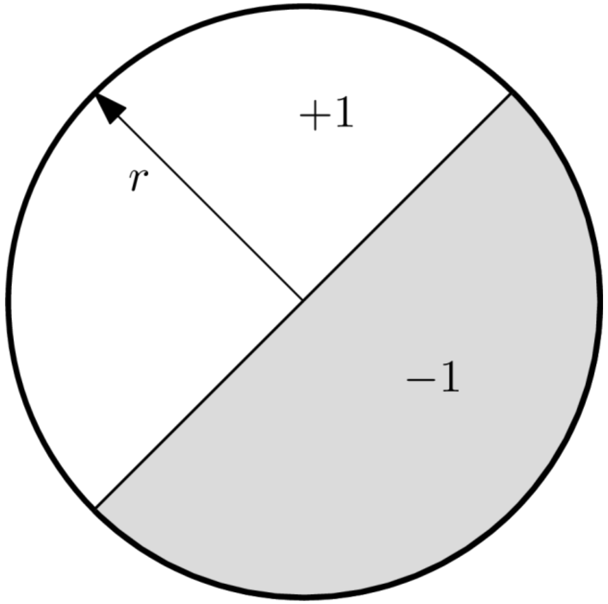
Hyperplane LSH

- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**

Hyperplane LSH

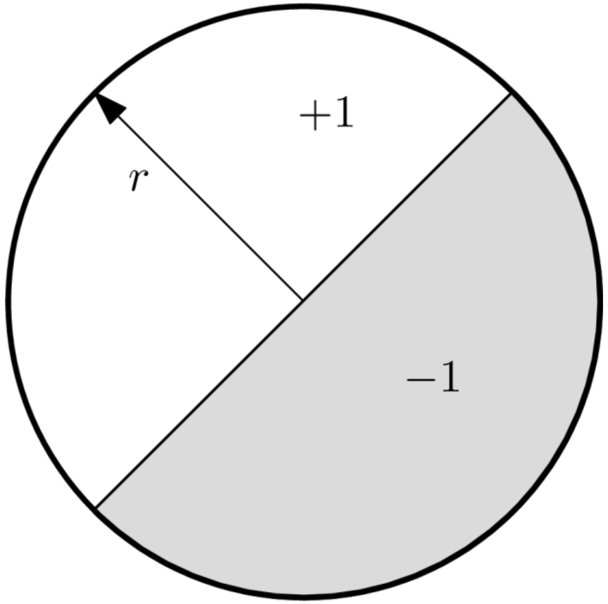
- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**

Hyperplane LSH



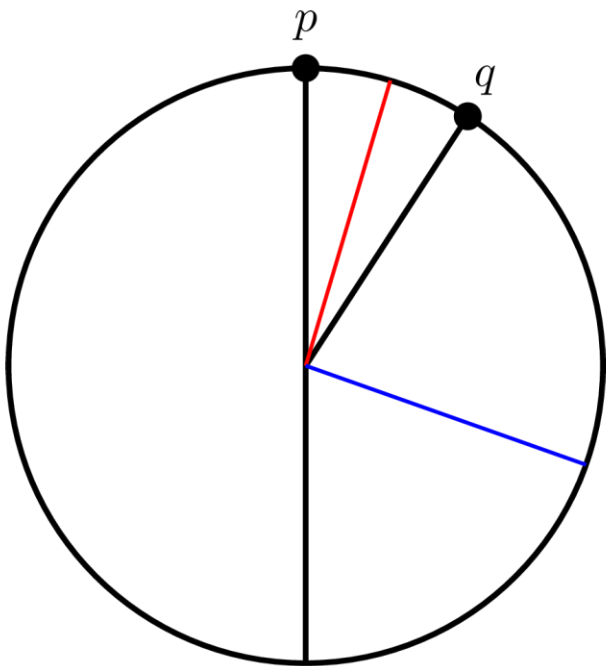
- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**

Hyperplane LSH



- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**
- **Pr[h(p) = h(q)] = 1 – α / π**, where **α** is the angle between **p** and **q**

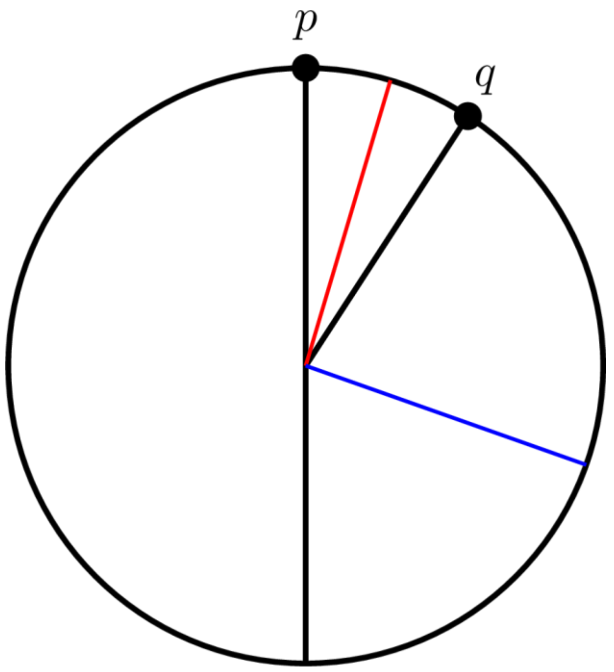
Hyperplane LSH

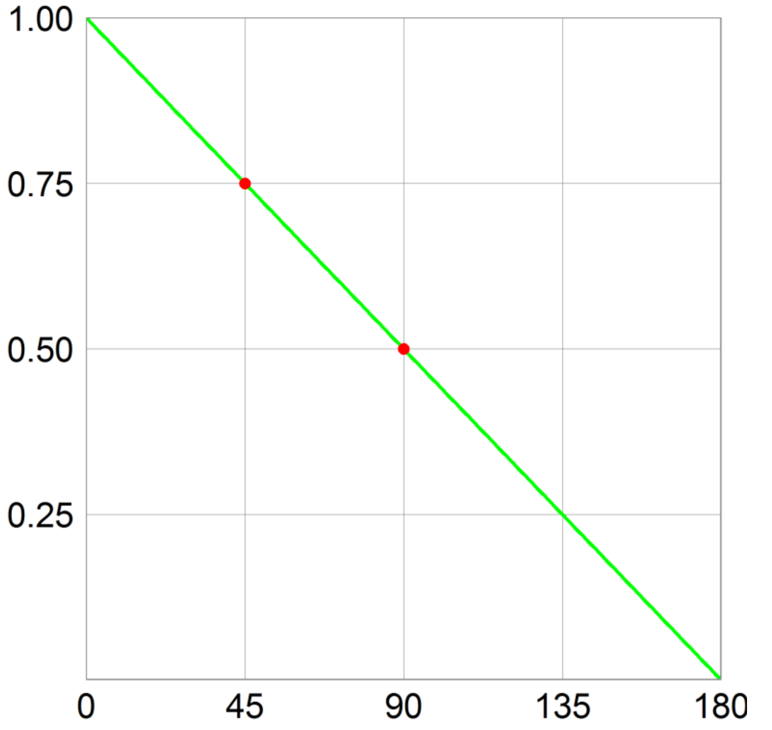


- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**
- **Pr[h(p) = h(q)] = 1 – α / π**, where **α** is the angle between **p** and **q**

Hyperplane LSH

- Introduced in **[Charikar 2002]**, inspired by **[Goemans, Williamson 1995]**
- Sample *unit* **r** uniformly, hash **p** into **sgn <r, p>**
- **Pr[h(p) = h(q)] = 1 – α / π**, where **α** is the angle between **p** and **q**





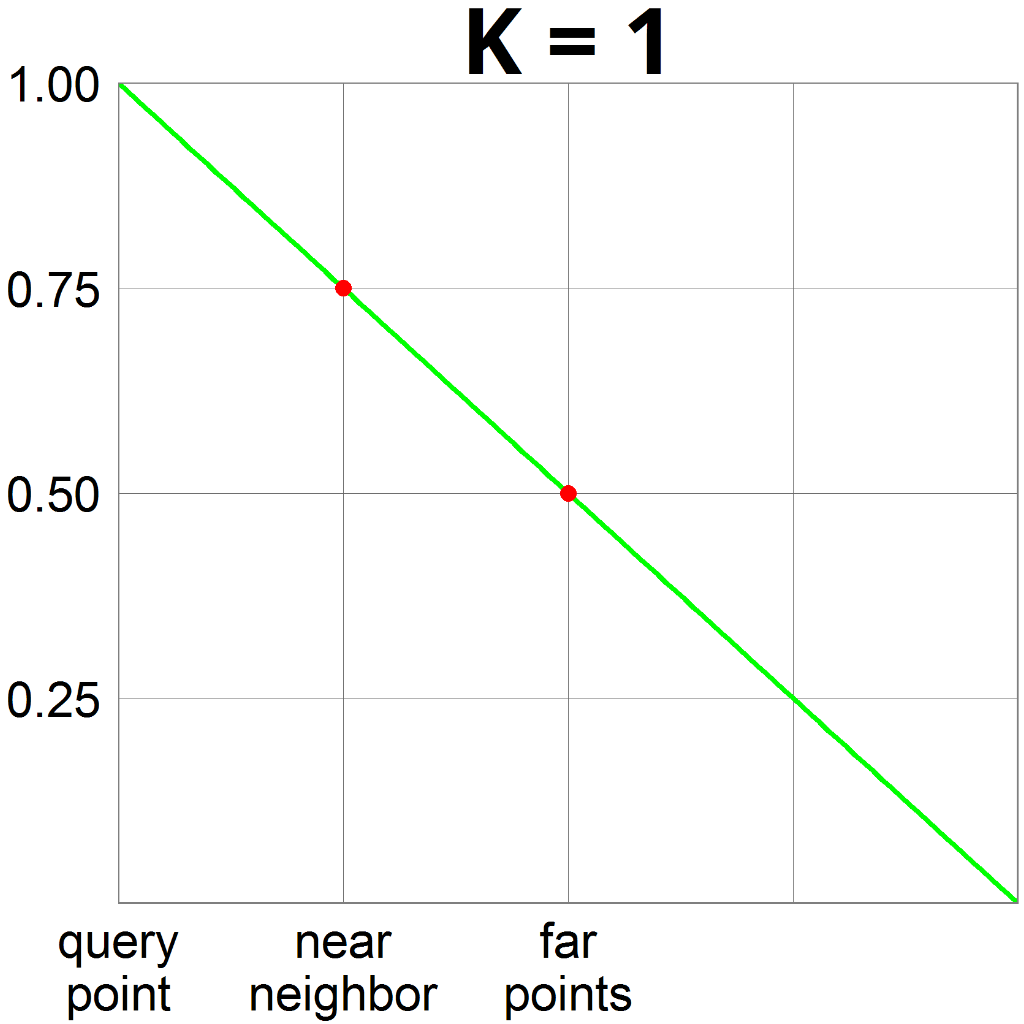
Using LSH to solve ANN

Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)

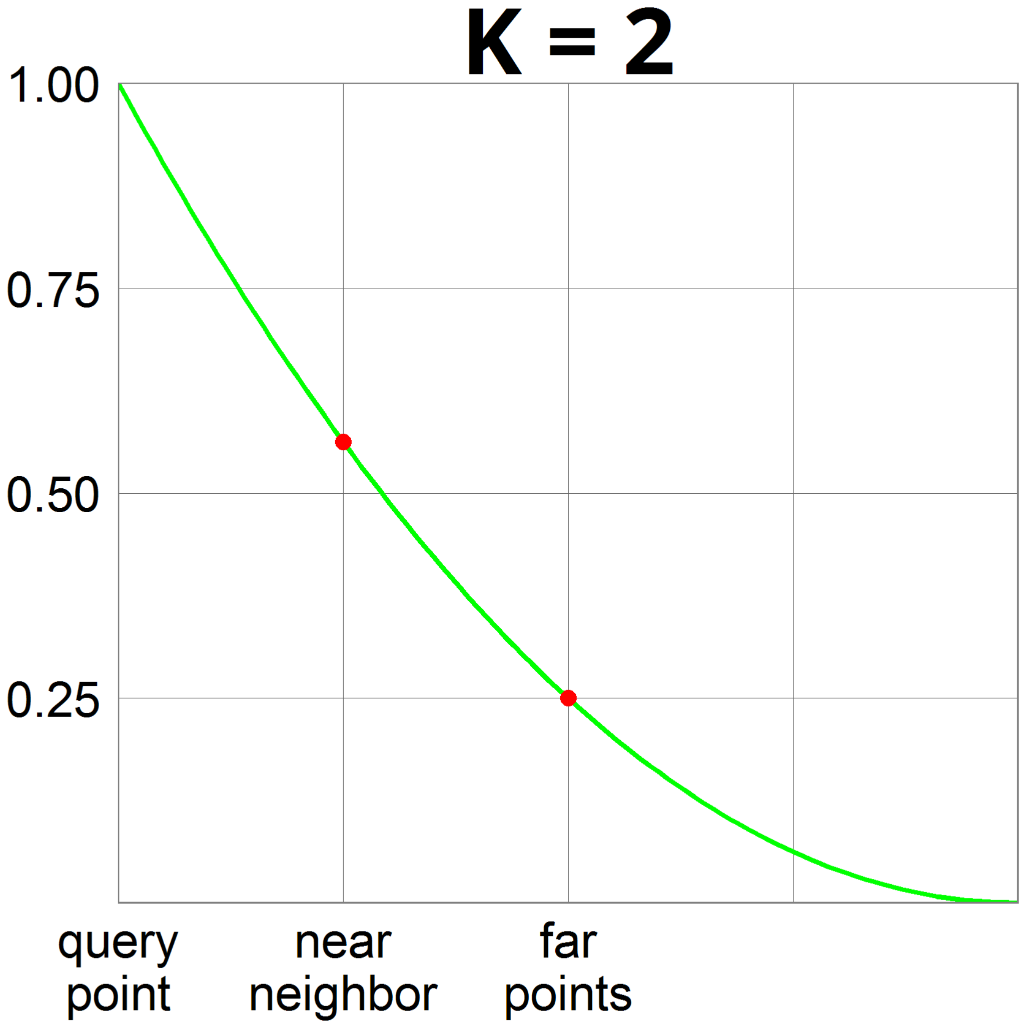
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), ..., h_K(p))**)



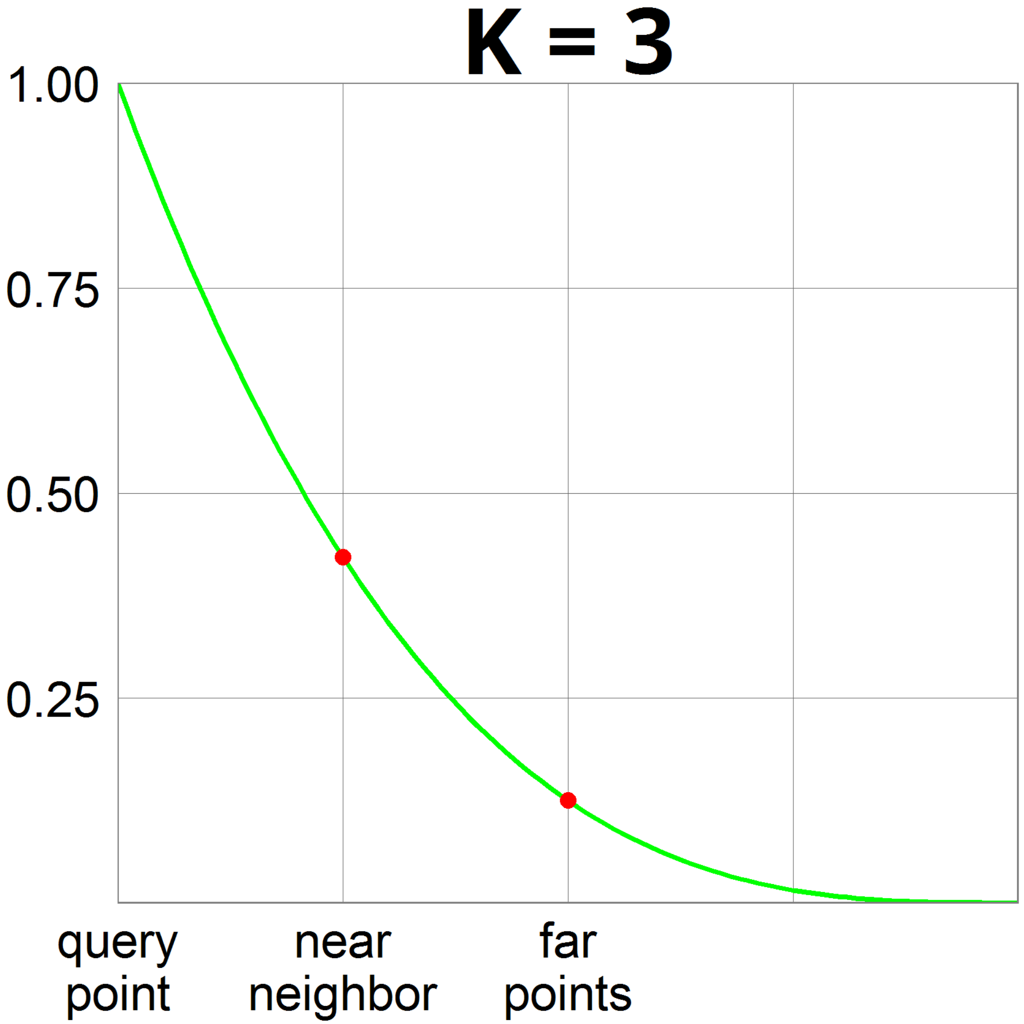
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)



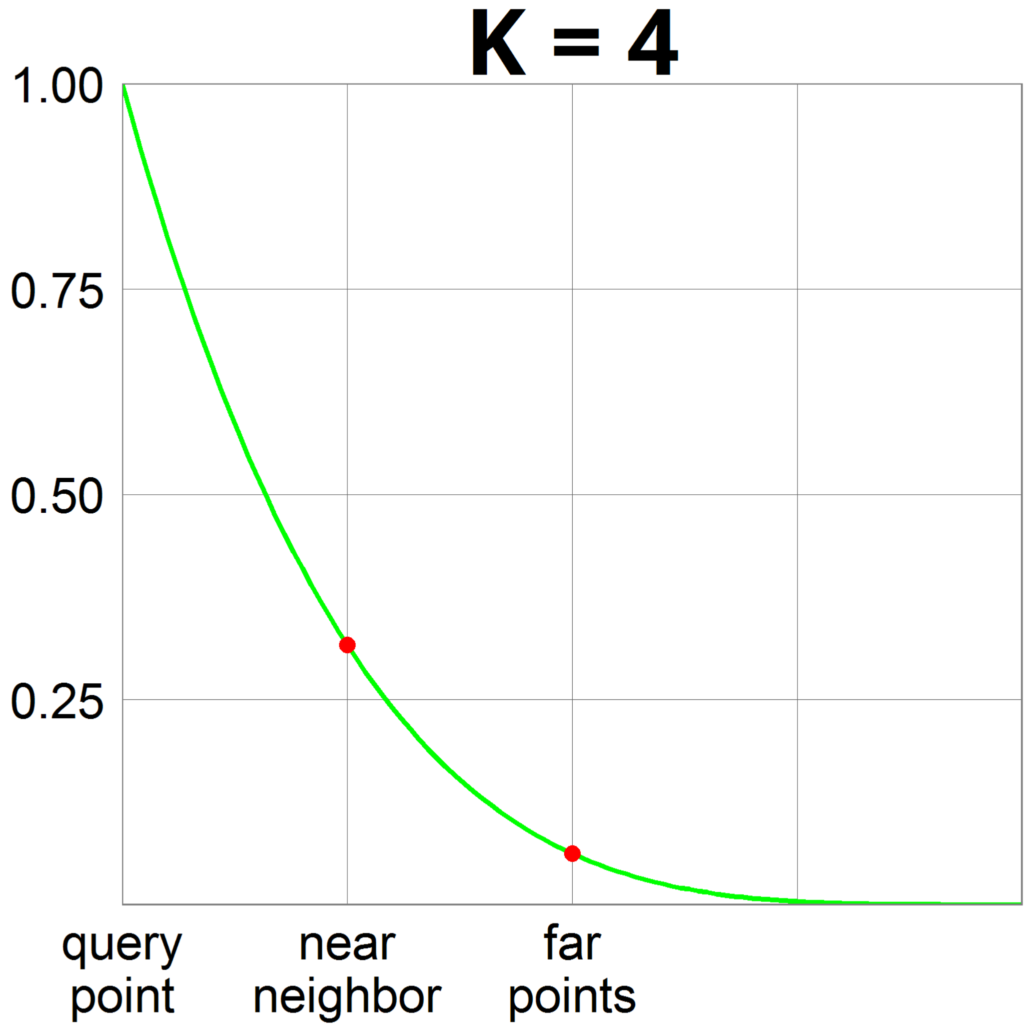
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)



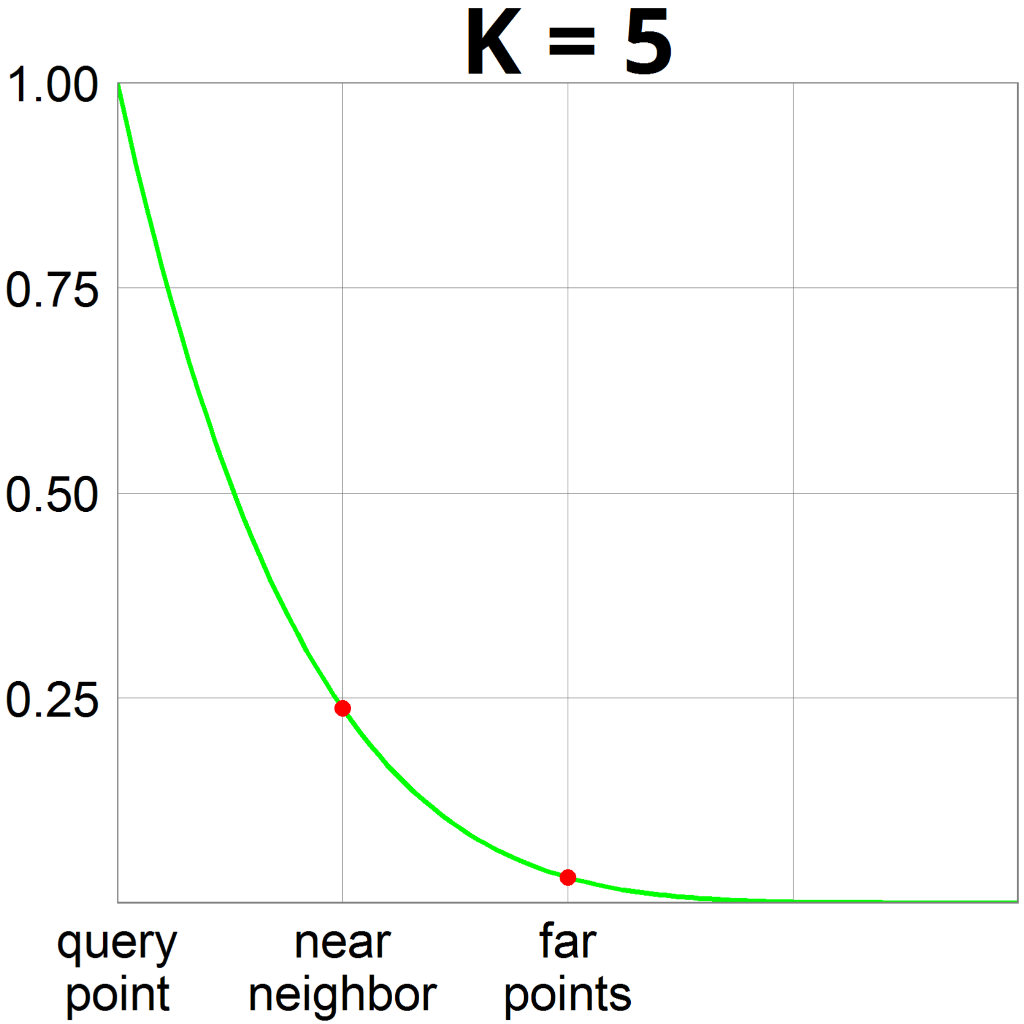
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **$(h_1(p), \ldots, h_K(p))$**)



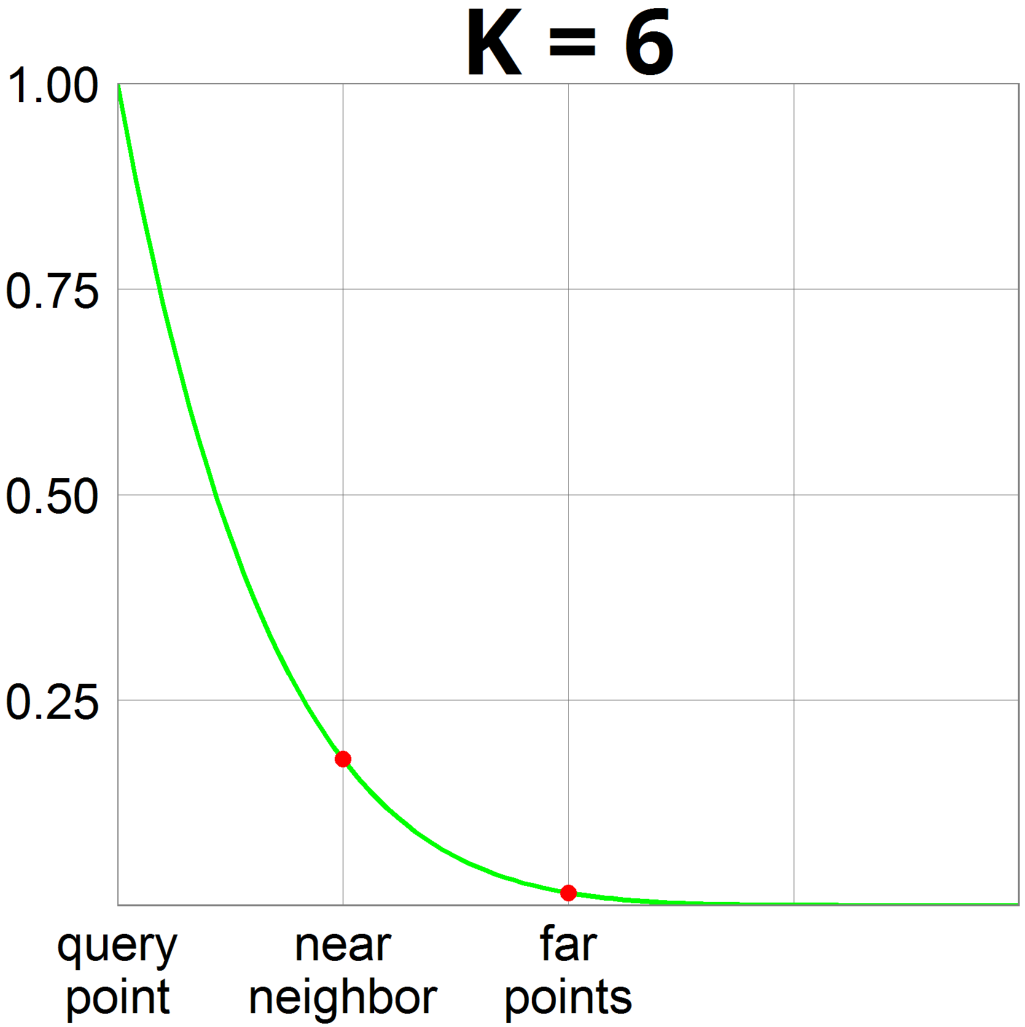
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)



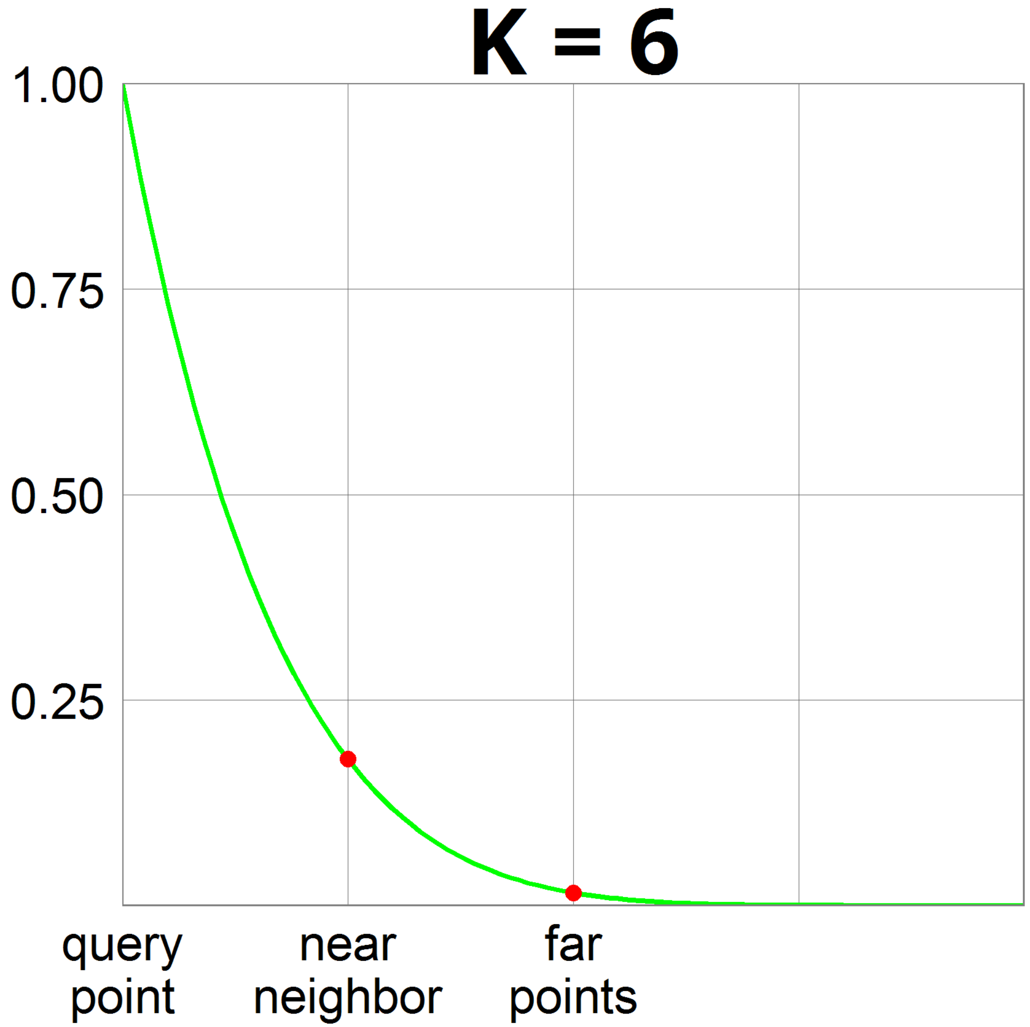
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)



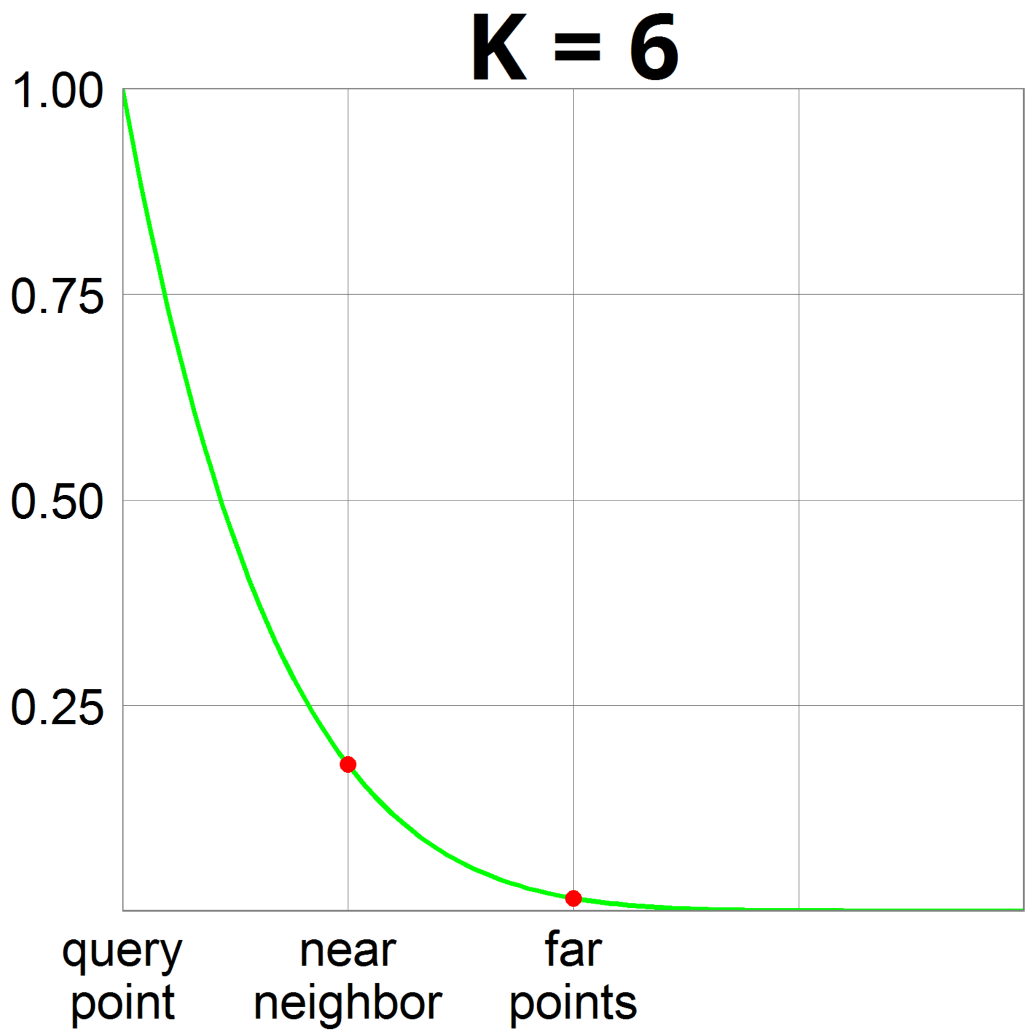
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)
- If **$0.5^K \sim 1/n$**, then **O(1)** far points in a query bin



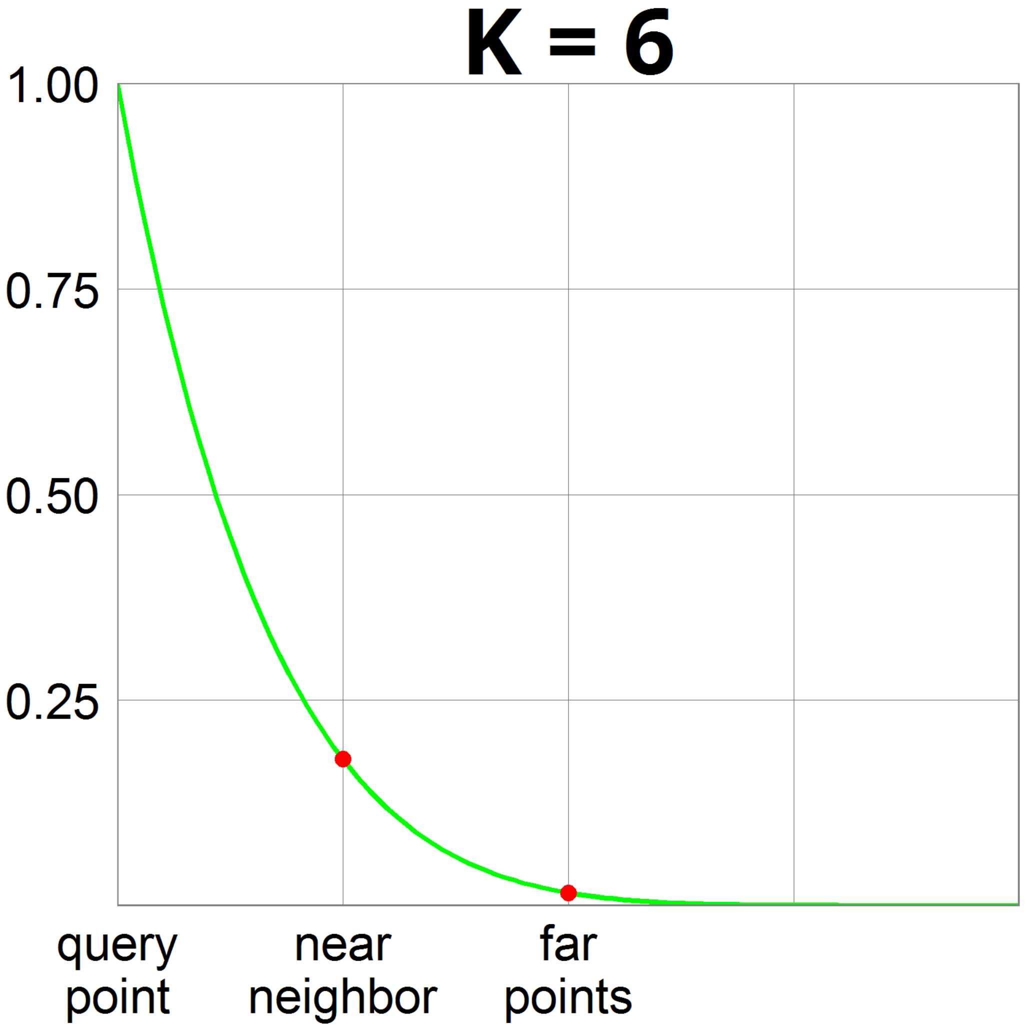
Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **($h_1(p), \ldots, h_K(p)$)**)
- If **$0.5^K \sim 1/n$**, then **O(1)** far points in a query bin
- Collides with near neighbor with probability **$0.75^K \sim 1/n^{0.42}$**
- Thus, need **$L = O(n^{0.42})$** tables to boost the success probability to **0.99**



Using LSH to solve ANN

- **K** hash functions at once (hash **p** into **(h_1(p), …, h_K(p))**)
- If **$0.5^K \sim 1/n$**, then **O(1)** far points in a query bin
- Collides with near neighbor with probability **$0.75^K \sim 1/n^{0.42}$**
- Thus, need **L = O($n^{0.42}$)** tables to boost the success probability to **0.99**
- Overall: **O($n^{1.42}$)** space, **O($n^{0.42}$)** query time, **K·L** hyperplanes



Using LSH to solve ANN (in general)

Using LSH to solve ANN (in general)

In general **[Indyk, Motwani 1998]**: can always choose **K** (# of functions / table) and **L** (# of tables) to get space $\mathbf{O(n^{1+\rho})}$ and query time $\mathbf{O(n^{\rho})}$, where

Using LSH to solve ANN (in general)

In general **[Indyk, Motwani 1998]**: can always choose **K** (# of functions / table) and **L** (# of tables) to get space $\mathbf{O(n^{1+\rho})}$ and query time $\mathbf{O(n^{\rho})}$, where

$$\rho = \ln(1/p_1) / \ln(1/p_2)$$

Using LSH to solve ANN (in general)

In general **[Indyk, Motwani 1998]**: can always choose **K** (# of functions / table) and **L** (# of tables) to get space $\mathbf{O(n^{1+\rho})}$ and query time $\mathbf{O(n^{\rho})}$, where

$$\rho = \ln(1/p_1) / \ln(1/p_2)$$

Recap:

- $\mathbf{p_1}$ is collision probability for close pairs
- $\mathbf{p_2}$ — for far pairs

Better than Hyperplane LSH?

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**
  - **[Andoni, R ??]:** tight for hashing-based approaches!

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**
  - **[Andoni, R ??]:** tight for hashing-based approaches!
  - Works for the general case of ANN on a sphere

Better than Hyperplane LSH?

- Can one improve upon **$O(n^{1.42})$** space and **$O(n^{0.42})$** query time for the **45**-degree random instance?
- **Yes!**
  - **[Andoni, Indyk, Nguyen, R 2014], [Andoni, R 2015]:** can achieve space **$O(n^{1.18})$** and query time **$O(n^{0.18})$**
  - **[Andoni, R ??]:** tight for hashing-based approaches!
  - Works for the general case of ANN on a sphere

Can we use this (significant) improvement in practice?

Optimal LSH family: Voronoi LSH

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$g_1, g_2, \ldots, g_T$$

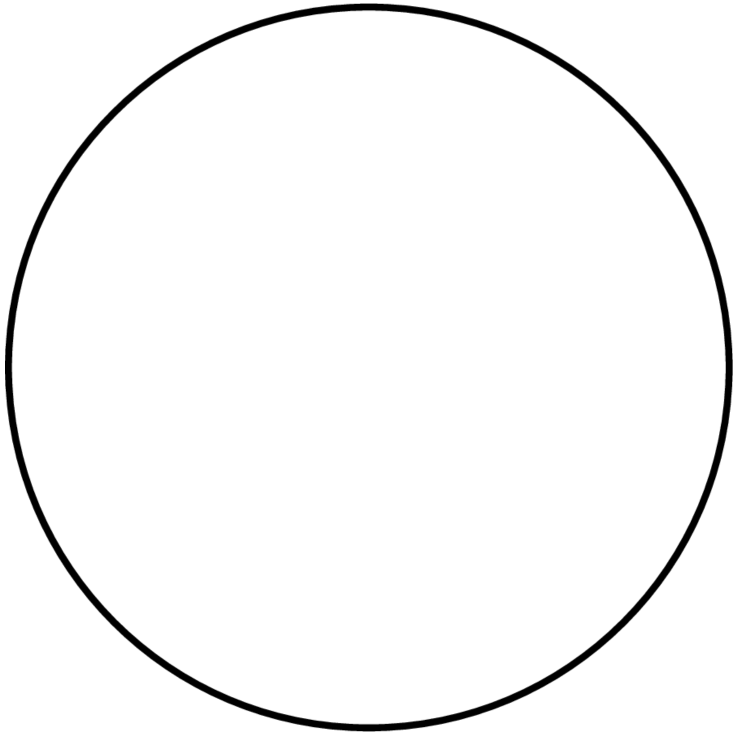
- Hash **p** into $h(p) = \mathrm{argmax}_{1 \le i \le T} \langle p, g_i \rangle$

Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$g_1, g_2, \ldots, g_T$$

- Hash **p** into $h(p) = \operatorname{argmax}_{1 \le i \le T} \langle p, g_i \rangle$

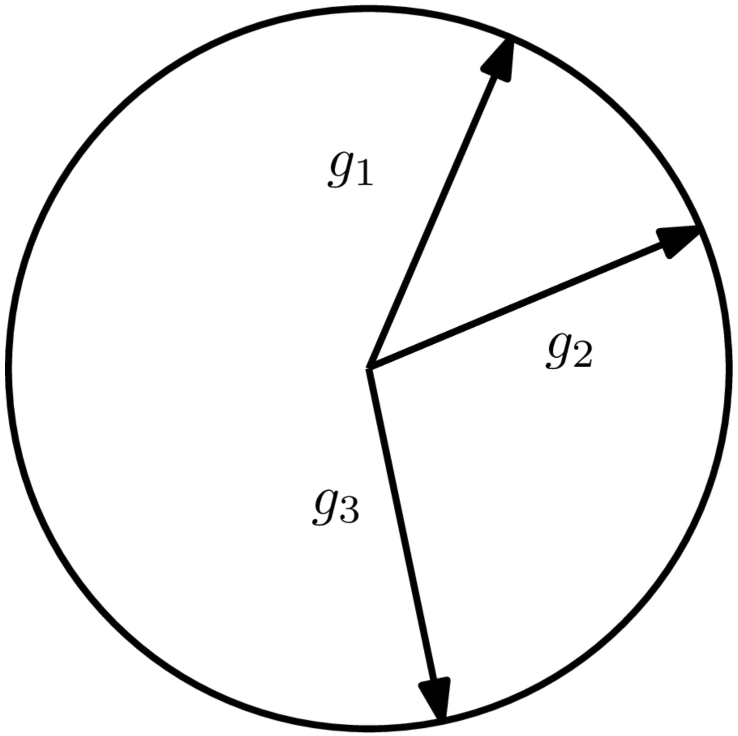


Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

- Hash **p** into $\mathbf{h(p) = \operatorname{argmax}_{1 \le i \le T} \langle p, g_i \rangle}$

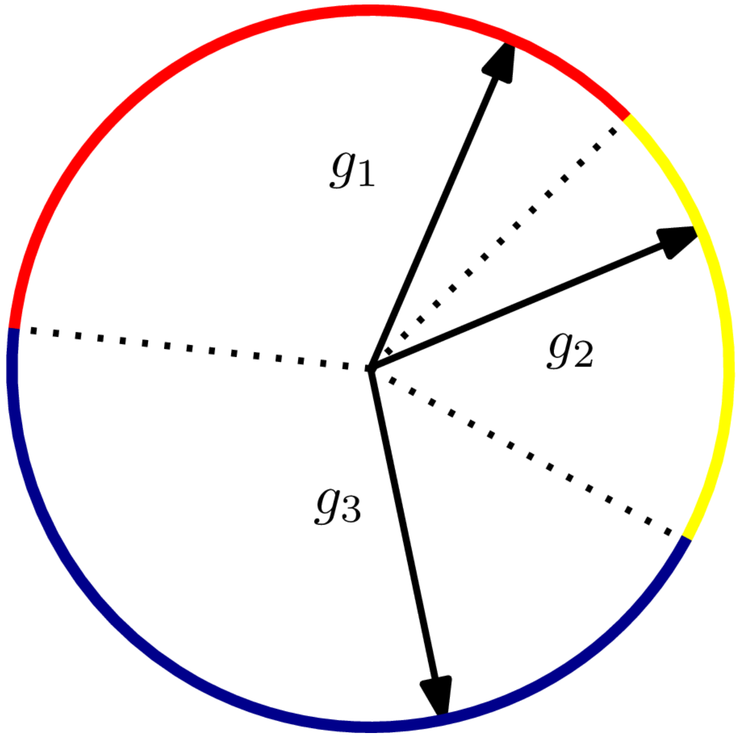


Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

- Hash **p** into $\mathbf{h(p) = argmax_{1 \leq i \leq T} \langle p, g_i \rangle}$

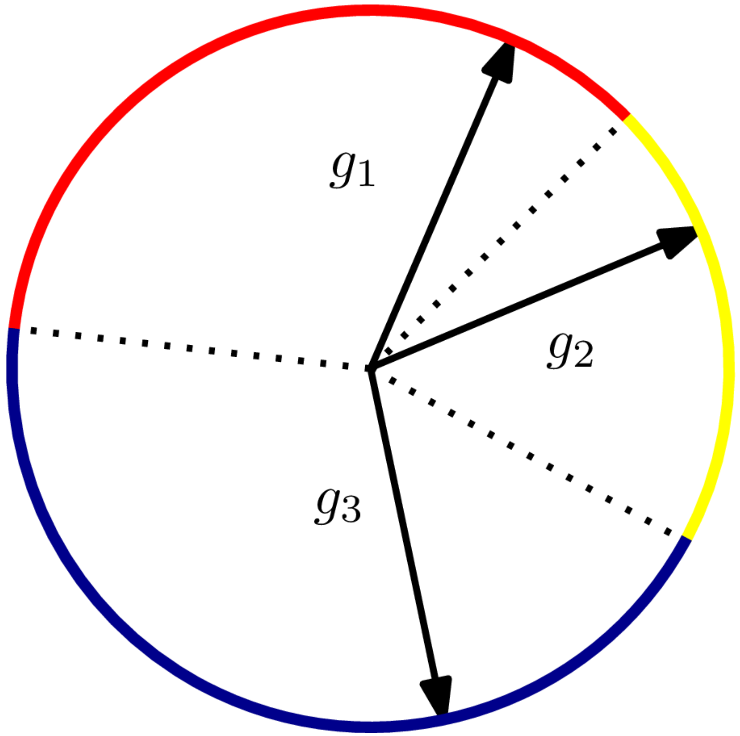


Optimal LSH family: Voronoi LSH

- From **[Andoni, Indyk, Nguyen, R 2014]**, **[Andoni, R 2015]**; inspired by **[Karger, Motwani, Sudan 1998]**: **Voronoi LSH**
- Sample **T** i.i.d. standard **d**-dimensional Gaussians

$$\mathbf{g_1, g_2, \ldots, g_T}$$

- Hash **p** into $\mathbf{h(p) = argmax_{1 \le i \le T} \langle p, g_i \rangle}$
- **T = 2** is simply Hyperplane LSH



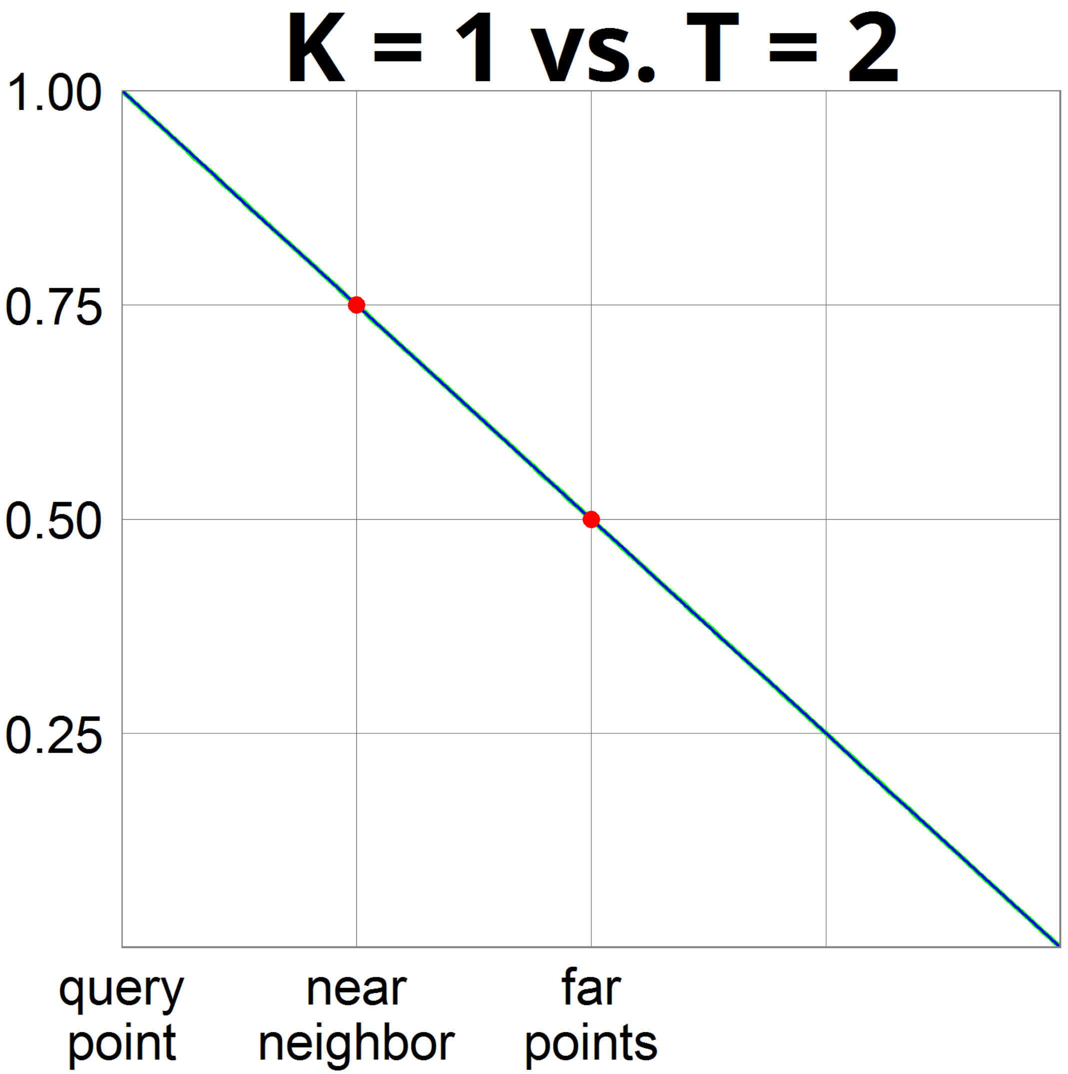
Hyperplane LSH vs. Voronoi LSH

Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with $\mathbf{T = 2^K}$ (in both cases **K**-bit hashes)

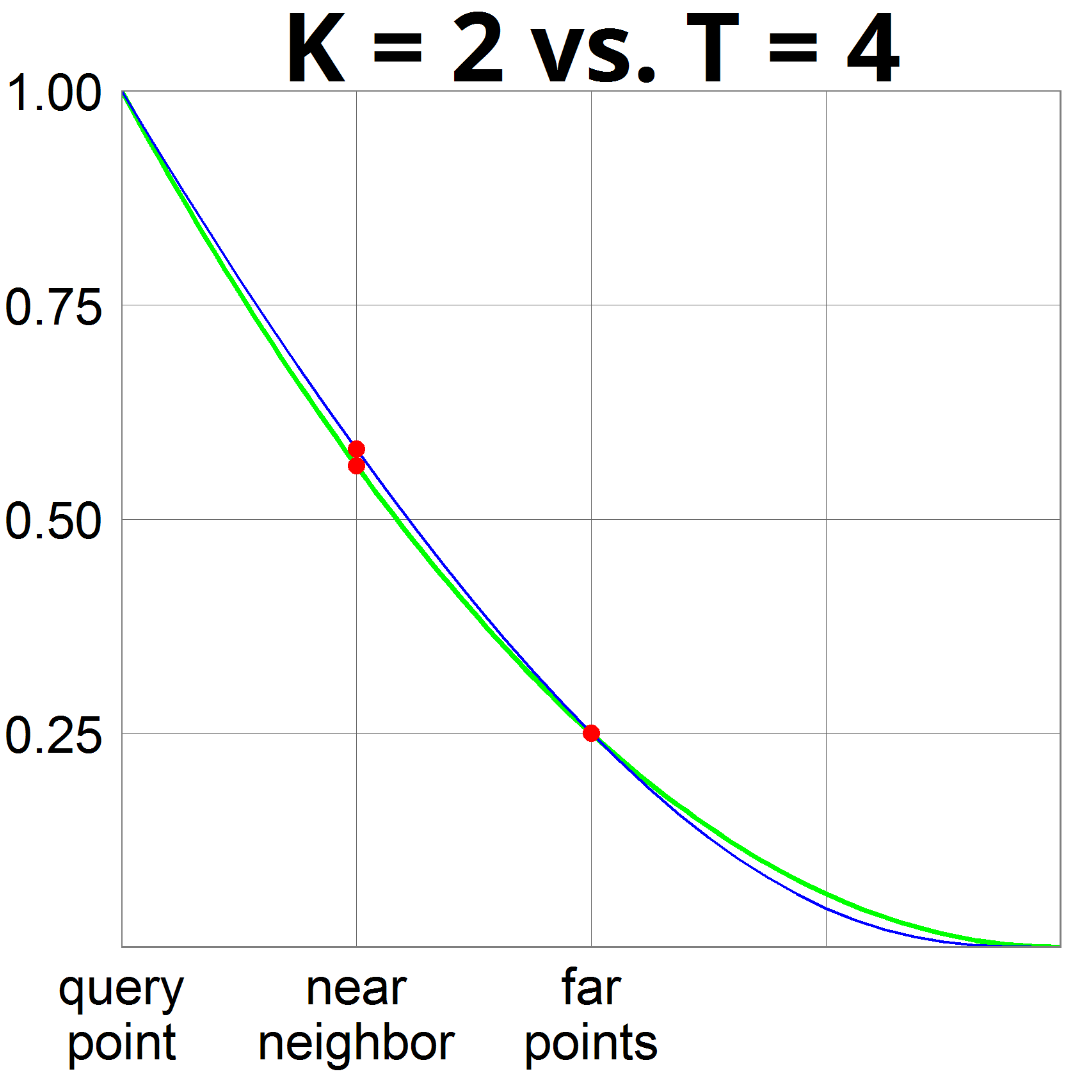
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

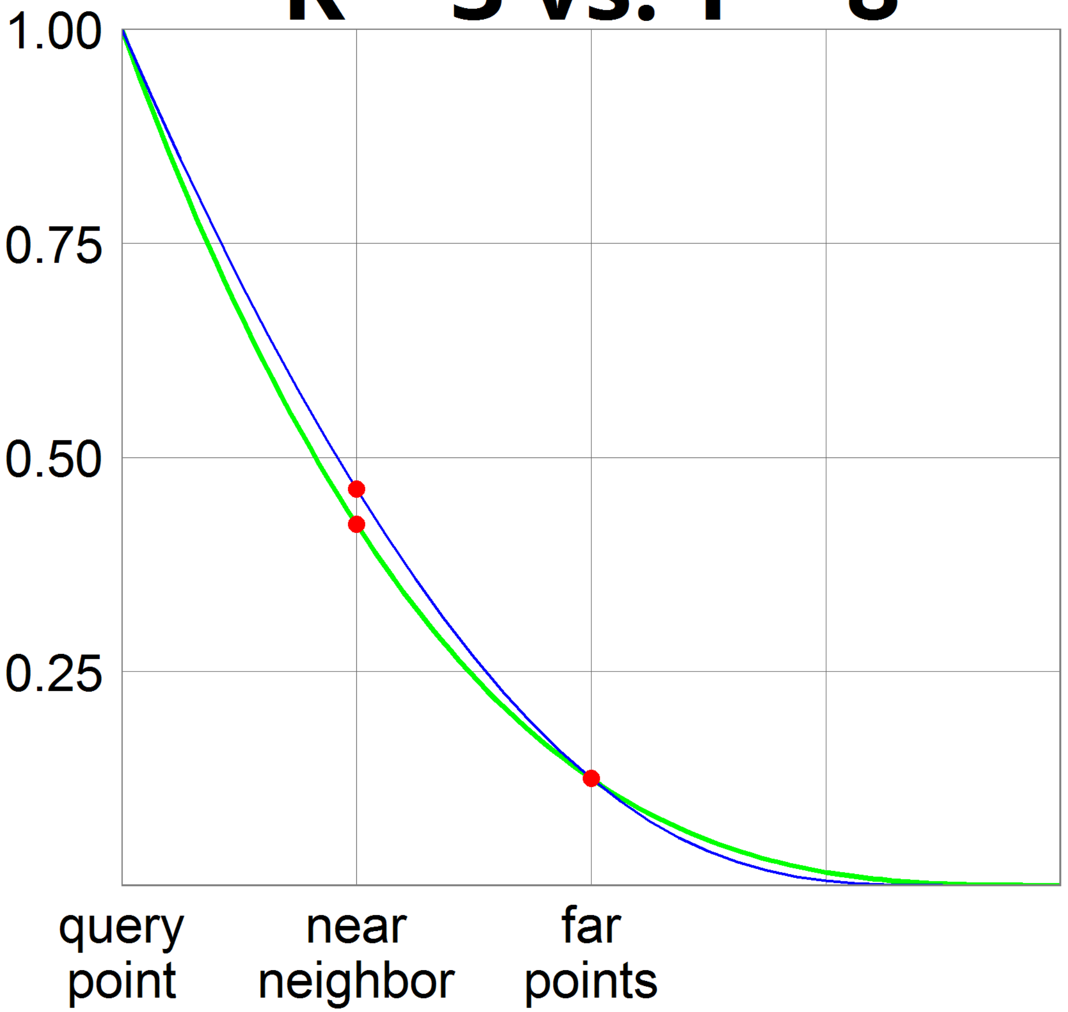
- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

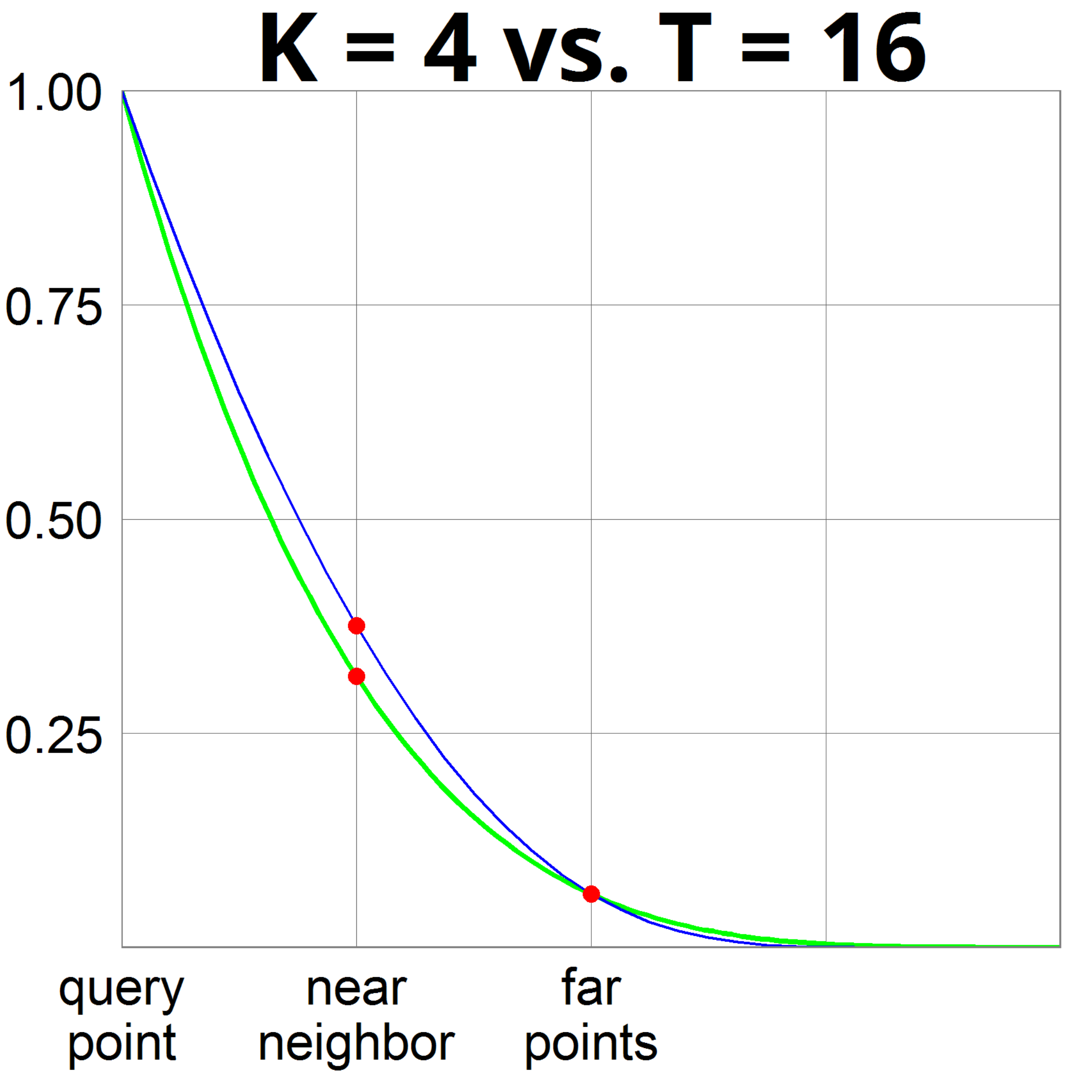
- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)

K = 3 vs. T = 8



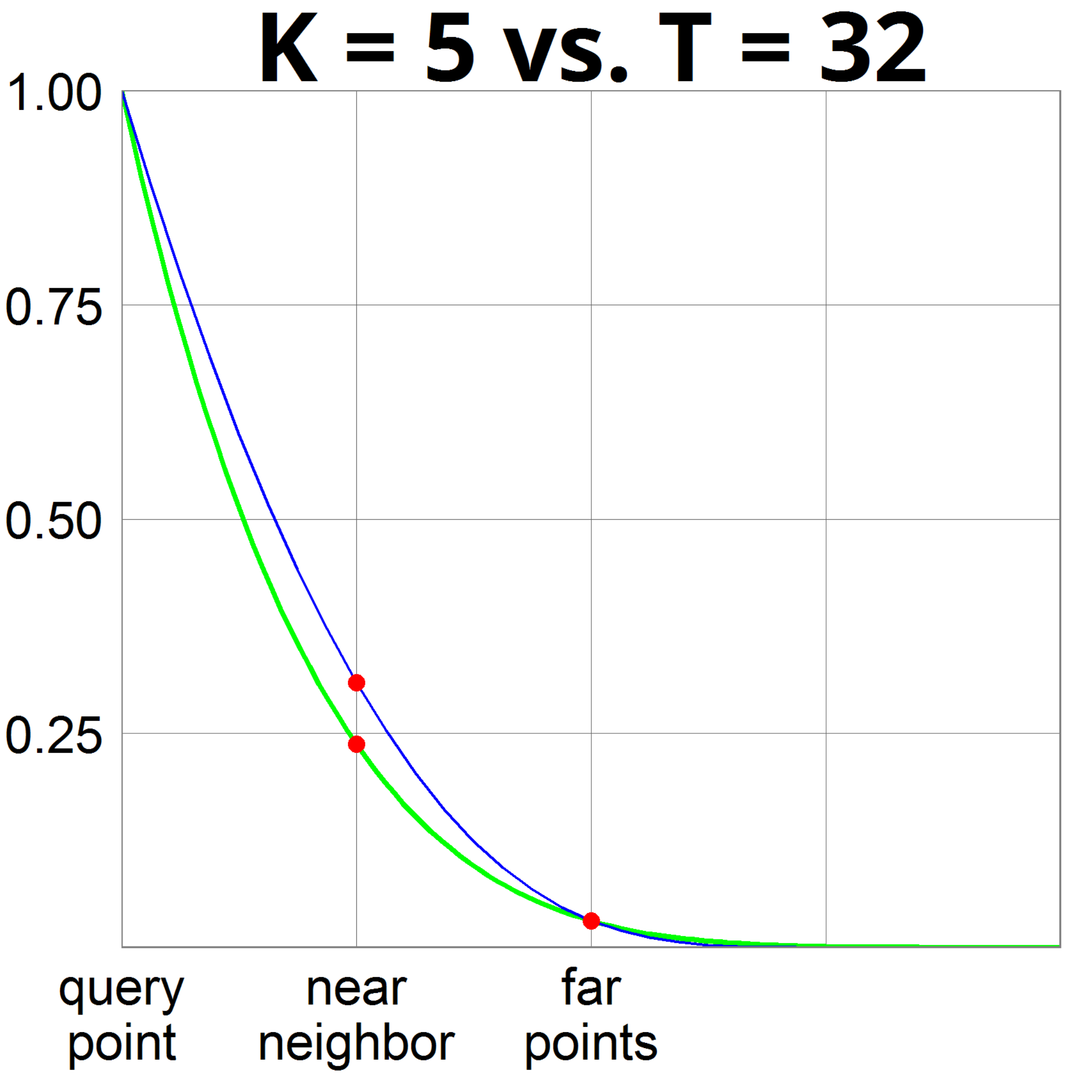
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)



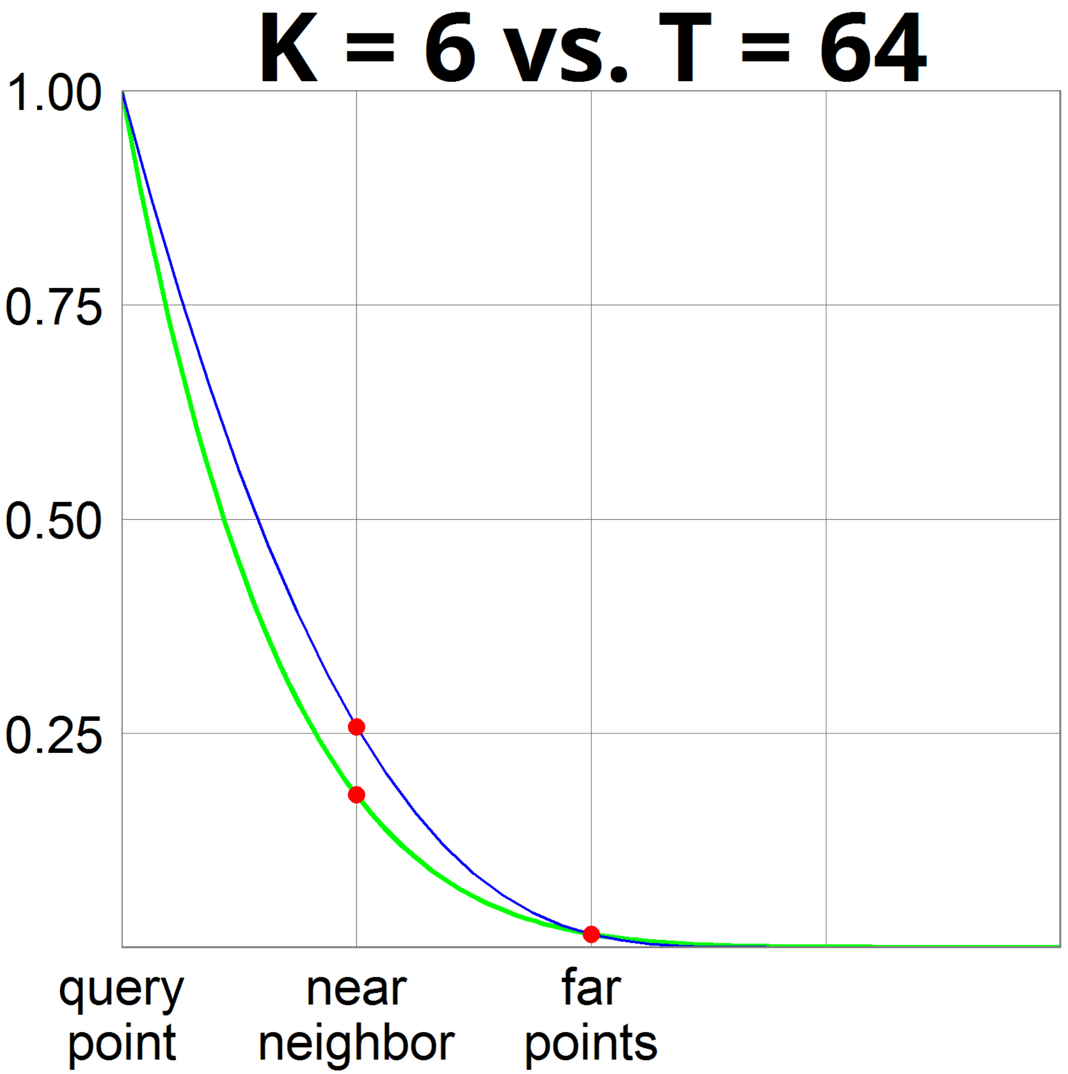
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^{K}** (in both cases **K**-bit hashes)



Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^{K}** (in both cases **K**-bit hashes)



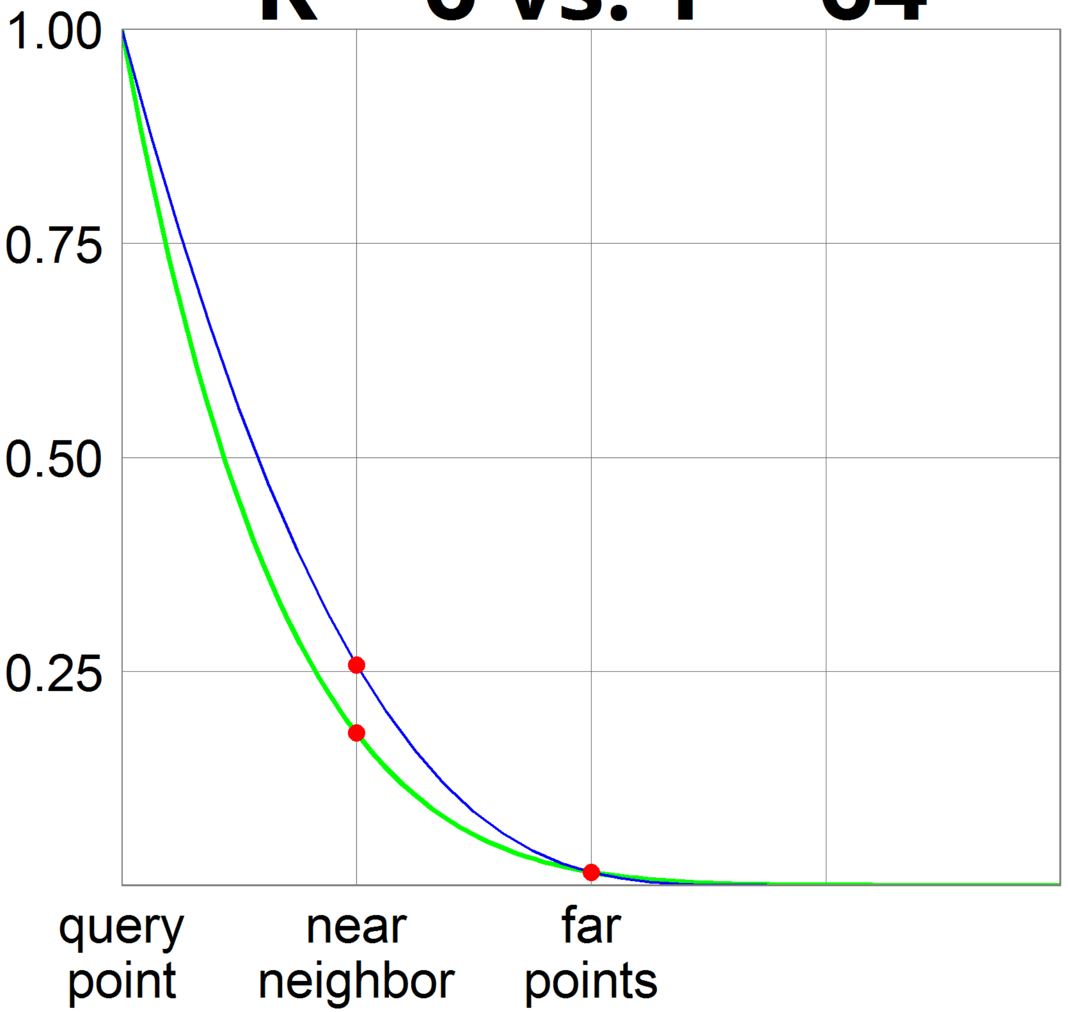
Hyperplane LSH vs. Voronoi LSH

- Let us compare **K** hyperplanes vs. Voronoi LSH with **T = 2^K** (in both cases **K**-bit hashes)
- As **T** grows, the gap between Hyperplane LSH and Voronoi LSH increases and

$$\rho = \ln(1/p_1) / \ln(1/p_2)$$

approaches **0.18**

K = 6 vs. T = 64



Practicality

Practicality

Is Voronoi LSH practical?

Practicality

Is Voronoi LSH practical?

No!

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement
- Takes **$O(d \cdot T)$** time (even say **$T = 64$** is bad)

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: **$\Theta(1 / \log T)$**
- Large **T** to notice any improvement
- Takes **$O(d \cdot T)$** time (even say **$T = 64$** is bad)

At the same time:

- Hyperplane LSH is *very* useful in practice
- Can practice benefit from theory?

Practicality

Is Voronoi LSH practical?

No!

- *Slow* convergence to the optimal exponent: $\boldsymbol{\Theta(1 / \log T)}$
- Large $\mathbf{T}$ to notice any improvement
- Takes $\mathbf{O(d \cdot T)}$ time (even say $\mathbf{T = 64}$ is bad)

At the same time:

- Hyperplane LSH is *very* useful in practice
- Can practice benefit from theory?

This work: yes!

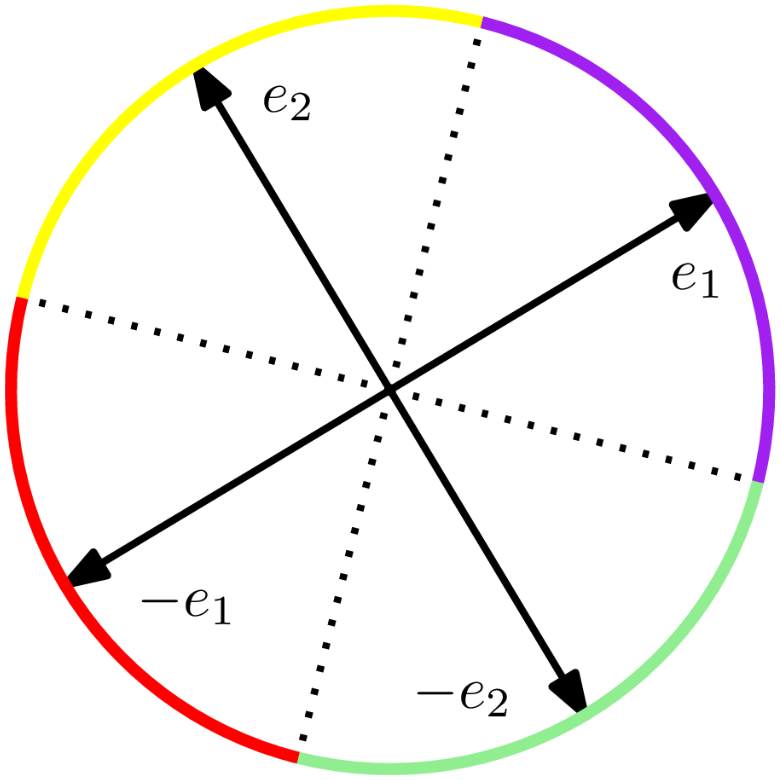
First idea: Cross-polytope LSH

First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **$\{\pm e_i\}$** closest to **Sp**

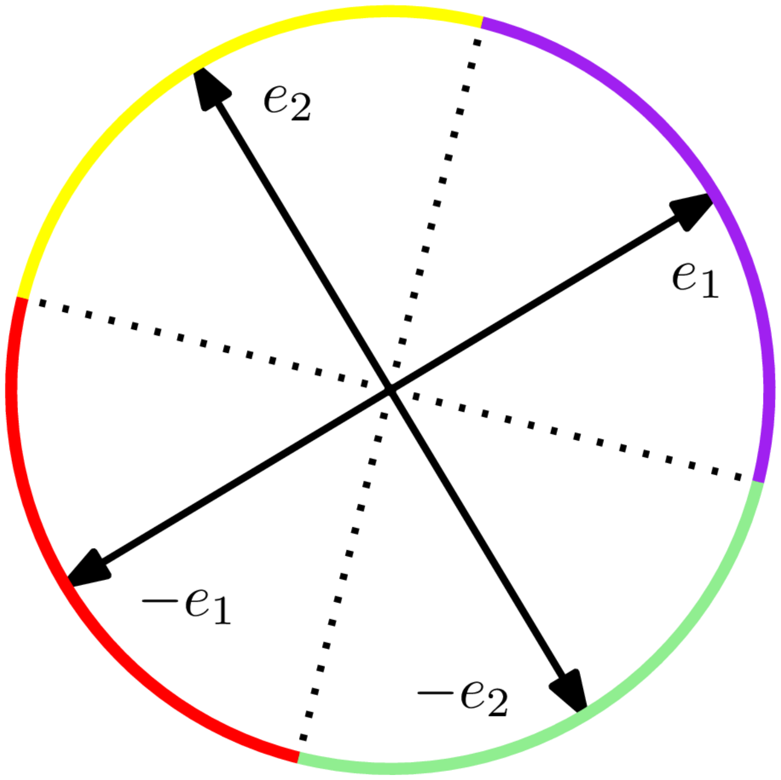
First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **{±e$_i$}** closest to **Sp**



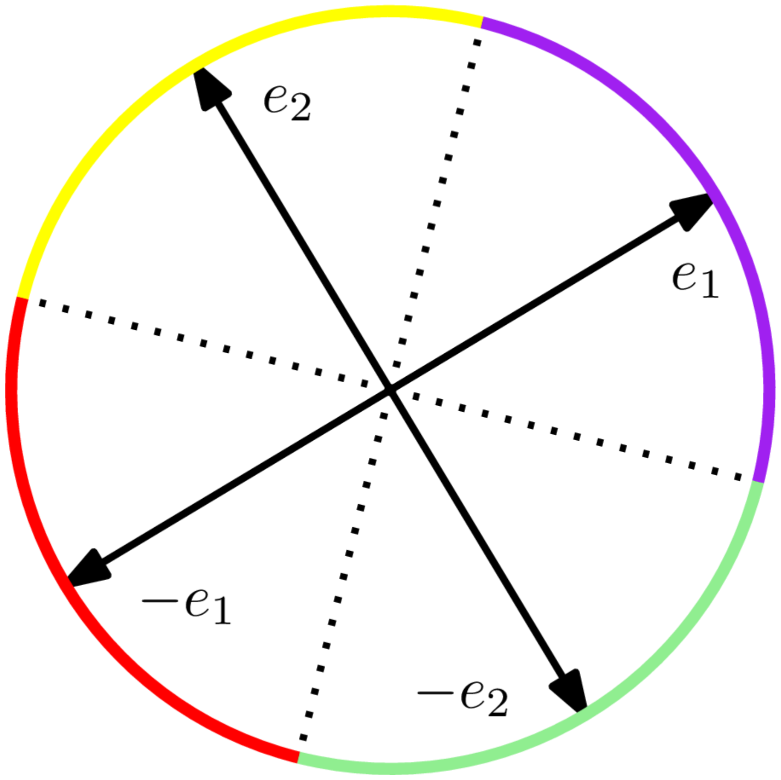
First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **$\{\pm e_i\}$** closest to **Sp**
- Almost the same quality as Voronoi LSH with **T = 2d**
  - *Blessing of dimensionality:* exponent improves as **d** grows!



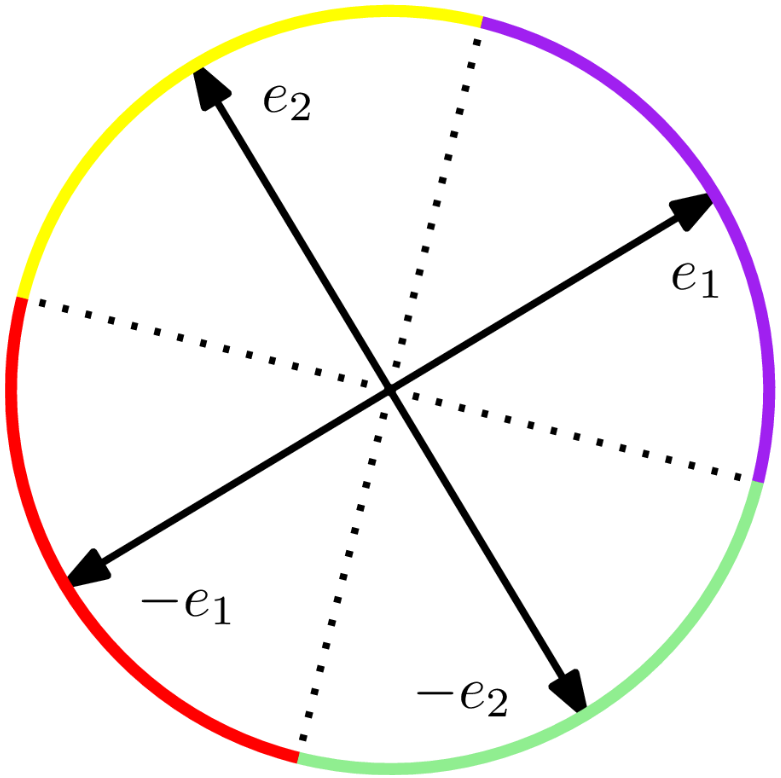
First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **$\{\pm e_i\}$** closest to **Sp**
- Almost the same quality as Voronoi LSH with **T = 2d**
  - *Blessing of dimensionality:* exponent improves as **d** grows!
- *Impractical:* a random rotation costs **$O(d^2)$** time and space



First idea: Cross-polytope LSH

- Cross-polytope LSH introduced by **[Terasawa, Tanaka 2007]:**
  - To hash **p**, apply a *random rotation* **S** to **p**
  - Set hash value to a vertex of a cross-polytope **{±e_i}** closest to **Sp**
- Almost the same quality as Voronoi LSH with **T = 2d**
  - *Blessing of dimensionality:* exponent improves as **d** grows!
- *Impractical:* a random rotation costs **O(d^2)** time and space
- The second step is cheap (only **O(d)** time)



Second idea: *pseudo*-random rotations

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009]**, used in **[Dasgupta, Kumar, Sarlos 2011]**, **[Ailon, Rauhut 2014]**, **[Ve, Sarlos, Smola, 2013]** etc

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009]**, used in **[Dasgupta, Kumar, Sarlos 2011]**, **[Ailon, Rauhut 2014]**, **[Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009]**, used in **[Dasgupta, Kumar, Sarlos 2011]**, **[Ailon, Rauhut 2014]**, **[Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!
- **Hadamard transform:** an orthogonal map that
  - "Mixes well"
  - **Fast:** can be computed in time **O(d log d)**

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009], used in [Dasgupta, Kumar, Sarlos 2011], [Ailon, Rauhut 2014], [Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!
- **Hadamard transform:** an orthogonal map that
  - "Mixes well"
  - **Fast:** can be computed in time **O(d log d)**

$$H_0 = 1$$

$$H_n = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}$$

Second idea: *pseudo*-random rotations

- Introduced in **[Ailon, Chazelle 2009]**, used in **[Dasgupta, Kumar, Sarlos 2011]**, **[Ailon, Rauhut 2014]**, **[Ve, Sarlos, Smola, 2013]** etc
- True random rotations are expensive!
- **Hadamard transform:** an orthogonal map that
  - "Mixes well"
  - **Fast:** can be computed in time **O(d log d)**

$p = (p_1, p_2, \ldots, p_n)$

Flip signs

$p' = (\pm p_1, \pm p_2, \ldots, \pm p_n)$

Hadamard

Hp'

Repeat (2-3 times)

$$H_0 = 1$$

$$H_n = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{pmatrix}$$

Overall hashing scheme

Overall hashing scheme

- Perform **2–3** rounds of “flip signs / Hadamard”

Overall hashing scheme

- Perform **2–3** rounds of “flip signs / Hadamard”
- Find the closest vector from $\{\pm e_i\}$ (maximum coordinate)

Overall hashing scheme

- Perform **2–3** rounds of “flip signs / Hadamard”
- Find the closest vector from $\{\pm e_i\}$ (maximum coordinate)
- Evaluation time **$O(d \log d)$**

Overall hashing scheme

- Perform **2–3** rounds of “flip signs / Hadamard”
- Find the closest vector from **$\{\pm e_i\}$** (maximum coordinate)
- Evaluation time **$O(d \log d)$**
- Equivalent to Voronoi LSH with **$T = 2d$** Gaussians

Memory consumption

Memory consumption

- *LSH consumes lots of memory:* myth or reality?

Memory consumption

- *LSH consumes lots of memory:* myth or reality?
- For **$n = 10^6$** random points and queries within **45** degrees, need **725 tables** for success probability **0.9** (if using Hyperplane LSH)

Memory consumption

- *LSH consumes lots of memory:* myth or reality?
- For **$n = 10^6$** random points and queries within **45** degrees, need **725 tables** for success probability **0.9** (if using Hyperplane LSH)
- Can be reduced substantially via **Multiprobe LSH** **[Lv, Josephson, Wang, Charikar, Li 2007]**

Third idea: Multiprobe LSH

Third idea: Multiprobe LSH

- Instead of trying a single bucket, try **P** buckets, where the near neighbor is most likely to end up

Third idea: Multiprobe LSH

- Instead of trying a single bucket, try **P** buckets, where the near neighbor is most likely to end up
- A single probe: query the bucket

$$(\mathrm{sgn}\,\langle q, r_1\rangle, \mathrm{sgn}\,\langle q, r_2\rangle, \ldots, \mathrm{sgn}\,\langle q, r_K\rangle)$$

Third idea: Multiprobe LSH

- Instead of trying a single bucket, try **P** buckets, where the near neighbor is most likely to end up
- A single probe: query the bucket

$$(\mathrm{sgn}\ \langle q, r_1\rangle, \mathrm{sgn}\ \langle q, r_2\rangle, \ldots, \mathrm{sgn}\ \langle q, r_K\rangle)$$

- To generate **P** buckets, flip signs, for which $\langle q, r_i\rangle$ is close to zero

Third idea: Multiprobe LSH

- Instead of trying a single bucket, try **P** buckets, where the near neighbor is most likely to end up
- A single probe: query the bucket

  $$(\mathrm{sgn}\,\langle q, r_1\rangle, \mathrm{sgn}\,\langle q, r_2\rangle, \ldots, \mathrm{sgn}\,\langle q, r_K\rangle)$$

- To generate **P** buckets, flip signs, for which $\langle q, r_i\rangle$ is close to zero
- By increasing **P**, can reduce **L** (# of tables)

Third idea: Multiprobe LSH

- Instead of trying a single bucket, try **P** buckets, where the near neighbor is most likely to end up
- A single probe: query the bucket

$$\mathbf{(sgn <q, r_1>, sgn <q, r_2>, \ldots, sgn <q, r_K>)}$$

- To generate **P** buckets, flip signs, for which $\mathbf{<q, r_i>}$ is close to zero
- By increasing **P**, can reduce **L** (# of tables)
- A similar procedure for Cross-polytope LSH (more complicated, since the range is non-binary)

Fourth idea: Hashing Trick

Fourth idea: Hashing Trick

- For **s**-sparse vectors, Hyperplane LSH takes time **O(s)**

Fourth idea: Hashing Trick

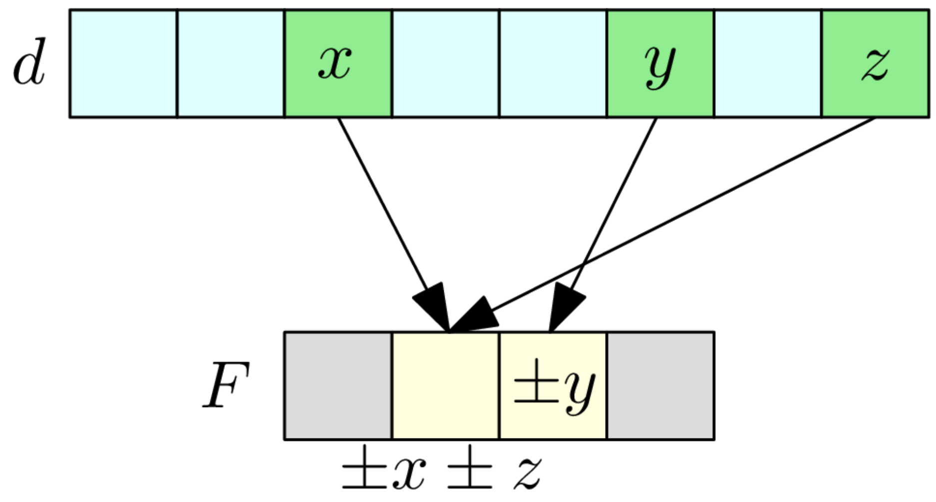
- For **s**-sparse vectors, Hyperplane LSH takes time **O(s)**
- Can Cross-polytope LSH exploit sparsity?

Fourth idea: Hashing Trick

- For **s**-sparse vectors, Hyperplane LSH takes time **$O(s)$**
- Can Cross-polytope LSH exploit sparsity?
- Hashing trick (a.k.a. Count-Sketch)

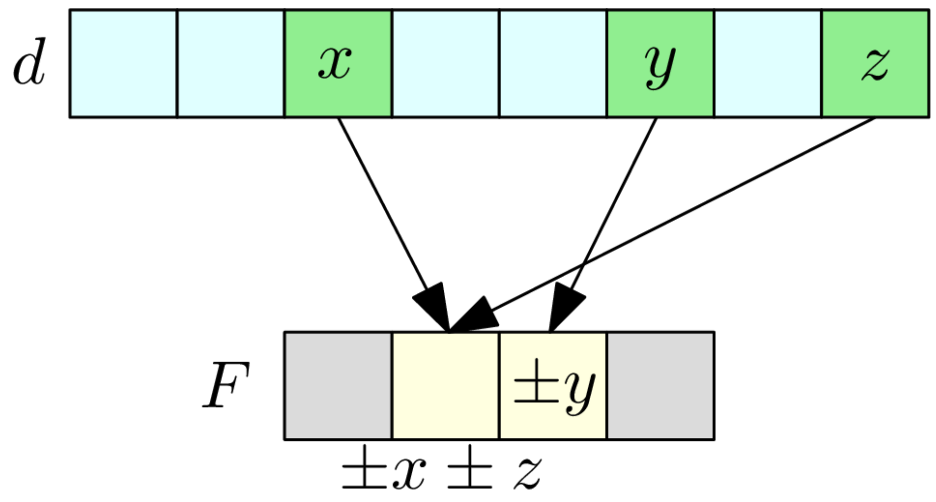
Fourth idea: Hashing Trick

- For **s**-sparse vectors, Hyperplane LSH takes time **O(s)**
- Can Cross-polytope LSH exploit sparsity?
- Hashing trick (a.k.a. Count-Sketch)



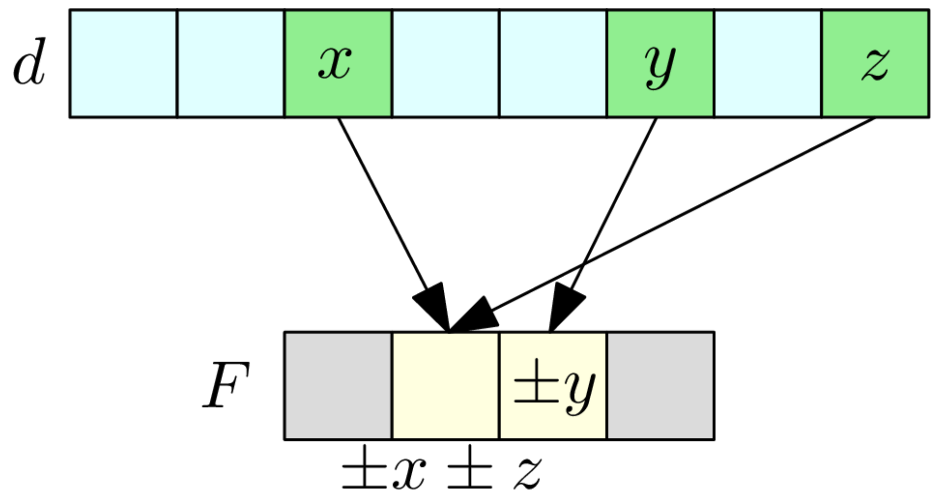
Fourth idea: Hashing Trick

- For **s**-sparse vectors, Hyperplane LSH takes time **O(s)**
- Can Cross-polytope LSH exploit sparsity?
- Hashing trick (a.k.a. Count-Sketch)
- For target dimension **F** yields time **O(s + F log F)**



Fourth idea: Hashing Trick

- For **s**-sparse vectors, Hyperplane LSH takes time **O(s)**
- Can Cross-polytope LSH exploit sparsity?
- Hashing trick (a.k.a. Count-Sketch)
- For target dimension **F** yields time **O(s + F log F)**
- Equivalent to Voronoi LSH with **T = 2F**.



Choosing parameters in practice

Choosing parameters in practice

- Aim at finding the **exact nearest neighbor**

Choosing parameters in practice

- Aim at finding the **exact nearest neighbor**
- Probability of success (**0.9**)

Choosing parameters in practice

- Aim at finding the **exact nearest neighbor**
- Probability of success (**0.9**)
- Intermediate dimension **F** (**~1000**; as large as possible, while not slowing hashing down)

Choosing parameters in practice

- Aim at finding the **exact nearest neighbor**
- Probability of success (**0.9**)
- Intermediate dimension **F** (**~1000**; as large as possible, while not slowing hashing down)
- # of tables **L** (depending on RAM budget, even **~10** would do)

Choosing parameters in practice

- Aim at finding the **exact nearest neighbor**
- Probability of success (**0.9**)
- Intermediate dimension **F** (**~1000**; as large as possible, while not slowing hashing down)
- # of tables **L** (depending on RAM budget, even **~10** would do)
- # of hash functions / table **K** (few data points in most of the buckets)

Choosing parameters in practice

- Aim at finding the **exact nearest neighbor**
- Probability of success (**0.9**)
- Intermediate dimension **F** (**~1000**; as large as possible, while not slowing hashing down)
- # of tables **L** (depending on RAM budget, even **~10** would do)
- # of hash functions / table **K** (few data points in most of the buckets)
- Determine # of probes **P** that gives the desired probability of success (on sample queries)

Implementation details

Implementation details

- An actual implementation of Multiprobe Hyperplane and Cross-polytope LSH in C++11, 11k LOC, template-based

Implementation details

- An actual implementation of Multiprobe Hyperplane and Cross-polytope LSH in C++11, 11k LOC, template-based
- Supports dense and sparse data

Implementation details

- An actual implementation of Multiprobe Hyperplane and Cross-polytope LSH in C++11, 11k LOC, template-based
- Supports dense and sparse data
- Very polished (w.r.t. performance)
  - Uses Eigen to speed-up hash and distance computations
  - Vectorized Hadamard transform (using AVX), several times faster than FFTW (surprise!)

Implementation details

- An actual implementation of Multiprobe Hyperplane and Cross-polytope LSH in C++11, 11k LOC, template-based
- Supports dense and sparse data
- Very polished (w.r.t. performance)
  - Uses Eigen to speed-up hash and distance computations
  - Vectorized Hadamard transform (using AVX), several times faster than FFTW (surprise!)
- Available at **http://falconn-lib.org** together with Python bindings
  - **http://github.com/falconn-lib/ffht** for FHT

Experiments: the set-up

Experiments: the set-up

- Success probability **0.9** for finding exact nearest neighbors

Experiments: the set-up

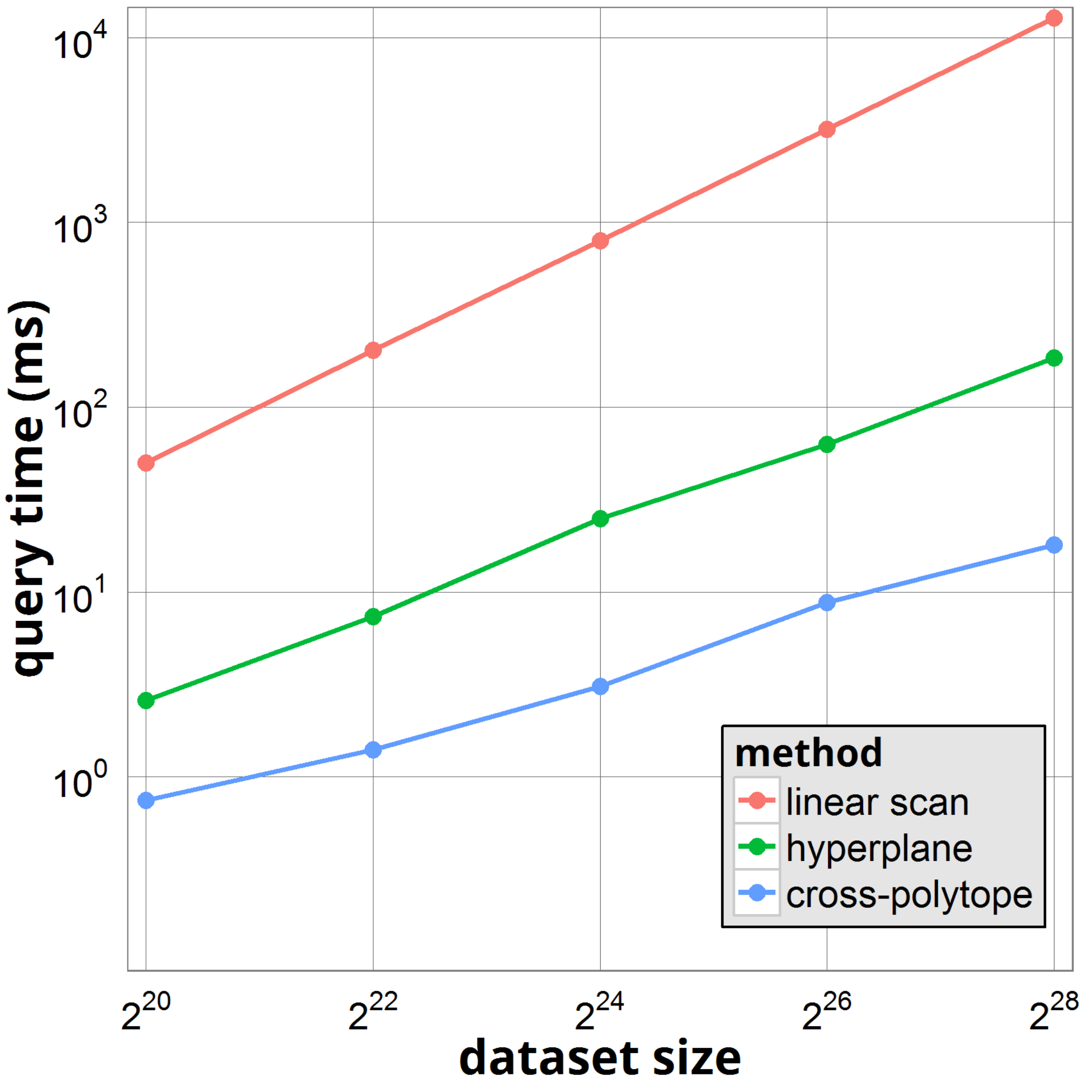
- Success probability **0.9** for finding exact nearest neighbors
- Choose **L** s.t. space for tables ≈ space for a dataset (except one instance)

Experiments: the set-up

- Success probability **0.9** for finding exact nearest neighbors
- Choose **L** s.t. space for tables ≈ space for a dataset (except one instance)
- (Optimized) linear scan vs. Hyperplane vs. Cross-polytope

Experiments: random data (d = 128)

Experiments: random data (d = 128)



Experiments: ANN_SIFT1M

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M, d = 128**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**
- Hyperplane: **3.7ms**, Cross-polytope: **3.1ms**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **n = 1M**, **d = 128**
- Linear scan: **38ms**
- Hyperplane: **3.7ms**, Cross-polytope: **3.1ms**
- Clustering and re-centering helps
  - Hyperplane: **2.75ms**
  - Cross-polytope: **1.75ms**

Experiments: ANN_SIFT1M

- SIFT features for a dataset of images
- **$n = 1M$, $d = 128$**
- Linear scan: **38ms**
- Hyperplane: **3.7ms**, Cross-polytope: **3.1ms**
- Clustering and re-centering helps
  - Hyperplane: **2.75ms**
  - Cross-polytope: **1.75ms**
- Adding more memory helps

Experiments: Pubmed

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity
- **n = 8.2M**, **d = 140k**, average sparsity **90**

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity
- **$n = 8.2$M**, **$d = 140$k**, average sparsity **90**
- Need the hashing trick (down to **2048** dimensions)

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity
- **n = 8.2M**, **d = 140k**, average sparsity **90**
- Need the hashing trick (down to **2048** dimensions)
- Filter "interesting" queries

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity
- **n = 8.2M**, **d = 140k**, average sparsity **90**
- Need the hashing trick (down to **2048** dimensions)
- Filter “interesting” queries
- Linear scan: **3.6s**

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity
- **n = 8.2M**, **d = 140k**, average sparsity **90**
- Need the hashing trick (down to **2048** dimensions)
- Filter “interesting” queries
- Linear scan: **3.6s**
- Hyperplane: **857ms**, Cross-polytope: **213ms**

Experiments: Pubmed

- Bag of words dataset of Pubmed abstracts
- TF-IDF vectors with cosine similarity
- **n = 8.2M**, **d = 140k**, average sparsity **90**
- Need the hashing trick (down to **2048** dimensions)
- Filter "interesting" queries
- Linear scan: **3.6s**
- Hyperplane: **857ms**, Cross-polytope: **213ms**
- Adding more memory helps

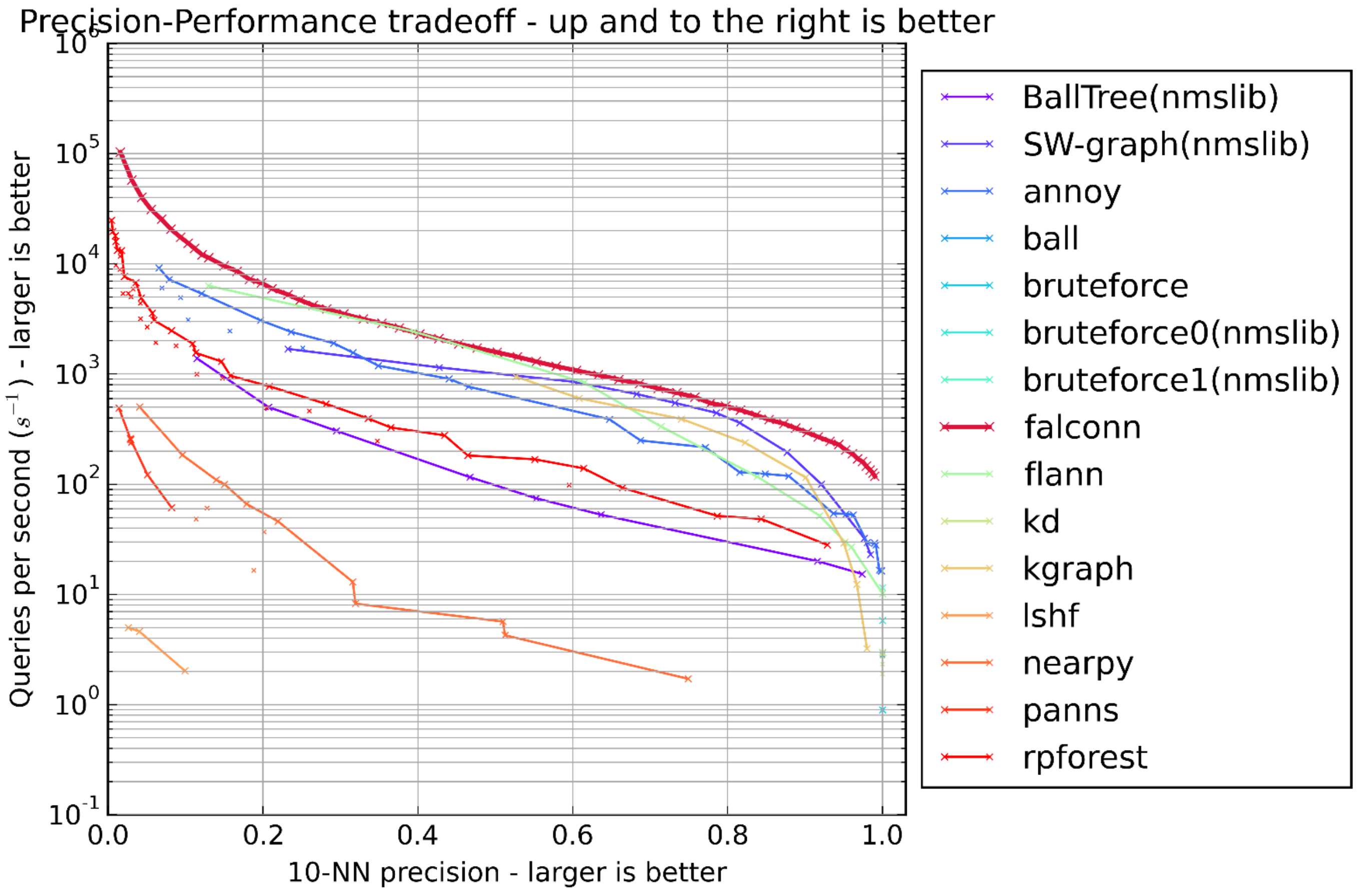
Experiments: GloVe

Experiments: GloVe

[Pennington, Socher, Manning 2014] **n = 1.2M**, **d = 100**, aim at **10** nearest neighbors

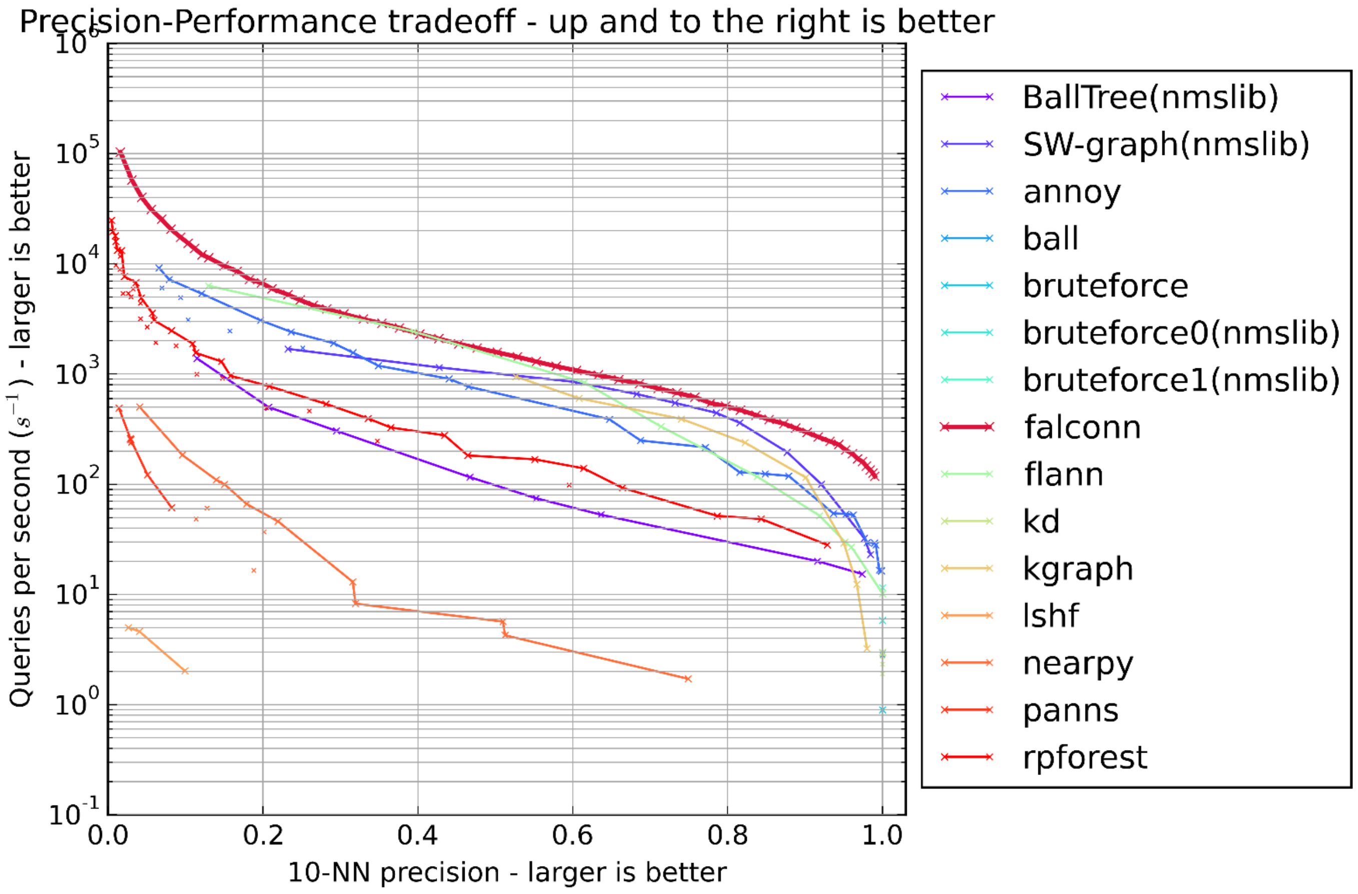
Experiments: GloVe

[Pennington, Socher, Manning 2014] n = 1.2M, d = 100, aim at **10** nearest neighbors



Experiments: GloVe

[Pennington, Socher, Manning 2014] **n = 1.2M**, **d = 100**, aim at **10** nearest neighbors



- **16**-bit hashes
- **1…1400** tables
- Single probe
- Accuracy **0.016…0.99**
- **10μs** to **8.5ms** query
- From **5 Mb** to **7 Gb**

Additional tricks

Additional tricks

- Centering
  - Hierarchical centering?

Additional tricks

- Centering
  - Hierarchical centering?
- “Compressed” index

Additional tricks

- Centering
  - Hierarchical centering?
- “Compressed” index
- Data prefetching

Additional tricks

- Centering
  - Hierarchical centering?
- “Compressed” index
- Data prefetching
- Sorting is expensive

Lower bound

Lower bound

- The convergence to the optimal exponent is **$\Theta(1 / \log T)$**

Lower bound

- The convergence to the optimal exponent is **$\Theta(1 / \log T)$**
- Tight for any LSH!

Lower bound

- The convergence to the optimal exponent is **$\Theta(1 / \log T)$**
- Tight for any LSH!
- Any LSH family with range of size **S** must be at least **$\Omega(1 / \log S)$** off the optimum

Lower bound

- The convergence to the optimal exponent is **$\Theta(1 / \log T)$**
- Tight for any LSH!
- Any LSH family with range of size **S** must be at least **$\Omega(1 / \log S)$** off the optimum
- For **45**-degree random instance:
  - The best exponent is **0.18**
  - To get below **0.2**, need **$S \geq 10^{12}$**

Lower bound

- The convergence to the optimal exponent is **$\Theta(1 / \log T)$**
- Tight for any LSH!
- Any LSH family with range of size **S** must be at least **$\Omega(1 / \log S)$** off the optimum
- For **45**-degree random instance:
  - The best exponent is **0.18**
  - To get below **0.2**, need **$S \geq 10^{12}$**
- For the further progress, need evaluation time sublinear in the range size!
  - Complexity of “decoding” for almost-orthogonal vectors

ANN with fast query time via sketches

Sketching metrics

Sketching metrics

- **Alice** and **Bob** each hold a point from a metric space (say x and y)

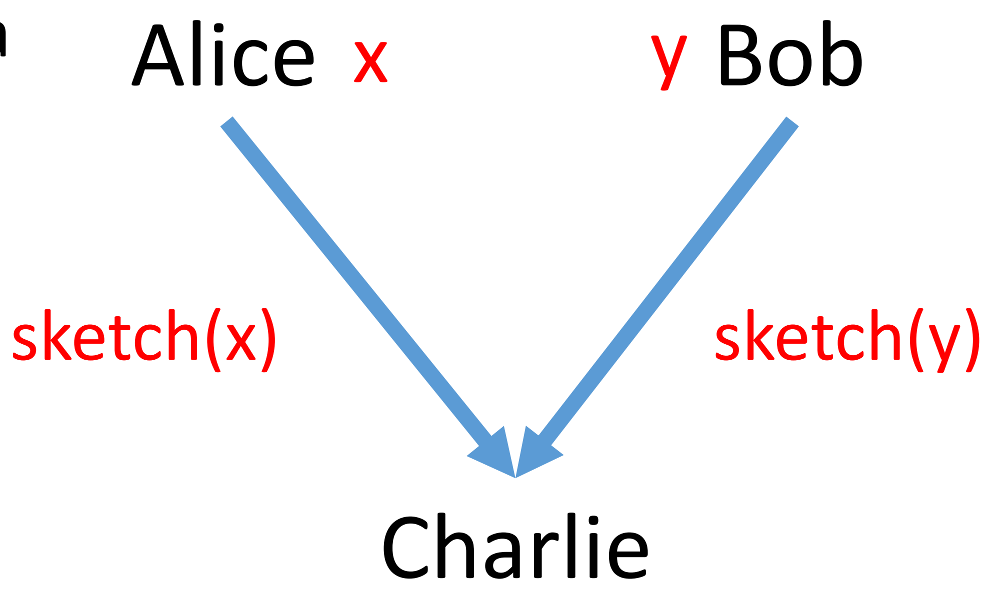
Alice x

y Bob

Charlie

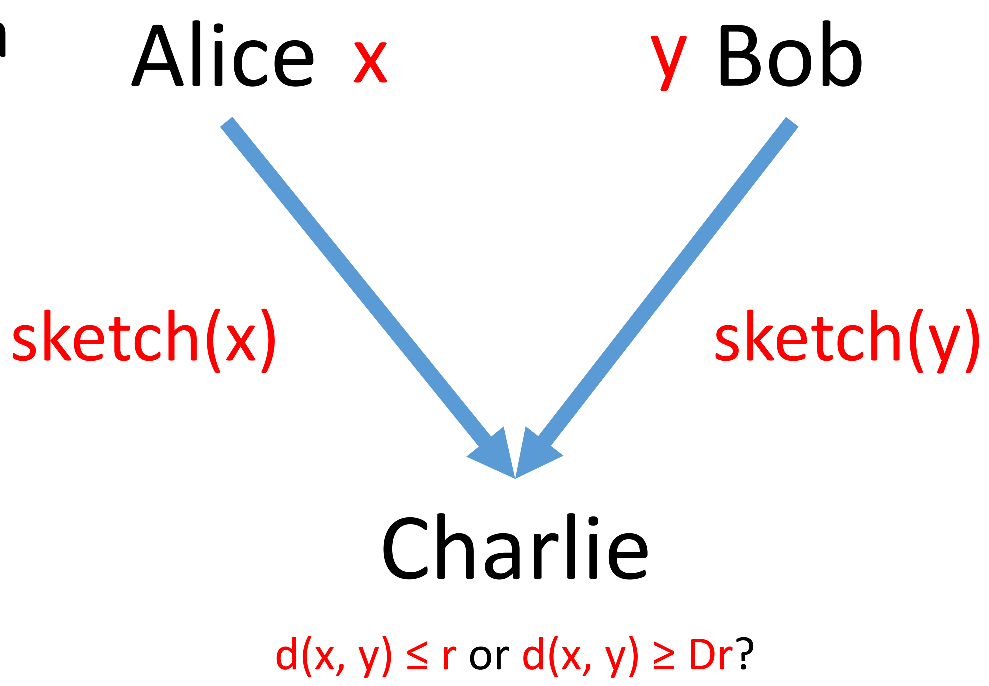
Sketching metrics

- **Alice** and **Bob** each hold a point from a metric space (say x and y)
- Both send s-bit *sketches* to **Charlie**



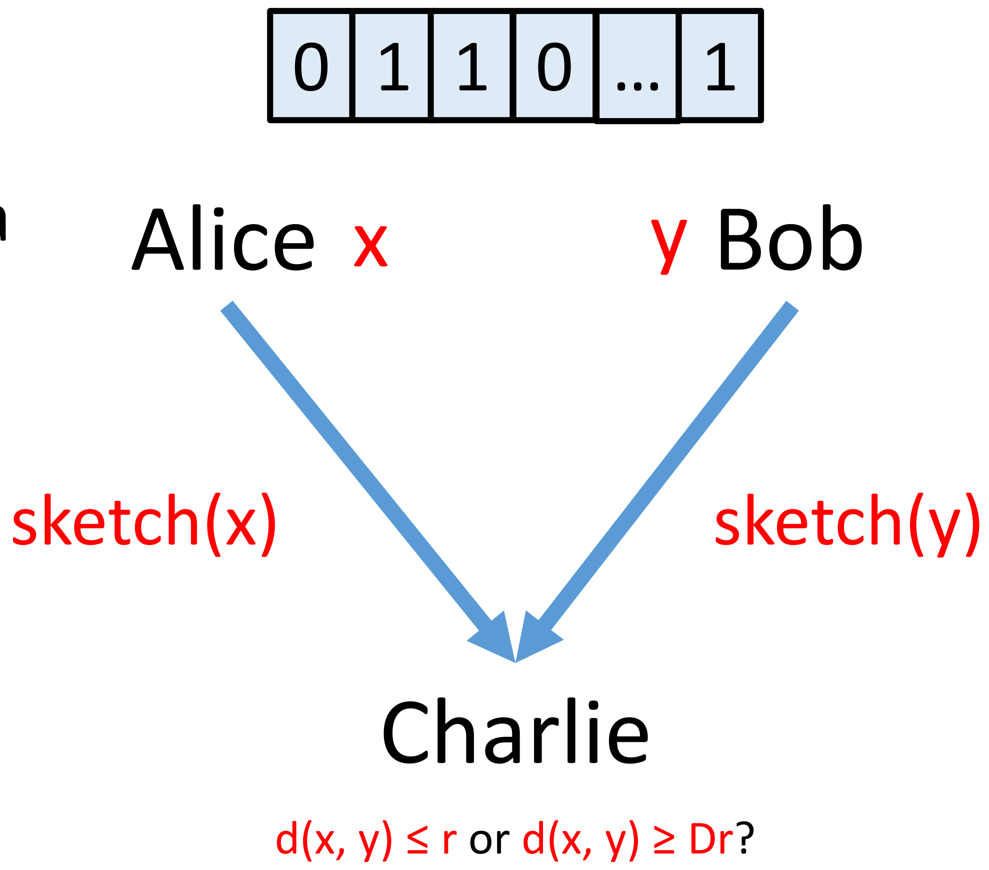
Sketching metrics

- **Alice** and **Bob** each hold a point from a metric space (say x and y)
- Both send s-bit *sketches* to **Charlie**
- For $r > 0$ and $D > 1$ distinguish
  - $d(x, y) \leq r$
  - $d(x, y) \geq Dr$



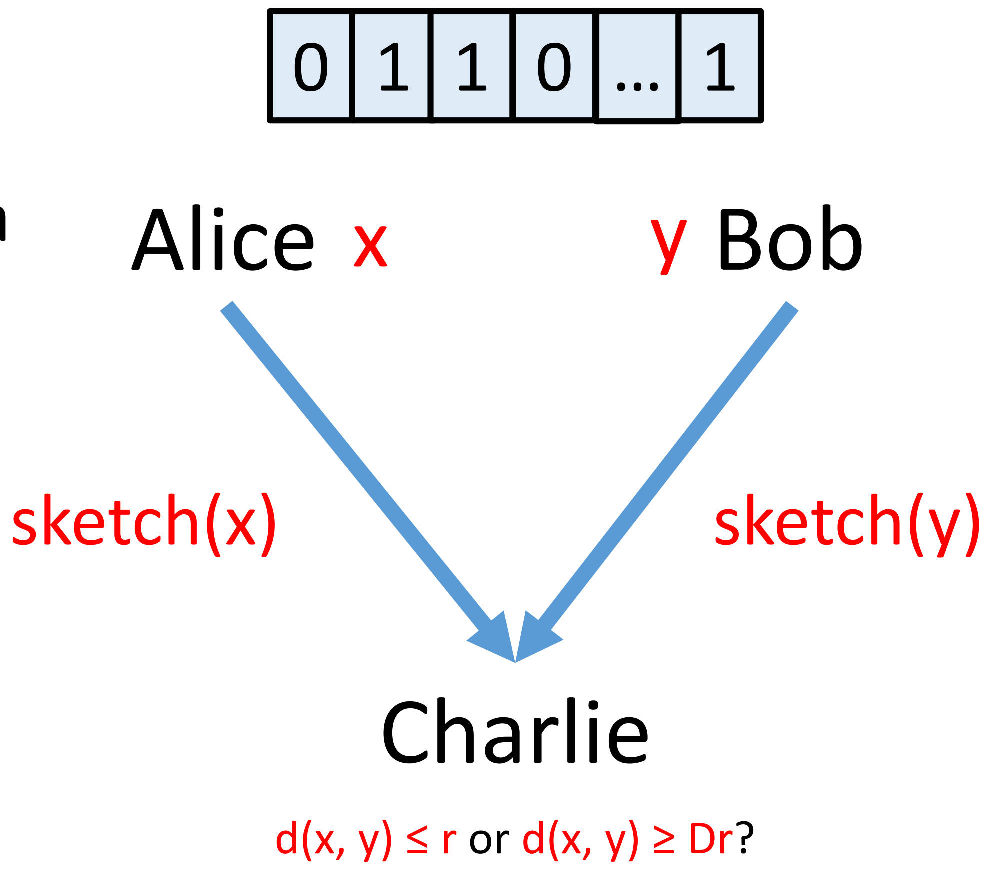
Sketching metrics

- **Alice** and **Bob** each hold a point from a metric space (say x and y)
- Both send s-bit *sketches* to **Charlie**
- For $r > 0$ and $D > 1$ distinguish
  - $d(x, y) \leq r$
  - $d(x, y) \geq Dr$
- Shared randomness, allow 1% probability of error



Sketching metrics

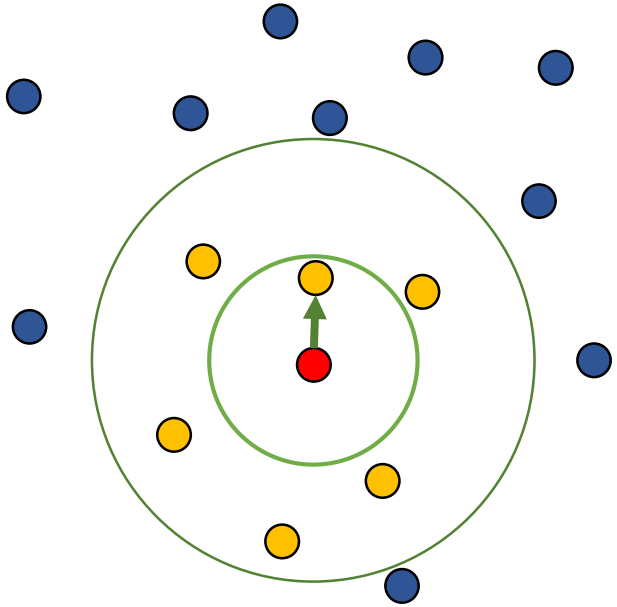
- **Alice** and **Bob** each hold a point from a metric space (say x and y)
- Both send s-bit *sketches* to **Charlie**
- For $r > 0$ and $D > 1$ distinguish
  - $d(x, y) \le r$
  - $d(x, y) \ge Dr$
- Shared randomness, allow 1% probability of error
- **Trade-off between s and D**



Near Neighbor Search via sketches

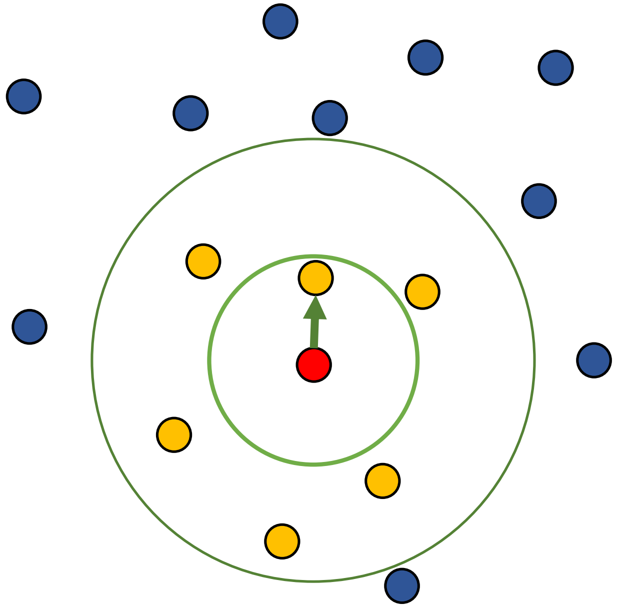
Near Neighbor Search via sketches

- **Near Neighbor Search** (NNS):
  - Given n-point dataset P
  - A query q within r from some data point
  - Return any data point within Dr from q



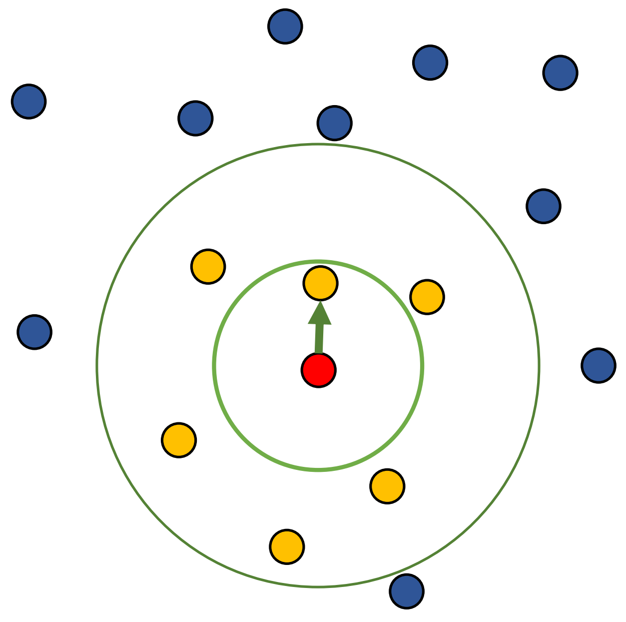
Near Neighbor Search via sketches

- **Near Neighbor Search** (NNS):
  - Given n-point dataset P
  - A query q within r from some data point
  - Return any data point within Dr from q
- Sketches of size s imply NNS with space $n^{O(s)}$ and a 1-probe query



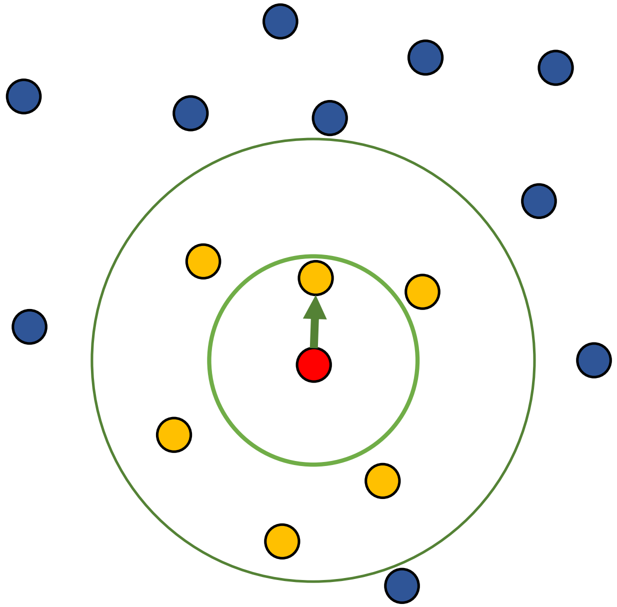
Near Neighbor Search via sketches

- **Near Neighbor Search** (NNS):
  - Given n-point dataset P
  - A query q within r from some data point
  - Return any data point within Dr from q
- Sketches of size s imply NNS with space $n^{O(s)}$ and a 1-probe query
- **Proof idea**: amplify probability of error to $1/n$ by increasing the size to $O(s \log n)$; sketch of q determines the answer



Near Neighbor Search via sketches

- **Near Neighbor Search** (NNS):
  - Given n-point dataset P
  - A query q within r from some data point
  - Return any data point within Dr from q
- Sketches of size s imply NNS with space $n^{O(s)}$ and a 1-probe query
- **Proof idea**: amplify probability of error to $1/n$ by increasing the size to $O(s \log n)$; sketch of q determines the answer
- For many metrics: the only approach



Sketching ℓ_p norms

Sketching ℓ_p norms

- (Indyk 2000): can sketch ℓ_p for $0 < p \leq 2$ via random projections using **p-stable distributions**
  - For $D = 1 + \varepsilon$ one gets $s = O(1 / \varepsilon^2)$
  - Tight by (Woodruff 2004)

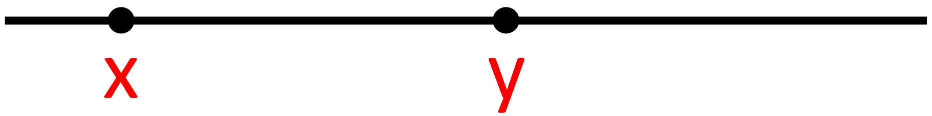
Sketching ℓ_p norms

- (Indyk 2000): can sketch ℓ_p for $0 < p \leq 2$ via random projections using **p-stable distributions**
  - For $D = 1 + \varepsilon$ one gets $s = O(1 / \varepsilon^2)$
  - Tight by (Woodruff 2004)
- For $p > 2$ sketching ℓ_p is somewhat hard (Bar-Yossef, Jayram, Kumar, Sivakumar 2002), (Indyk, Woodruff 2005)
  - To achieve $D = O(1)$ one needs sketch size to be $s = \Theta\sim(d^{1-2/p})$

Sketching real line

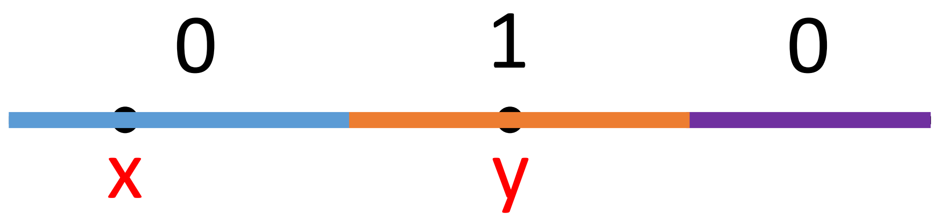
Sketching real line

- Distinguish $|x - y| \leq 1$ vs. $|x - y| \geq 1 + \varepsilon$



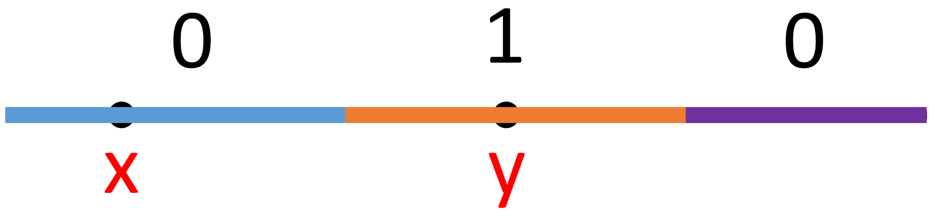
Sketching real line

- Distinguish $|x - y| \leq 1$ vs. $|x - y| \geq 1 + \varepsilon$
- Randomly shifted pieces of size $w = 1 + \varepsilon/2$



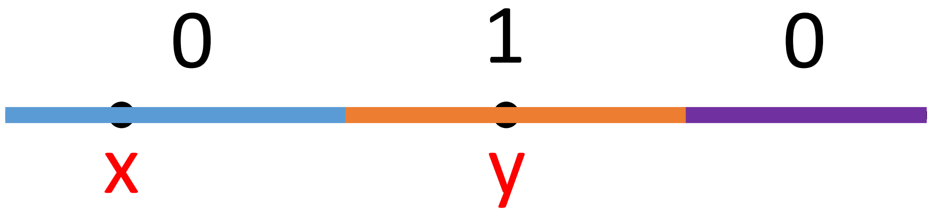
Sketching real line

- Distinguish $|x - y| \le 1$ vs. $|x - y| \ge 1 + \varepsilon$
- Randomly shifted pieces of size $w = 1 + \varepsilon/2$
- Repeat $O(1 / \varepsilon^2)$ times



Sketching real line

- Distinguish $|x - y| \leq 1$ vs. $|x - y| \geq 1 + \varepsilon$
- Randomly shifted pieces of size $w = 1 + \varepsilon/2$
- Repeat $O(1 / \varepsilon^2)$ times
- Overall:
  - $D = 1 + \varepsilon$
  - $s = O(1 / \varepsilon^2)$



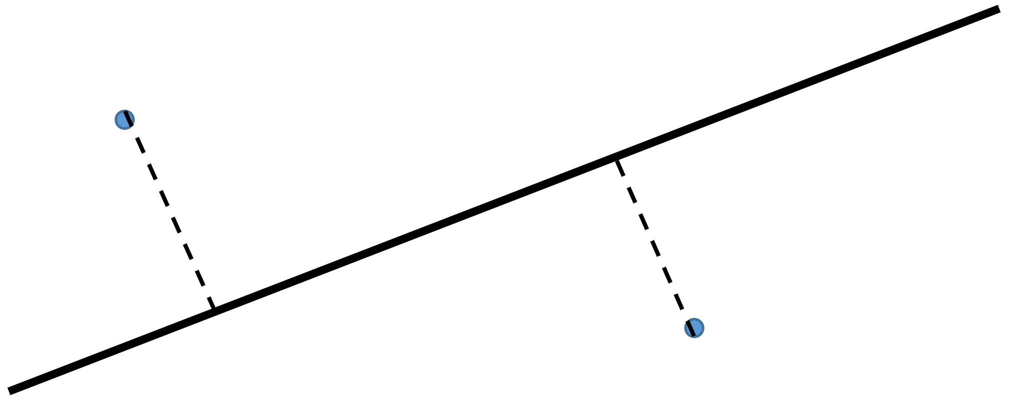
Sketching ℓ_p for $0 < p \leqslant 2$

Sketching ℓ_p for $0 < p \leqslant 2$

- (Indyk 2000): can reduce sketching of ℓ_p with $0 < p \leq 2$ to sketching reals via **random projections**

Sketching ℓ_p for $0 < p \leqslant 2$

- (Indyk 2000): can reduce sketching of ℓ_p with $0 < p \leq 2$ to sketching reals via **random projections**
- If $(G_1, G_2, \ldots, G_d)$ are i.i.d. $N(0, 1)$'s, then $\sum_i x_i G_i - \sum_i y_i G_i$ is distributed as $\|x - y\|_2 \bullet N(0, 1)$



Sketching ℓ_p for $0 < p \leqslant 2$

- (Indyk 2000): can reduce sketching of ℓ_p with $0 < p \leq 2$ to sketching reals via **random projections**
- If $(G_1, G_2, \ldots, G_d)$ are i.i.d. $N(0, 1)$'s, then $\sum_i x_i G_i - \sum_i y_i G_i$ is distributed as $\|x - y\|_2 \bullet N(0, 1)$
- For $0 < p < 2$ use **p-stable distributions** instead

Sketching ℓ_p for $0 < p \leqslant 2$

- (Indyk 2000): can reduce sketching of ℓ_p with $0 < p \leq 2$ to sketching reals via **random projections**
- If $(G_1, G_2, \ldots, G_d)$ are i.i.d. $N(0, 1)$'s, then $\sum_i x_i G_i - \sum_i y_i G_i$ is distributed as $\|x - y\|_2 \bullet N(0, 1)$
- For $0 < p < 2$ use **p-stable distributions** instead
- Again, get $D = 1 + \varepsilon$ with $s = O(1 / \varepsilon^2)$

Summary

- Space: $\mathbf{n^{O(1/\varepsilon^2)}}$
- Query time: **poly(log n / ε)**