In-Class Quiz 5

Question 1: Let x and y be two sets with Jaccard similarity $J(x, y) = 1/2$. Assume that you hash these sets using MinHash, 3 times independently into 3 different hash tables. What is the probability that they are in the same bucket in at least one of these tables? Show your work.

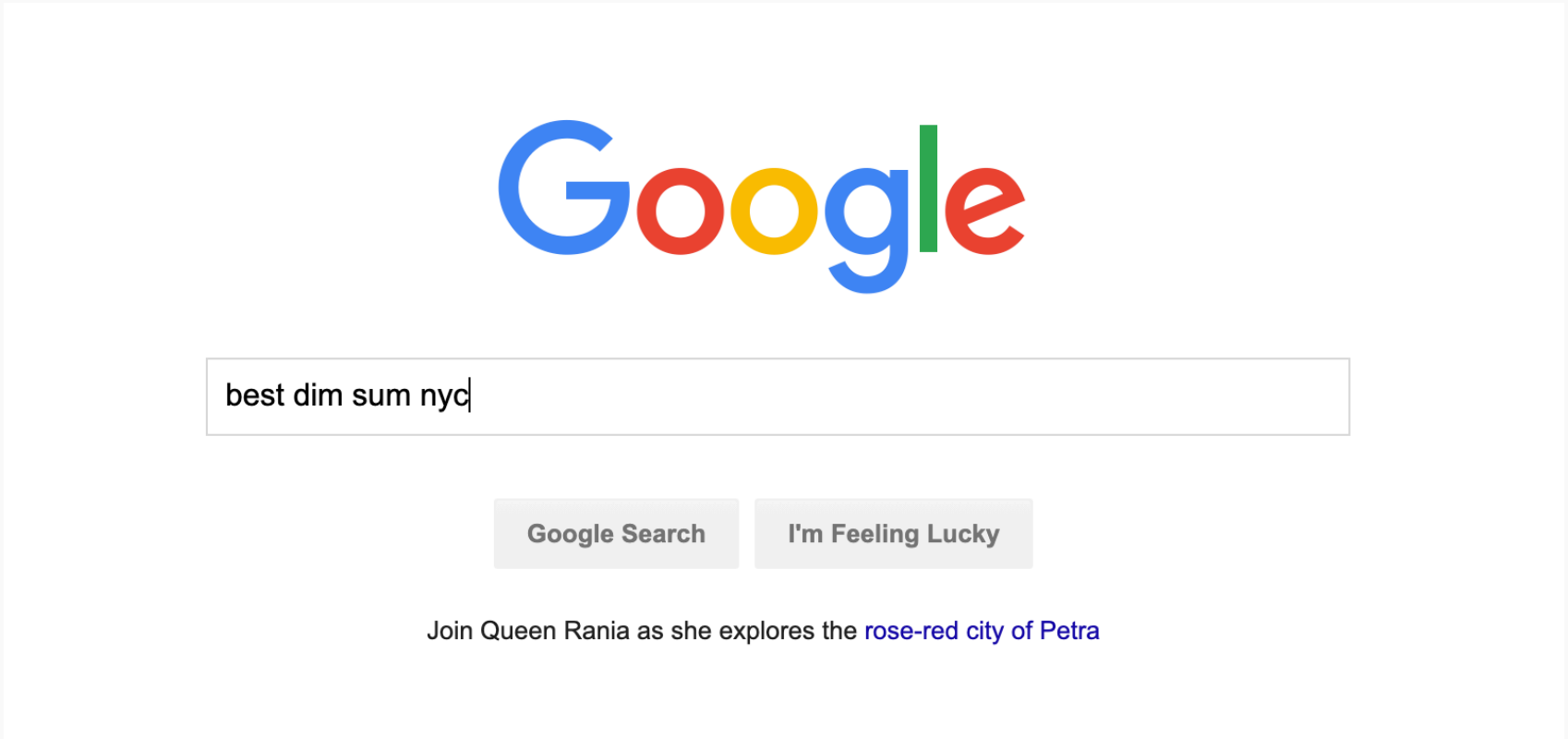
Question 2: True or False: a pairwise independent hash function can also be locality sensitive. Give a short explanation of your answer.

Graph-Based Methods for Vector Search

Christopher Musco
New York University

WHAT IS VECTOR SEARCH?

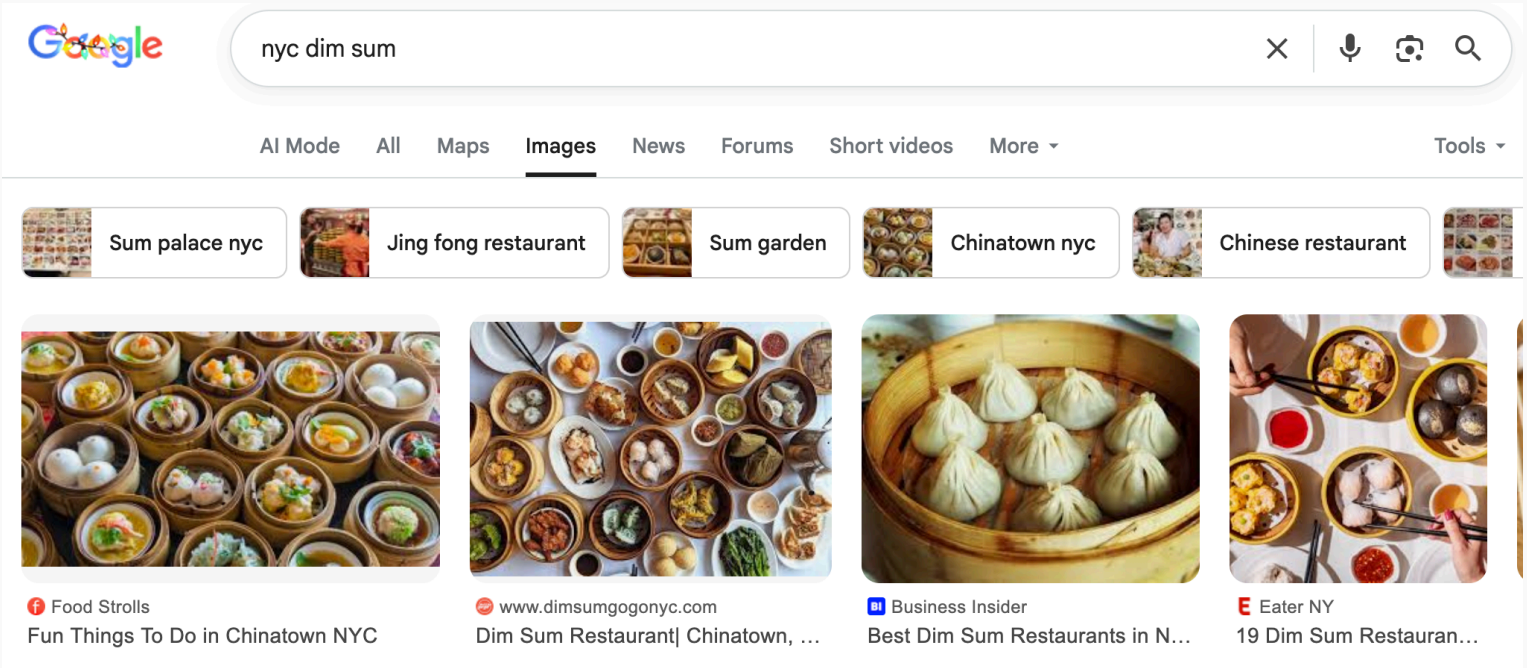
CLASSICAL SEARCH



For decades search has been based on keywords.

Return documents (webpages, emails, files on your computer) that contain many of the words in your query.

CLASSICAL SEARCH



Until fairly recently, this was even true for image and multimedia search!

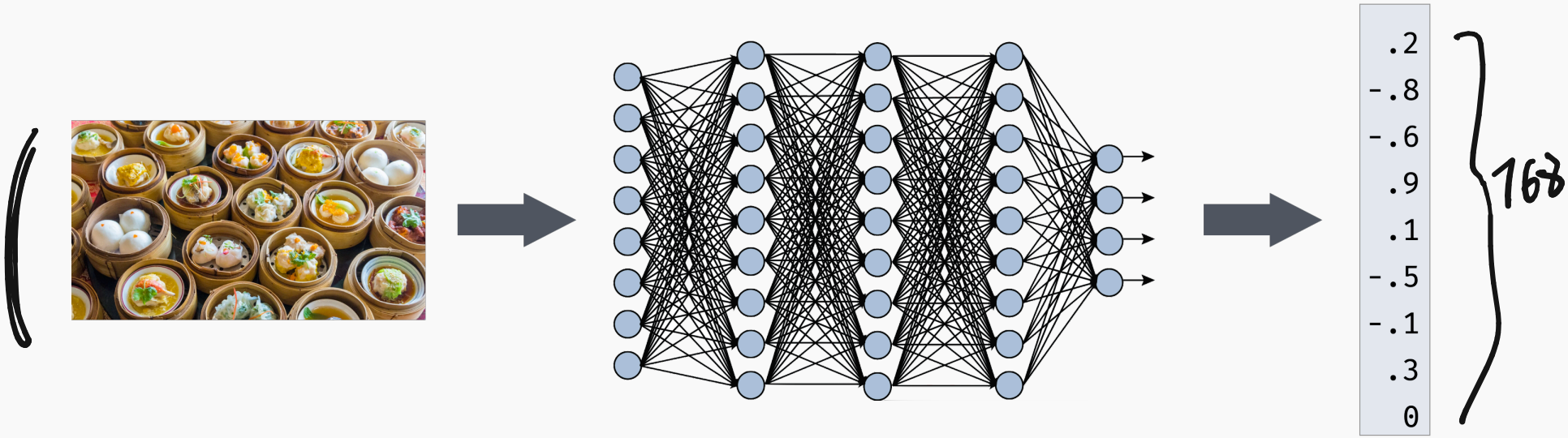
CLASSICAL SEARCH



```
1  <div class="image-block-wrapper" data-animation-role="image">
2      <div class="sqs-image-shape-container-element">
3          <img alt="table full of colorful authentic Dim Sum
           baskets in NYC - best things to do Chinatown NYC"
           src="https://images.squarespace-cdn.com/content/v1/
           5e18d3/1624117/best+things+to+do+Chinatown+NYC+-
           +Dim+Sum" width="2500" height="1406">
4      </div>
5  </div>
```

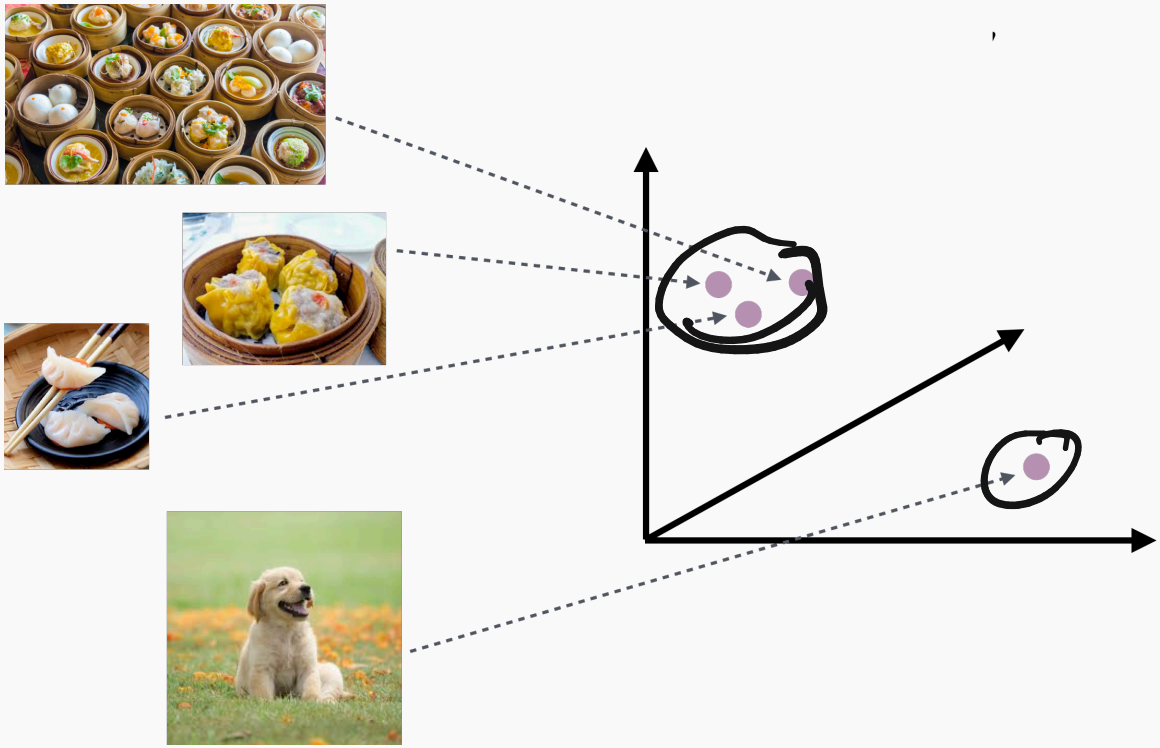
NEW SEARCH PARADIGM: VECTOR SEARCH

Step 1: Deep neural network (BERT, CLIP, etc.) is used to convert documents, images, videos, and more into high-dimensional, numerical vectors.



NEW SEARCH PARADIGM: VECTOR SEARCH

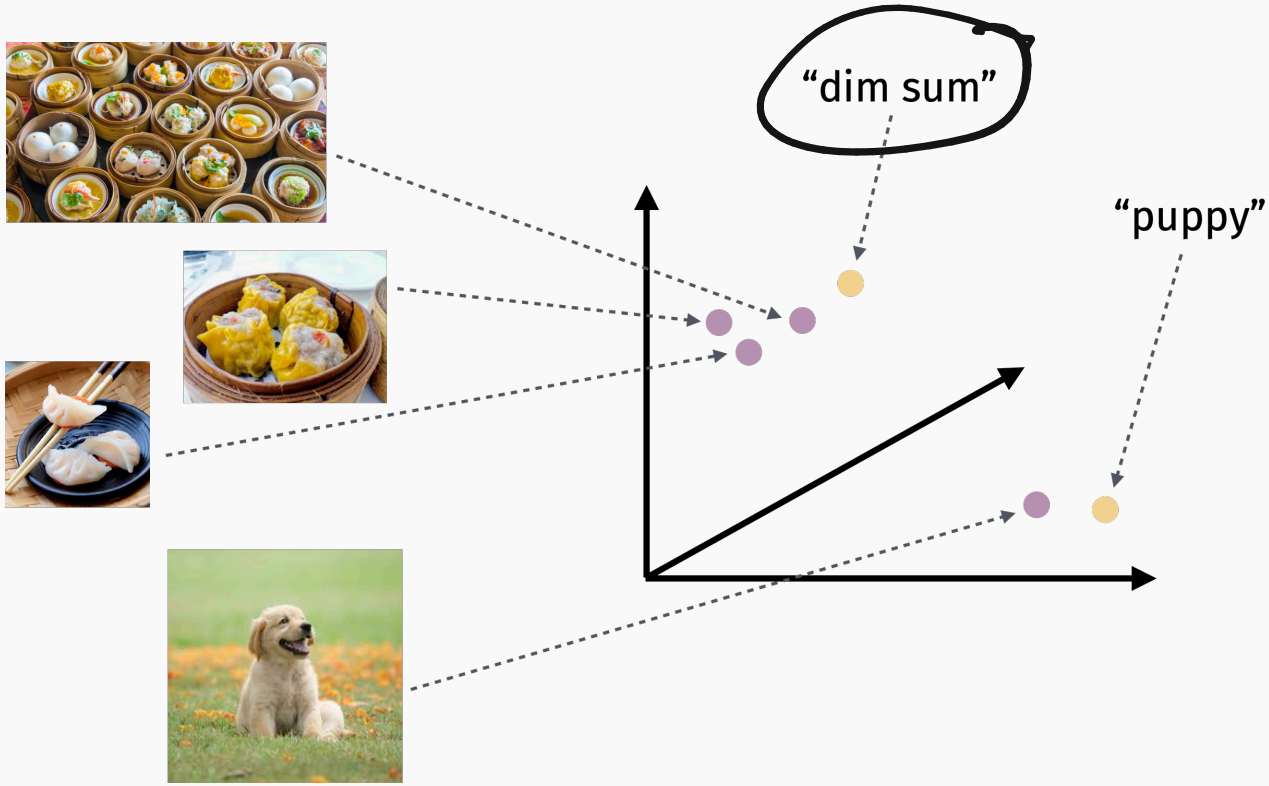
Step 1: Deep neural network (BERT, CLIP, etc.) is used to convert documents, images, videos, and more into high-dimensional, numerical vectors.



The network is trained so that semantically similar objects map to similar vectors.

NEW SEARCH PARADIGM: VECTOR SEARCH

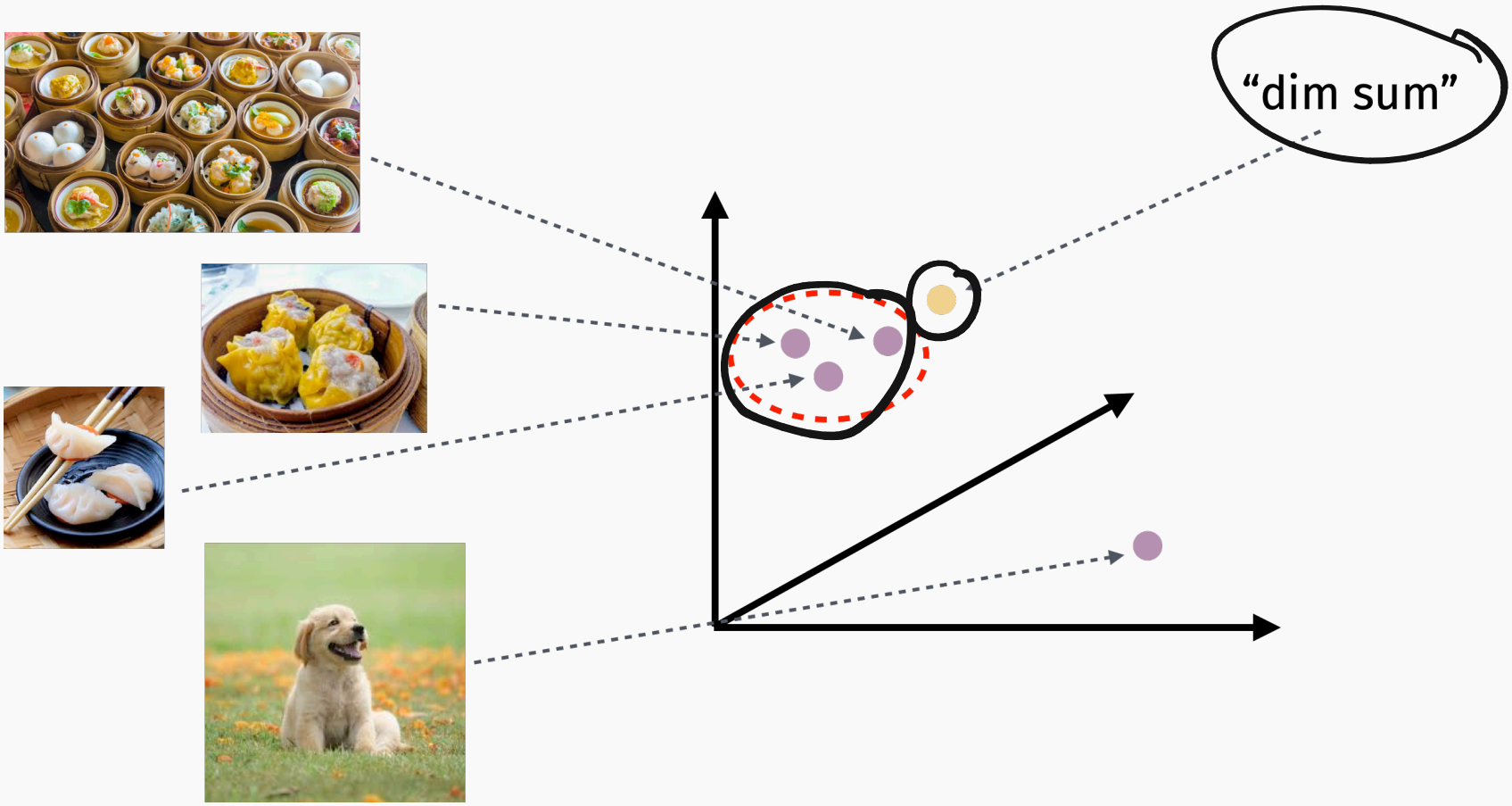
Step 1: Deep neural network (BERT, CLIP, etc.) is used to convert documents, images, videos, and more into high-dimensional, numerical vectors.



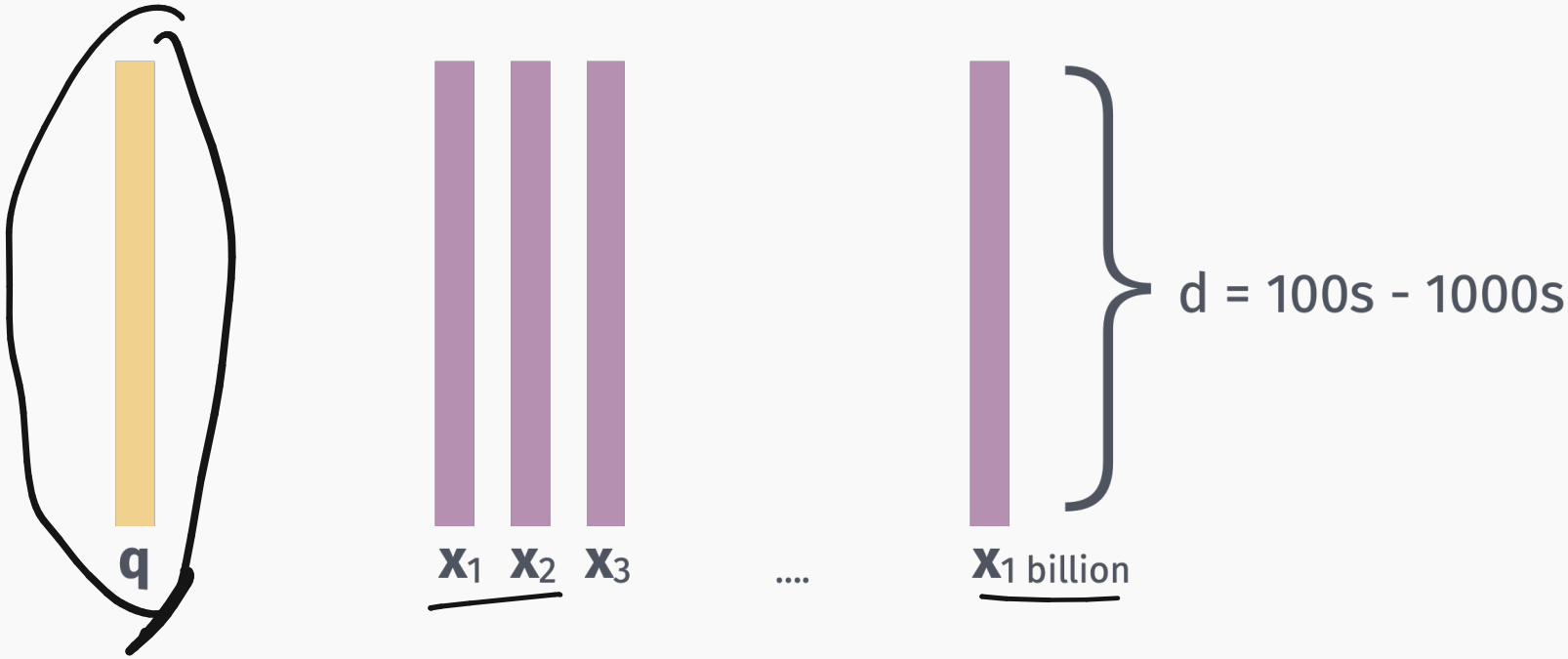
The network is trained so that semantically similar objects map to similar vectors.

NEW SEARCH PARADIGM: VECTOR SEARCH

Step 2: Find items matching a query by embedding the query into a vector, and finding nearby vectors.



NEAREST VECTOR SEARCH



Algorithmic problem: find the closest k vectors in a vector database $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ to a given query vector $\mathbf{q}$. For $k = 1$,

$$\text{return} \quad \underset{i \in \{1,\ldots,n\}}{\arg\min} \|\mathbf{x}_i - \mathbf{q}\|.$$

NEW SEARCH PARADIGM: VECTOR SEARCH

The fact that neural networks simplify multimedia search to a purely geometric problems is amazing!

$$\operatorname*{arg\,min}_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|.$$

NEW SEARCH PARADIGM: VECTOR SEARCH

The fact that neural networks simplify multimedia search to a purely geometric problems is amazing!

$$\arg\min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|.$$

But this is a really hard algorithmic problem...

For a database of n vectors in d dimensions, a naive linear scan takes $O(nd)$ time.

VECTOR SEARCH IN PRACTICE

Given its importance, a multi-billion dollar industry has developed around this problem in the last several years. Almost entirely focused on solving this problem faster.



Big focus on applications to Retrieval-Augmented Generation (RAG) systems for providing context (user emails, code snippets, documents, etc.) to language models.

VECTOR SEARCH ALGORITHMS

NAIVE APPROACH

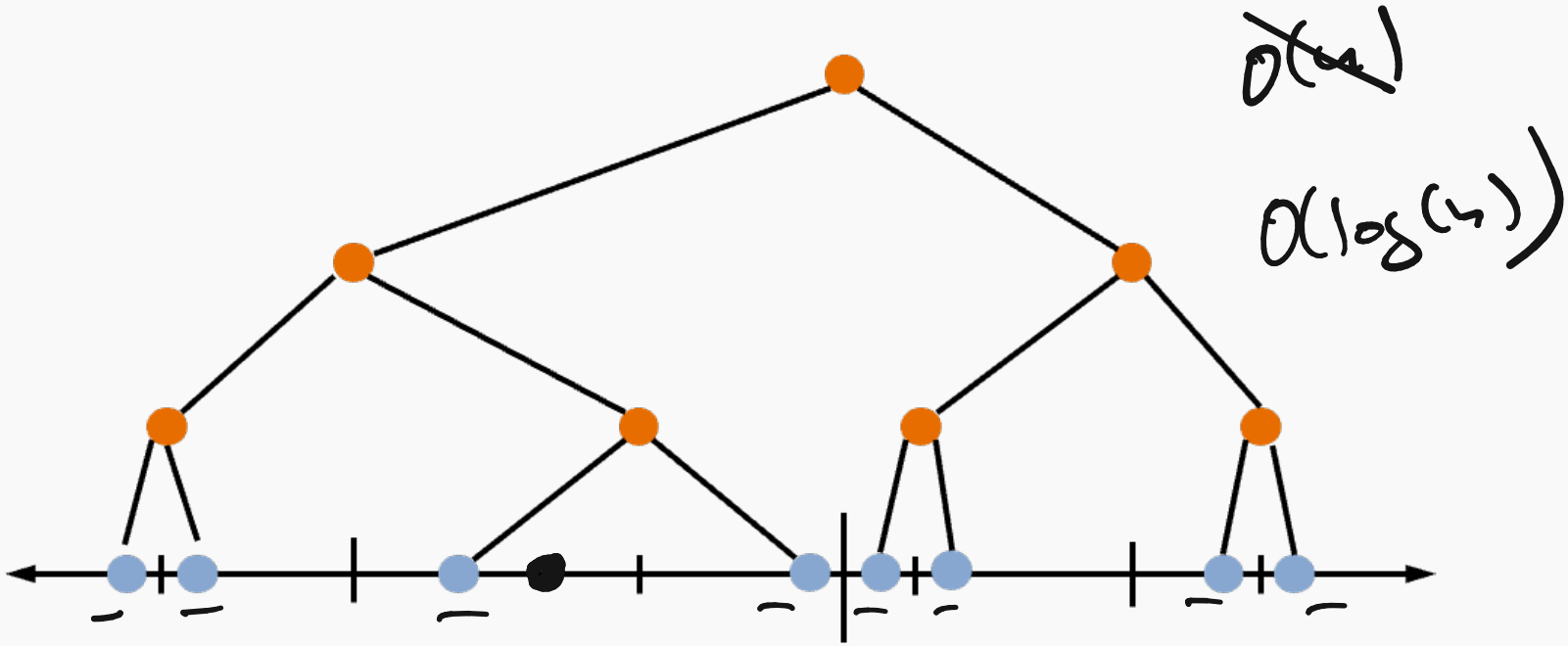
$O(nd)$

Goal: For a given query $\mathbf{q} \in \mathbb{R}^d$, find $\arg\min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|$.

Our only hope to do better than the naive $O(nd)$ time is if we allow for some **pre-processing.**

ONE DIMENSION

For 1-dimensional vectors (i.e., **q** is a number), we can solve the search problem by building a binary search tree.

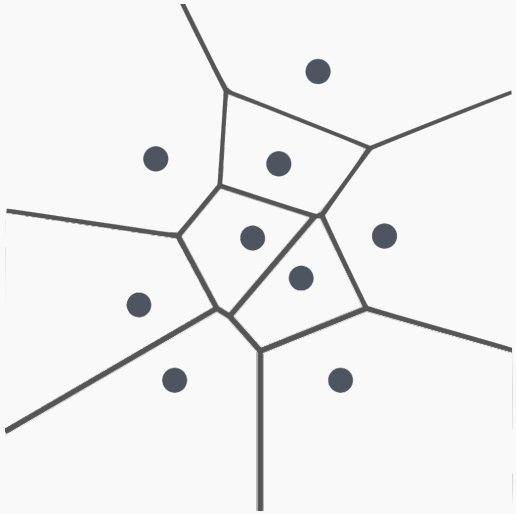


Preprocessing time: $O(n \log n)$.

Space complexity: $O(n)$.

Query time: $O(\log n)$.

Can also be solved with $O(n)$ space and $O(\log n)$ time in $d = 2$ dimensions.



In $d \geq 3$ dimensions, space or query complexity depends exponentially on d, even if we allow approximation.

$$\text{Find } \mathbf{x}_j \text{ such that } \|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|.$$

approx. factor $c = 2$

The “curse-of-dimensionality” was broken with the introduction of **Locality Sensitive Hashing (LSH).**

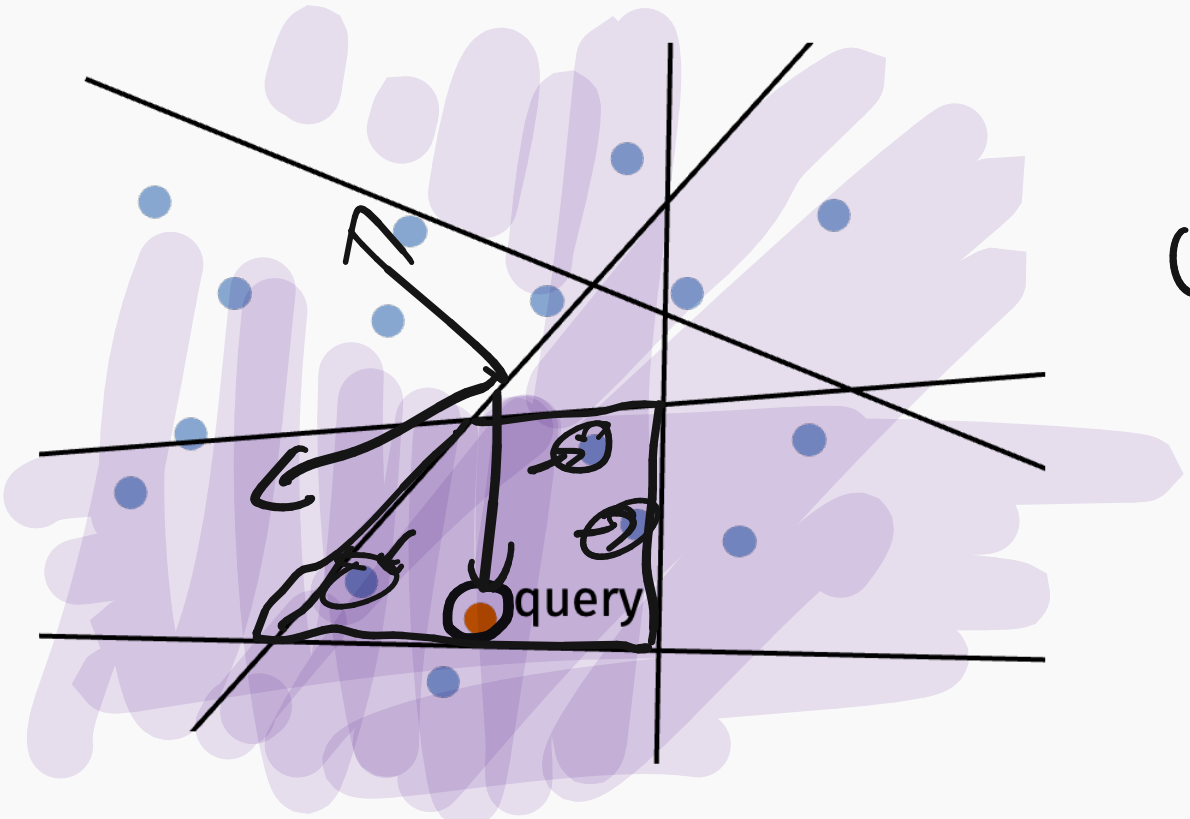



[Indyk, Motwani, 1998].

Rough idea behind Locality Sensitive Hashing (LSH):

1. Pick a bunch of random hyperplanes.
2. Check which side of each hyperplane **q** lies on.
   - Takes $O(d)$ time per hyperplane.
3. Return closest point that lies in the same region as **q**.

h :# of hyperplanes



O(hd)

O(d) per hyperplane

Possible to prove some pretty amazing theoretical guarantees!

Theorem (Andoni, Indyk, FOCS 2006)

For any $c \geq 1$, in $\tilde{O}\left(dn^{1/c^2}\right)$ time we can find $\mathbf{x}_j$ satisfying:

$$\|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|.$$

Space complexity of data structure is $\tilde{O}\left(nd + n^{1+1/c^2}\right)$

Possible to prove some pretty amazing theoretical guarantees!

Theorem (Andoni, Indyk, FOCS 2006)

For any $c \geq 1$, in $\tilde{O}\left(dn^{1/c^2}\right)$ time we can find $\mathbf{x}_j$ satisfying:

$$\|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|.$$

Space complexity of data structure is $\tilde{O}\left(nd + n^{1+1/c^2}\right)$

As an example, if $c = 2$, query time scales with $n^{1/4}$, which is pretty amazing! $(1 \text{ billion})^{1/4} < 200$.

Possible to prove some pretty amazing theoretical guarantees!

Theorem (Andoni, Indyk, FOCS 2006)

For any $c \geq 1$, in $\tilde{O}\left(dn^{1/c^2}\right)$ time we can find $\mathbf{x}_j$ satisfying:

$$\|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|.$$

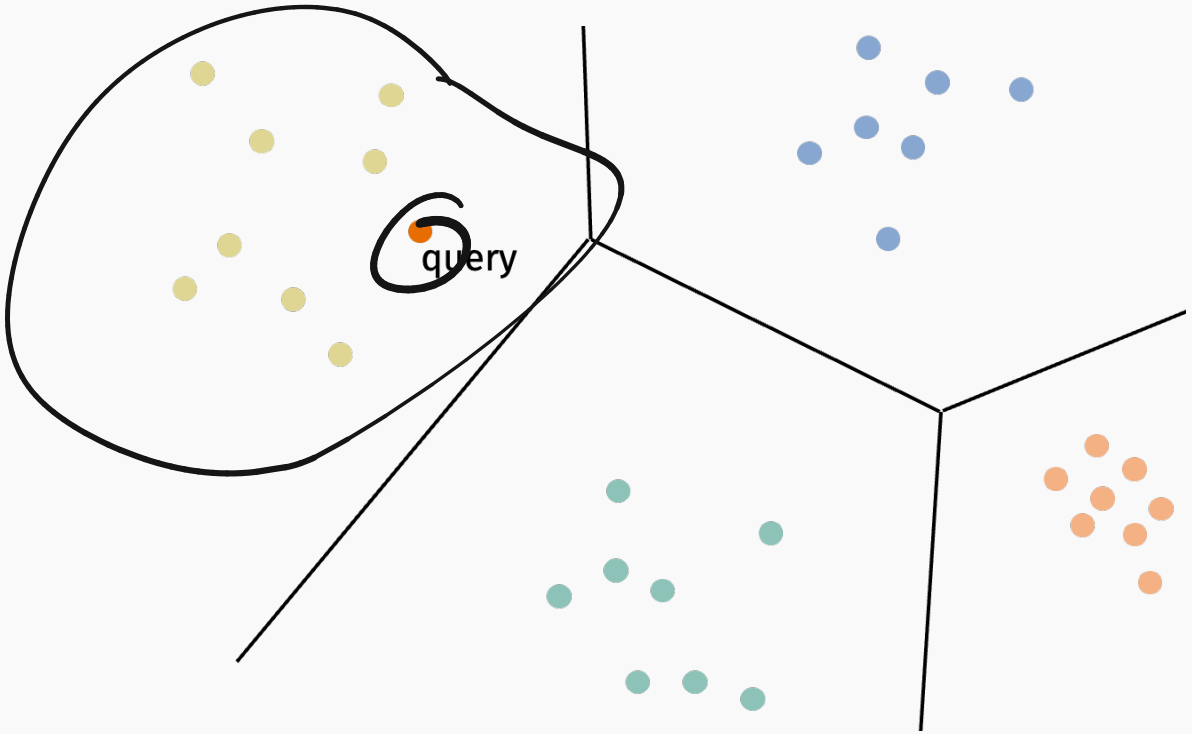
Space complexity of data structure is $\tilde{O}\left(nd + n^{1+1/c^2}\right)$

As an example, if $c = 2$, query time scales with $n^{1/4}$, which is pretty amazing! $(1 \text{ billion})^{1/4} < 200$.

Also works well in practice. **LSH and related developments won Indyk, Charikar, and Broder the 2012 ACM Paris Kanellakis Theory and Practice Award.**

SPACE PARTITIONING METHODS

Practical systems improve accuracy by choosing hyperplanes in a data-dependent way, e.g. via k-means clustering.



Key component of state-of-the-art vector search libraries like **Meta's FAISS** and **Google's SCANN**.

VECTOR SEARCH IN PRACTICE

But few new vector databases are using Locality Sensitive Hashing or data-dependent space partitioning.



New kid on the block: Graph-based Search.

Nearest-neighbor search in practice

New kid on the block: Graph-based Search.

- **Navigating Spreading-out Graphs (NSG)** [Fu, Xiang, Wang, Cai, 2017]
- **Hierarchical Navigable Small World (HNSW)** [Malkov, Yashunin, 2018]
- **Microsoft's DiskANN** [Subramanya, Devvrit, Kadekodi, Krishaswamy, Simhadri 2019]

Very similar methods studied for low-dimensions in the 1990s by Arya, Mount, Clarkson, Kleinberg, and others.

NEAREST-NEIGHBOR SEARCH IN PRACTICE

New kid on the block: Graph-based Search.

- **Navigating Spreading-out Graphs (NSG)** [Fu, Xiang, Wang, Cai, 2017]
- **Hierarchical Navigable Small World (HNSW)** [Malkov, Yashunin, 2018]
- **Microsoft's DiskANN** [Subramanya, Devvrit, Kadekodi, Krishaswamy, Simhadri 2019]

Very similar methods studied for low-dimensions in the 1990s by Arya, Mount, Clarkson, Kleinberg, and others.

These methods are inspired by the "small world" experiments of Stanley Milgram from the 1960s

STANLEY MILGRAM

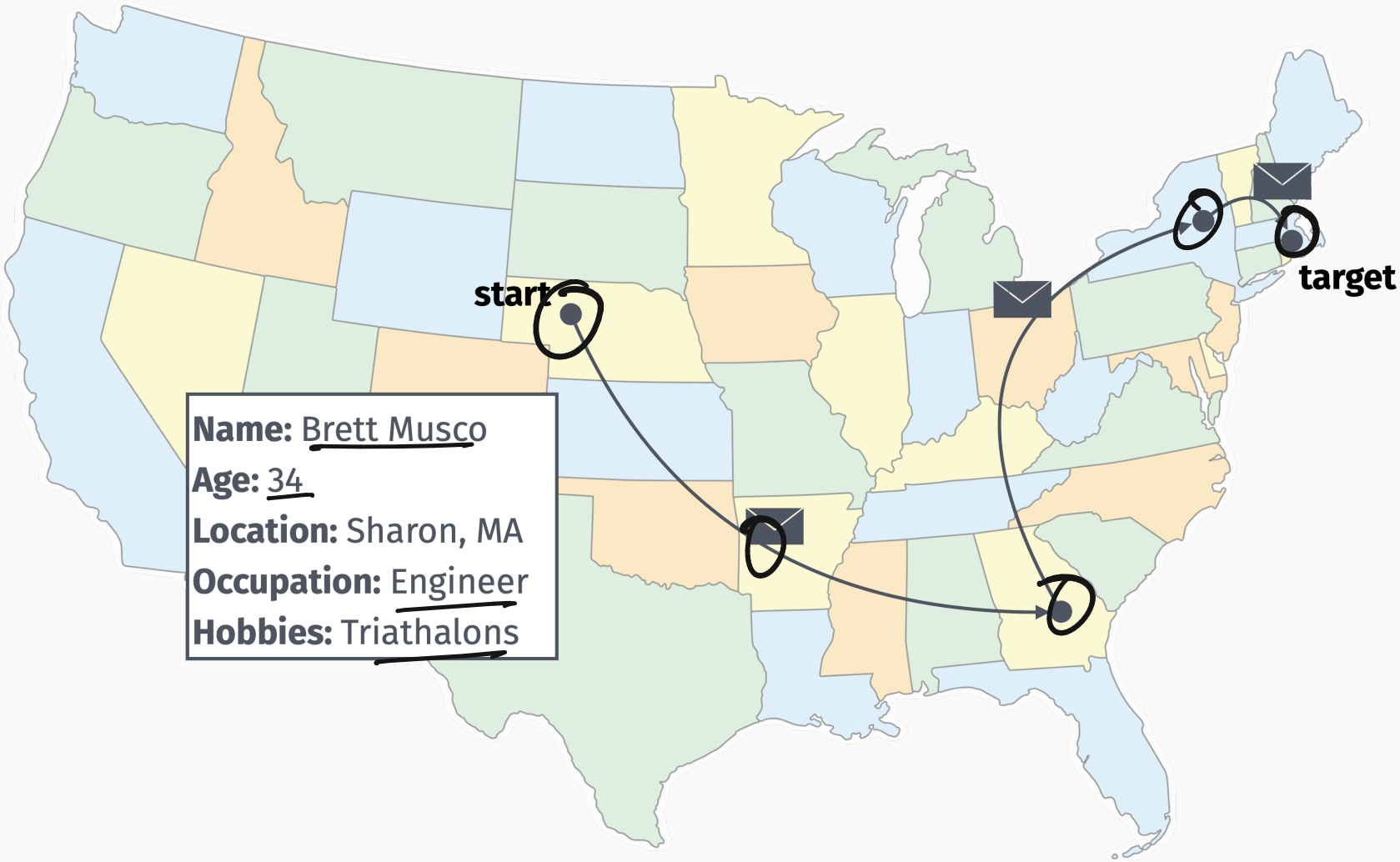


- Psychology professor at CUNY Graduate Center.
- Most famous for his experiments on obedience at Yale.
- Second claim to fame are his **small-world experiments.**

SMALL-WORLD EXPERIMENT

1. Give a random person a description of someone who lives in a completely different part of the country.
2. Ask them to either send that description directly to the person (if they know them) or forward it to an acquaintance who is most likely to know the person.

SMALL-WORLD EXPERIMENT



SMALL-WORLD EXPERIMENT

Not all descriptions reached their destination, but of those that did, the average path length was < 6!

Gave rise to the phrase **"six degrees of separation"**, which is also called the "small-world phenomenon".

SMALL-WORLD EXPERIMENT

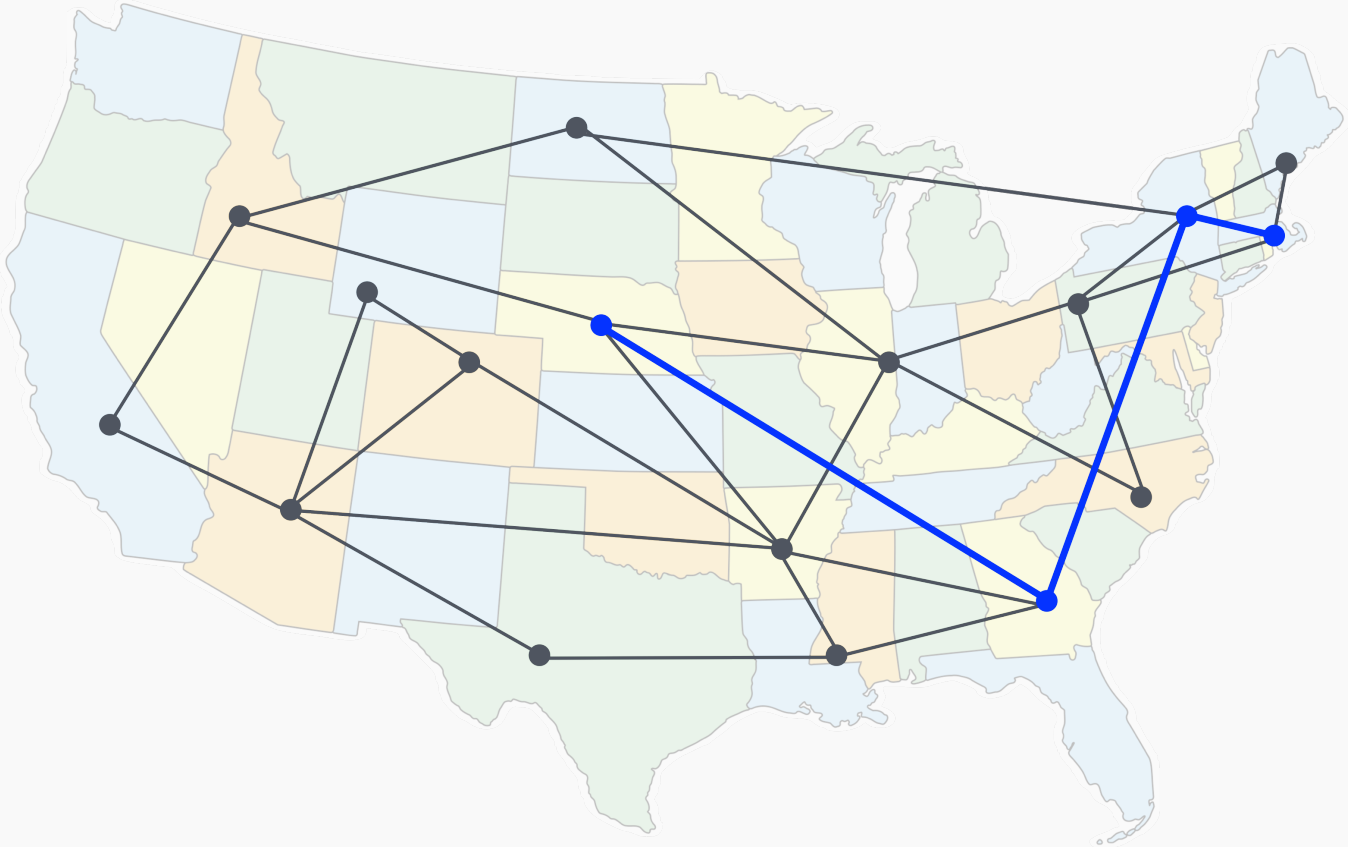
Not all descriptions reached their destination, but of those that did, the average path length was < 6!

Gave rise to the phrase **“six degrees of separation”**, which is also called the “small-world phenomenon”.

But to computer scientists, the key takeaway was a bit different.

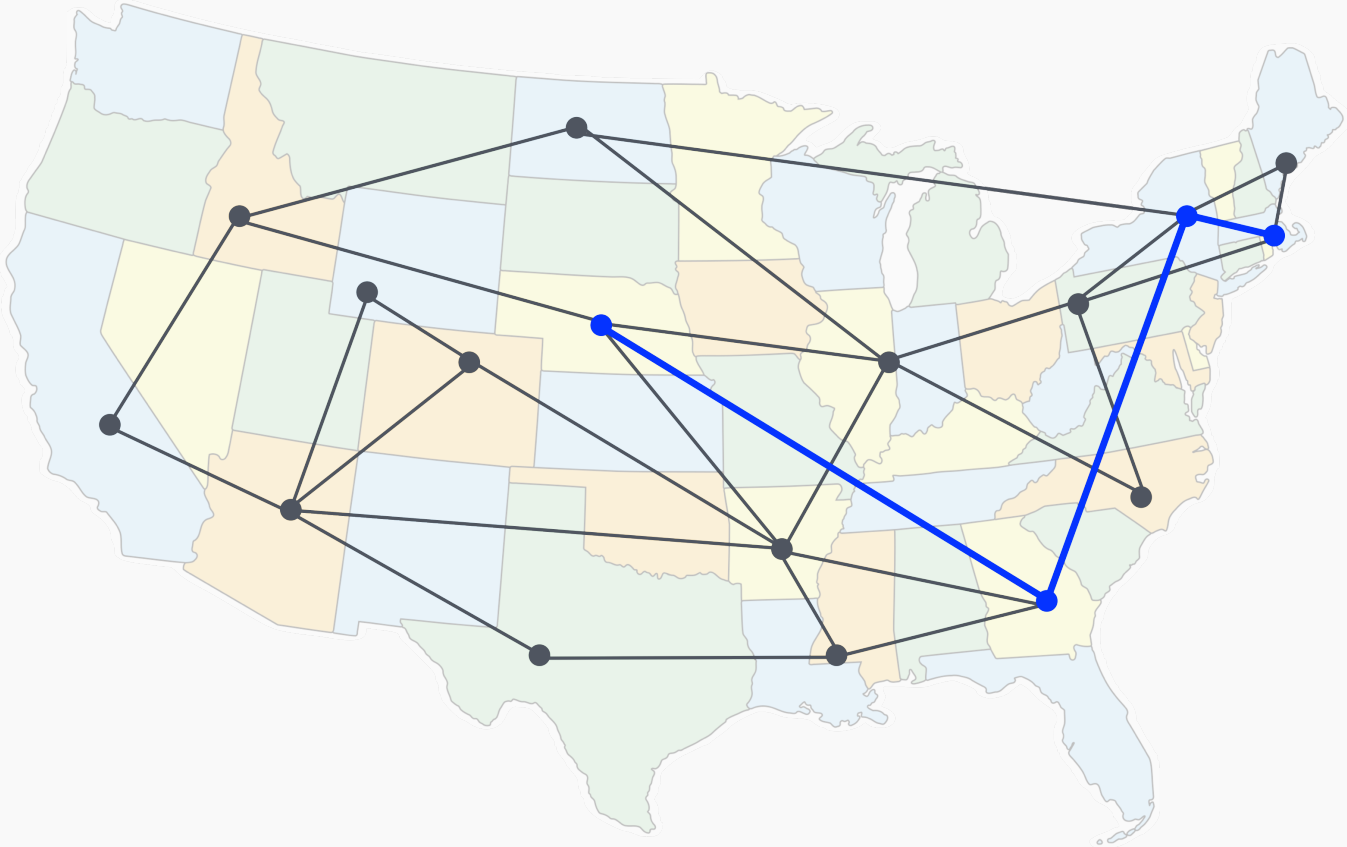
SMALL-WORLD EXPERIMENT

Not only do short-paths exist, but we did not need a shortest-path algorithm to find them! Greedy routing worked.



SMALL-WORLD EXPERIMENT

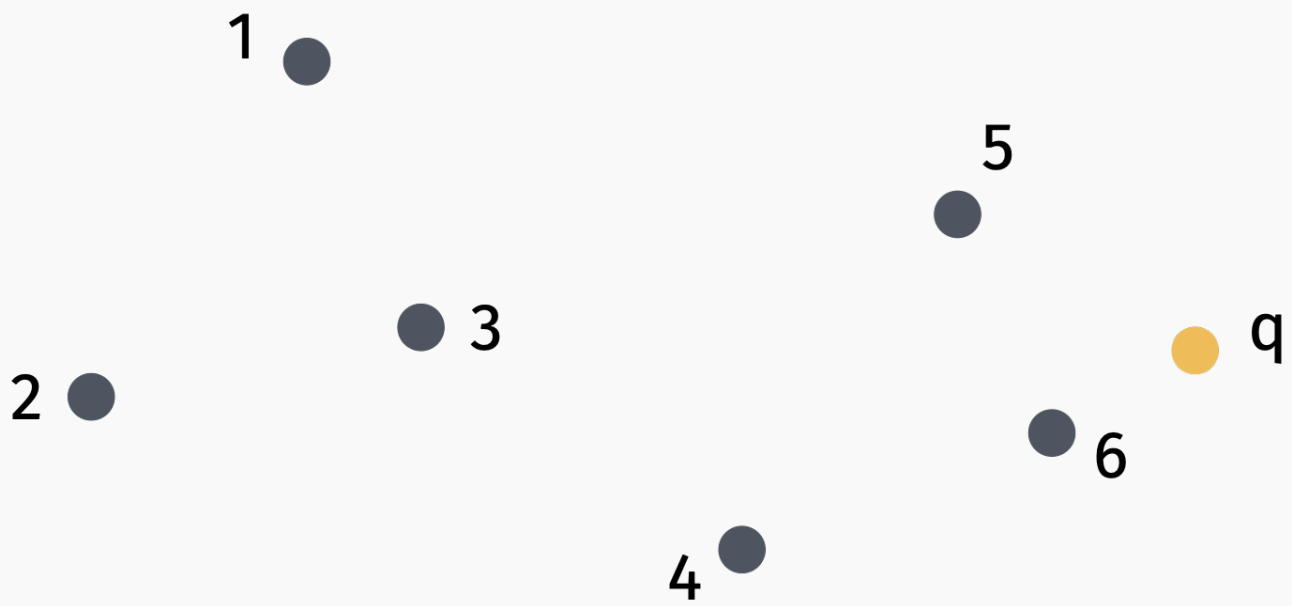
Not only do short-paths exist, but we did not need a shortest-path algorithm to find them! Greedy routing worked.



Question: Can we construct graphs with this property? I.e., for which short paths can be found with greedy search?

BASIC IDEA BEHIND GRAPH-BASED SEARCH

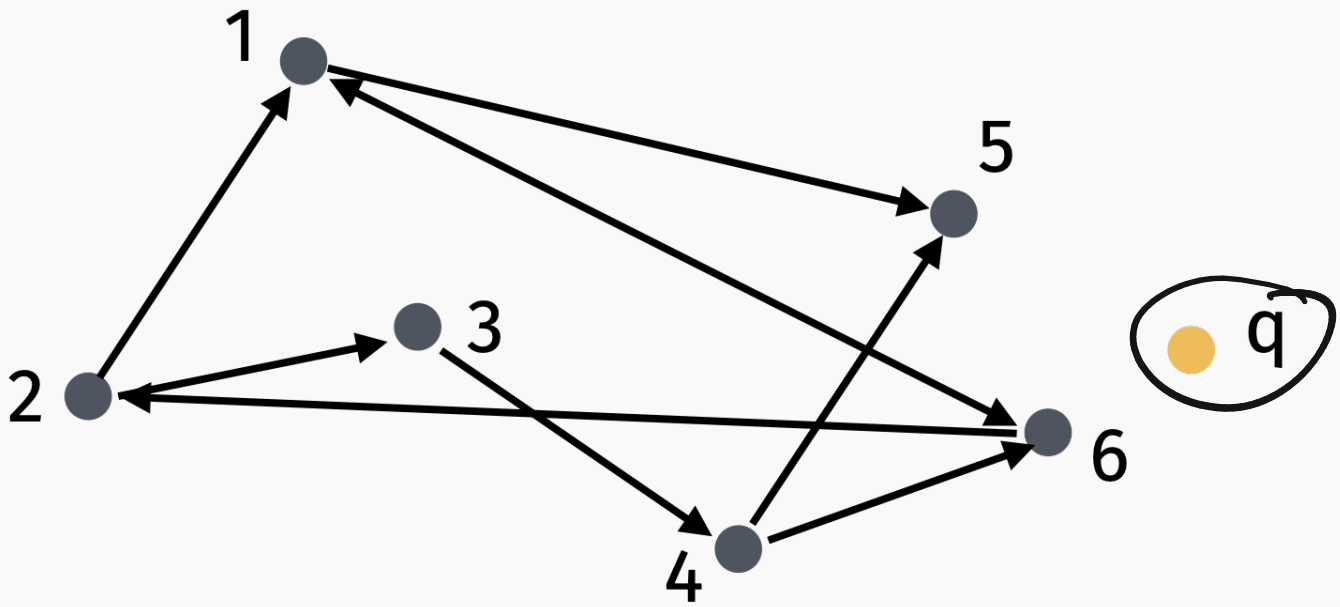
1. Construct a directed search graph over our dataset.



2. Run greedy search in the graph.

BASIC IDEA BEHIND GRAPH-BASED SEARCH

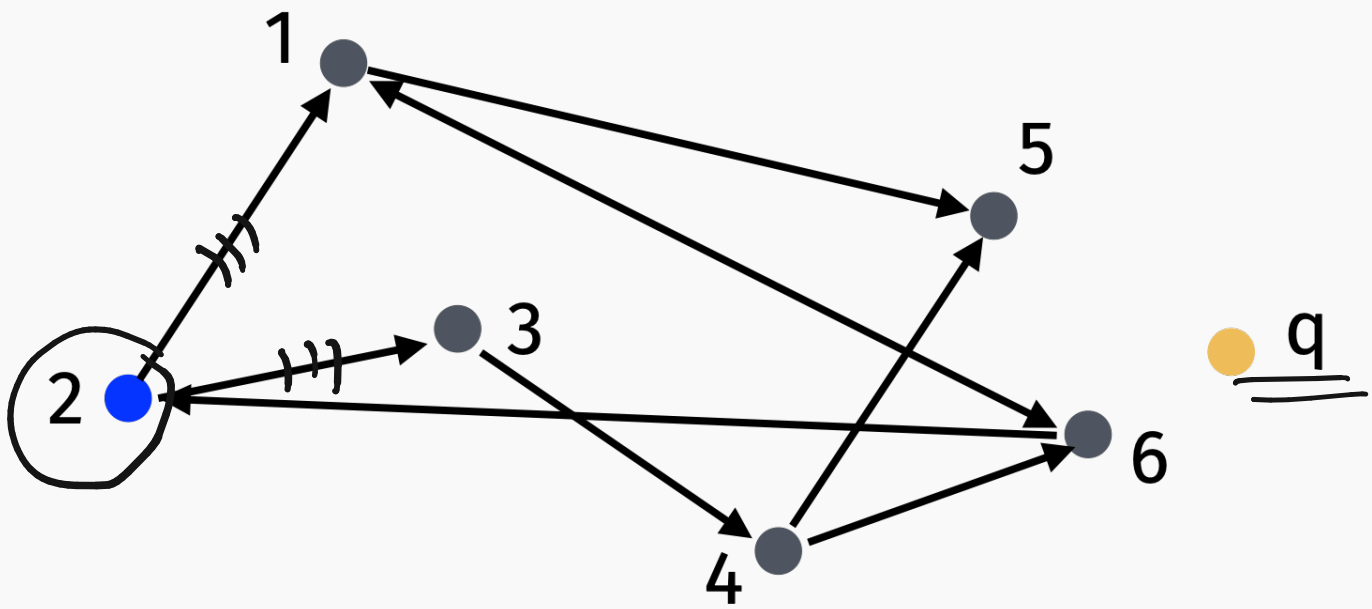
1. Construct a directed search graph over our dataset.



2. Run greedy search in the graph.

BASIC IDEA BEHIND GRAPH-BASED SEARCH

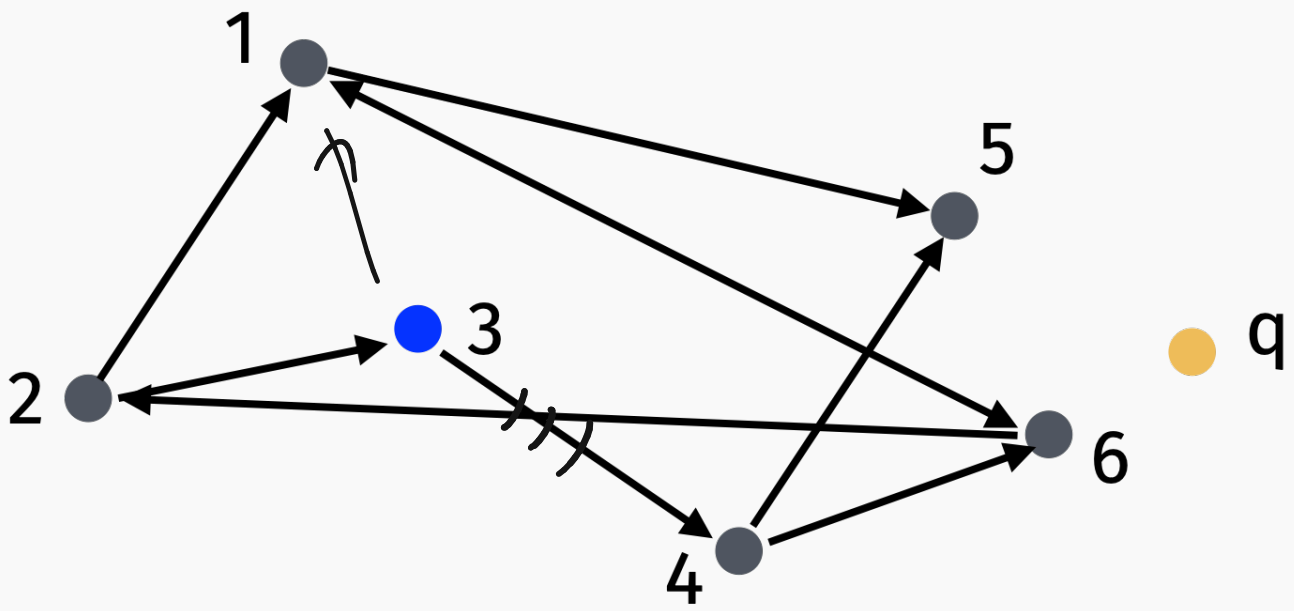
1. Construct a directed search graph over our dataset.



2. Run greedy search in the graph.

BASIC IDEA BEHIND GRAPH-BASED SEARCH

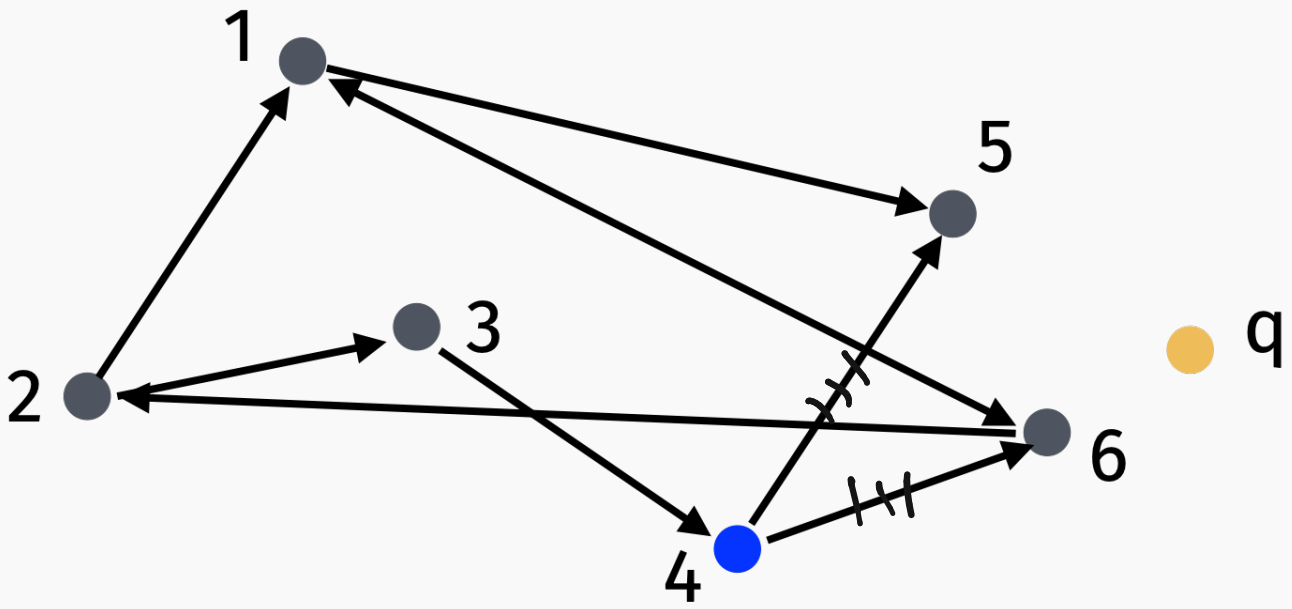
1. Construct a directed search graph over our dataset.



2. Run greedy search in the graph.

BASIC IDEA BEHIND GRAPH-BASED SEARCH

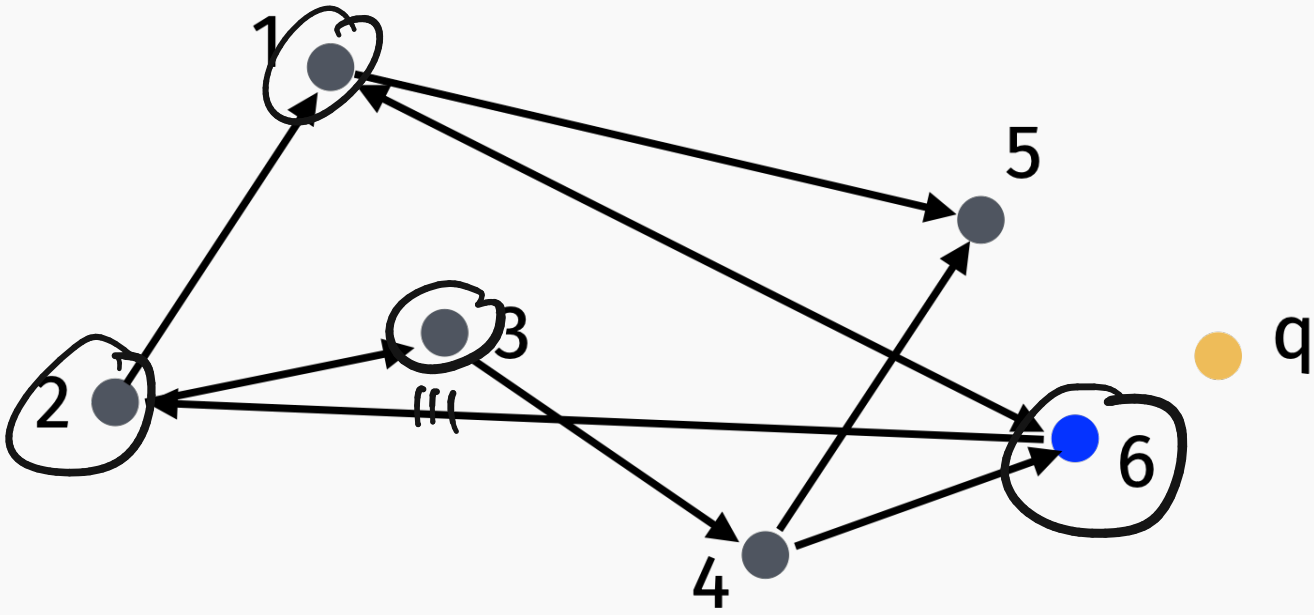
1. Construct a directed search graph over our dataset.



2. Run greedy search in the graph.

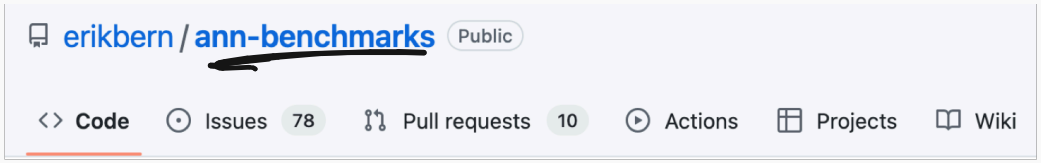
BASIC IDEA BEHIND GRAPH-BASED SEARCH

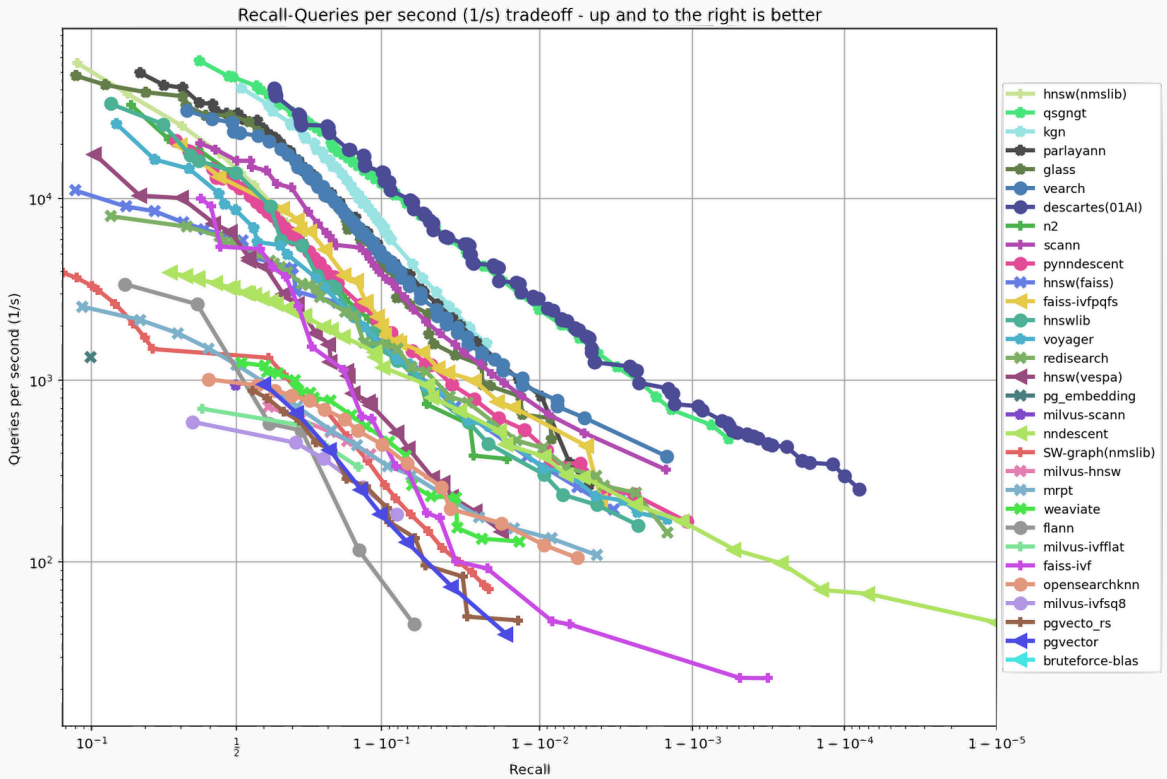
1. Construct a directed search graph over our dataset.



2. Run greedy search in the graph.

Graph-based methods like HNSW and DiskANNare topping benchmarks and competitions!





GRAPH-BASED SEARCH IN PRACTICE

Graph-based methods like HNSW and DiskANNare topping benchmarks and competitions!

Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search

Harsha Vardhan Simhadri[1] HARSHASI@MICROSOFT.COM
George Williams[2] GWILLIAMS@IEEE.ORG
Martin Aumüller[3] MAAU@ITU.DK
Matthijs Douze[4] MATTHIJS@FB.COM
Artem Babenko[5] ARTEM.BABENKO@PHYSTECH.EDU
Dmitry Baranchuk[5] DBARANCHUK@YANDEX-TEAM.RU
Qi Chen[1] CHEQI@MICROSOFT.COM
Lucas Hosseini[4] LUCAS.HOSSEINI@GMAIL.COM
Ravishankar Krishnaswamy[1] RAKRI@MICROSOFT.COM
Gopal Srinivasa[1] GOPALSR@MICROSOFT.COM
Suhas Jayaram Subramanya[6] SUHASJ@CS.CMU.EDU
Jingdong Wang[7] WANGJINGDONG@BAIDU.COM

[1] Microsoft Research [2] GSI Technology [3] IT University of Copenhagen
[4] Meta AI Research [5] Yandex [6] Carnegie Mellon University [7] Baidu

Results of the Big ANN: NeurIPS'23 competition

Harsha Vardhan Simhadri, Microsoft, harshasi@microsoft.com
Martin Aumüller, IT University of Copenhagen, maau@itu.dk
Amir Ingber, Pinecone, ingber@pinecone.io
Matthijs Douze, Meta AI Research, matthijs@meta.com
George Williams
Magdalen Dobson Manohar, Carnegie Mellon University
Dmitry Baranchuk, Yandex
Edo Liberty, Pinecone
Frank Liu, Zilliz
Ben Landrum, University of Maryland
Mazin Karjikar, University of Maryland
Laxman Dhulipala, University of Maryland
Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weiguo Zheng, Fudan University
Zihao Wang, Shanghai Jiao Tong University
Jie Yin, Baidu
Ben Huang, Baidu

GRAPH-BASED SEARCH IN PRACTICE

Graph-based methods like HNSW and DiskANNare topping benchmarks and competitions!

Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search

Harsha Vardhan Simhadri[1] HARSHASI@MICROSOFT.COM
George Williams[2] GWILLIAMS@IEEE.ORG
Martin Aumüller[3] MAAU@ITU.DK
Matthijs Douze[4] MATTHIJS@FB.COM
Artem Babenko[5] ARTEM.BABENKO@PHYSTECH.EDU
Dmitry Baranchuk[5] DBARANCHUK@YANDEX-TEAM.RU
Qi Chen[1] CHEQI@MICROSOFT.COM
Lucas Hosseini[4] LUCAS.HOSSEINI@GMAIL.COM
Ravishankar Krishnaswamy[1] RAKRI@MICROSOFT.COM
Gopal Srinivasa[1] GOPALSR@MICROSOFT.COM
Suhas Jayaram Subramanya[6] SUHASJ@CS.CMU.EDU
Jingdong Wang[7] WANGJINGDONG@BAIDU.COM

[1] Microsoft Research [2] GSI Technology [3] IT University of Copenhagen
[4] Meta AI Research [5] Yandex [6] Carnegie Mellon University [7] Baidu

Results of the Big ANN: NeurIPS'23 competition

Harsha Vardhan Simhadri, Microsoft, harshasi@microsoft.com
Martin Aumüller, IT University of Copenhagen, maau@itu.dk
Amir Ingber, Pinecone, ingber@pinecone.io
Matthijs Douze, Meta AI Research, matthijs@meta.com
George Williams
Magdalen Dobson Manohar, Carnegie Mellon University
Dmitry Baranchuk, Yandex
Edo Liberty, Pinecone
Frank Liu, Zilliz
Ben Landrum, University of Maryland
Mazin Karjikar, University of Maryland
Laxman Dhulipala, University of Maryland
Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weiguo Zheng, Fudan University
Zihao Wang, Shanghai Jiao Tong University
Jie Yin, Baidu
Ben Huang, Baidu

Another huge benefit of graph-based methods is that they work for any distance/similarity function.

Open research challenge: Can we explain the empirical success of graph-based vector search?

PLAN FOR TODAY

There has been a lot of interesting recent work[1], but we are still far from addressing this challenge in a satisfying way.

Goal for remainder of talk: Discuss connection between the performance of graph-based search and the concept of graph navigability.

Discuss some recent theory on the construction of navigable graphs.

[1] [Laarhoven 2018, Prokhorenkova, Shekhovtsov 2020, Indyk, Xu 2023, Gollapudi, Krishnaswamy, Shiragur, Wardhan 2025, Har-Peled, Raichel, Robson 2025, many more]

- **c-approximate nearest neighbor search:** Return j satisfying $\|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|$ for some $c \geq 1$.

NAVIGABLE GRAPHS

c-approximate nearest neighbor search: Return j satisfying $\|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|$ for some $c \geq 1$.

Observation: If query $\mathbf{q} = \mathbf{x}_j$ for some j, we must return j.

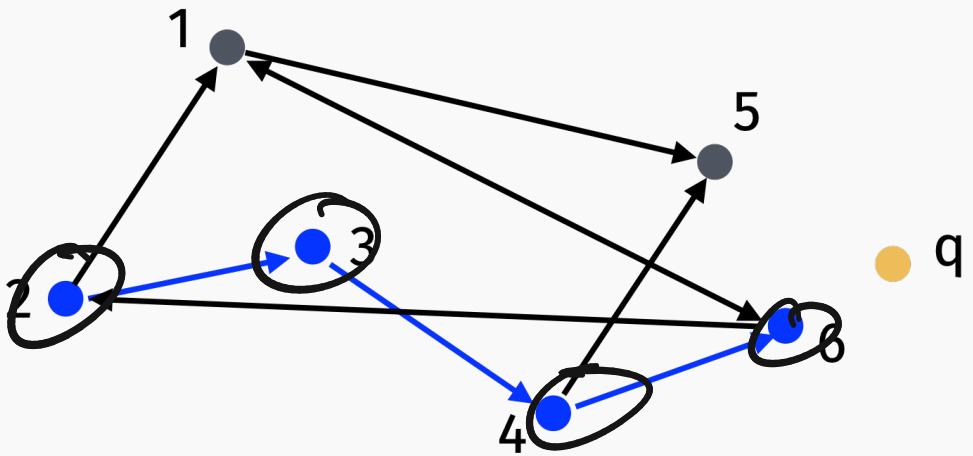
Search graph G should be chosen to at least ensure that we find $\mathbf{q}$ if it is in the dataset.

c-approximate nearest neighbor search: Return j satisfying $\|\mathbf{x}_j - \mathbf{q}\| \leq c \cdot \min_{i \in \{1,\ldots,n\}} \|\mathbf{x}_i - \mathbf{q}\|$ for some $c \geq 1$.

Observation: If query $\mathbf{q} = \mathbf{x}_j$ for some j, we must return j.

Search graph G should be chosen to at least ensure that we find q if it is in the dataset.

Ideally, G should also be sparse and q is found in just a few greedy steps. Runtime of search $\approx$ (out degree) $\times$ (# steps).



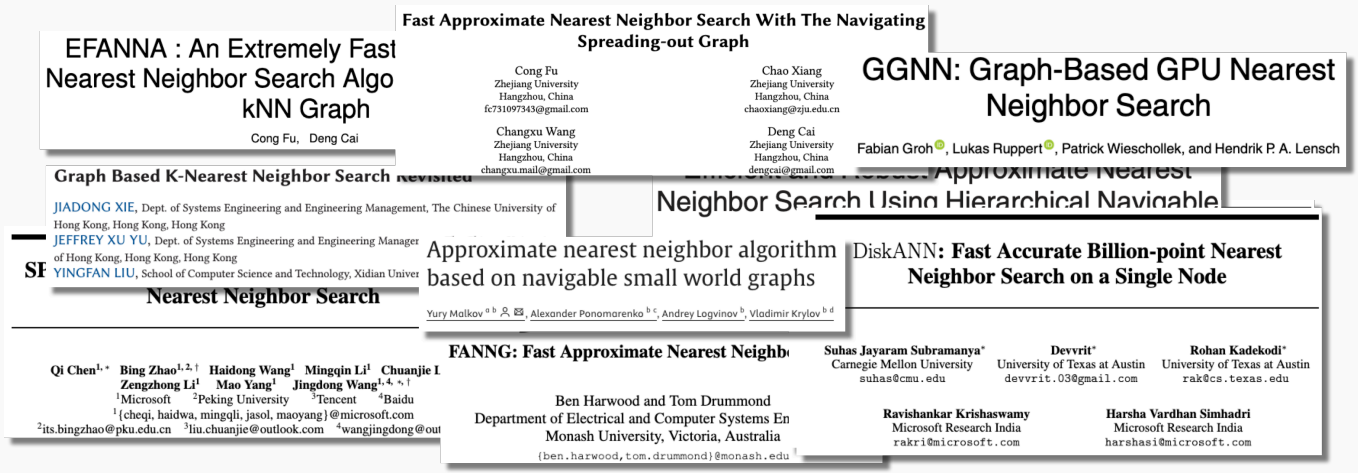
Definition (Navigable Graph)

A directed graph G over a point set $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is navigable if, for all $i, j \in \{1, \ldots, n\}$, greedy search run on G with start node i and query $\mathbf{x}_j$ returns j.

NAVIGABLE GRAPHS

Definition (Navigable Graph)

A directed graph G over a point set $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is navigable if, for all $i, j \in \{1, \ldots, n\}$, greedy search run on G with start node i and query $\mathbf{x}_j$ returns j.



Among dozens of papers on search graph construction, navigability is frequently listed as a desirably property. Lends its name to methods like "hierarchical navigable small world (HNSW) graphs" and "navigating spreading-out graphs (NSG)"

NAVIGABLE GRAPHS

However, few papers actually construct provably navigable graphs! Employ heuristic search graph constructions.

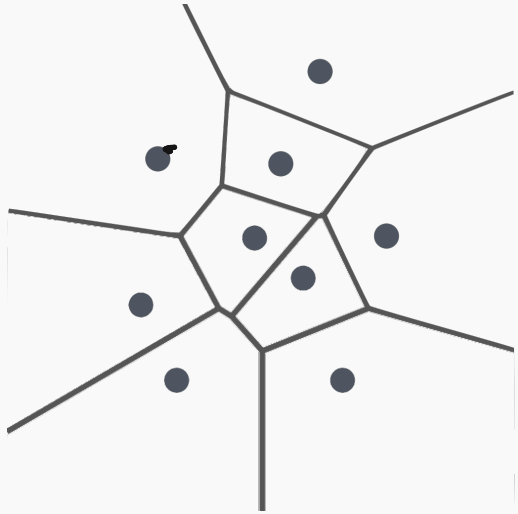
Natural Questions: Do sparse navigable graphs even exist? Can we find them efficiently?

Navigability can be viewed as a minimum necessary condition for graph-based greedy search to succeed, so answering this question is critical to understanding graph-based methods.

SPARSE NAVIGABLE GRAPHS

Known results in low-dimensional Euclidean space:

- **2-dimensions:** The Delaunay graph can be proven to be navigable. This graph has average degree ≤ 6.

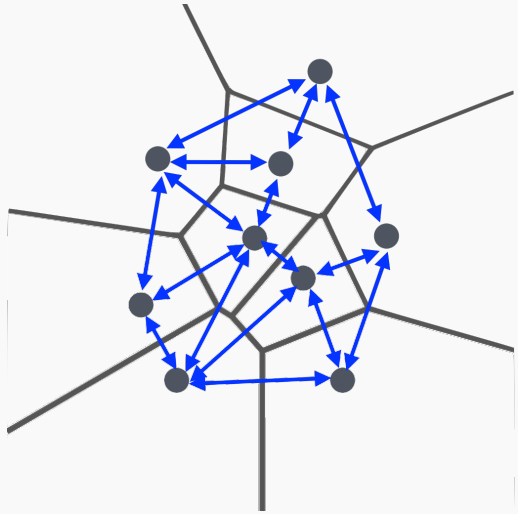


$O(n)$

SPARSE NAVIGABLE GRAPHS

Known results in low-dimensional Euclidean space:

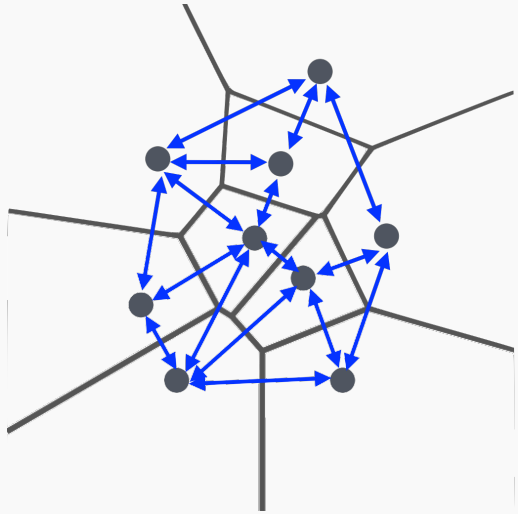
- **2-dimensions:** The Delaunay graph can be proven to be navigable. This graph has average degree ≤ 6.



SPARSE NAVIGABLE GRAPHS

Known results in low-dimensional Euclidean space:

- **2-dimensions:** The Delaunay graph can be proven to be navigable. This graph has average degree ≤ 6.



- **d-dimensions:** The Sparse Neighborhod Graph of Arya and Mount [SODA, 1993] is navigable and has average degree $O(2^d)$.

Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits. Diwan, Gou, Musco, Musco, Suel, NeurIPS 2024.

Claim

Any dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *admits a navigable graph with average degree* $\leq 2\sqrt{n}$*, under any distance function.*

Today we will prove a slightly weaker $O(\sqrt{n \log n})$ bound.

$O(n^{3/2})$

Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits. Diwan, Gou, Musco, Musco, Suel, NeurIPS 2024.

Claim

Any dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *admits a navigable graph with average degree* $\leq 2\sqrt{n}$*, under any distance function.*

Today we will prove a slightly weaker $O(\sqrt{n \log n})$ bound.

Lower bound: This bound cannot be improved past $O(\sqrt{n})$ even for points in $O(\log n)$-dimensions under the Euclidean distance.

ALTERNATIVE DEFINITION

Definition (Navigable Graph)

A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$,

$$d(j, k) < d(j, i).$$

for some k in i's out neighborhood, $\mathcal{N}(i)$

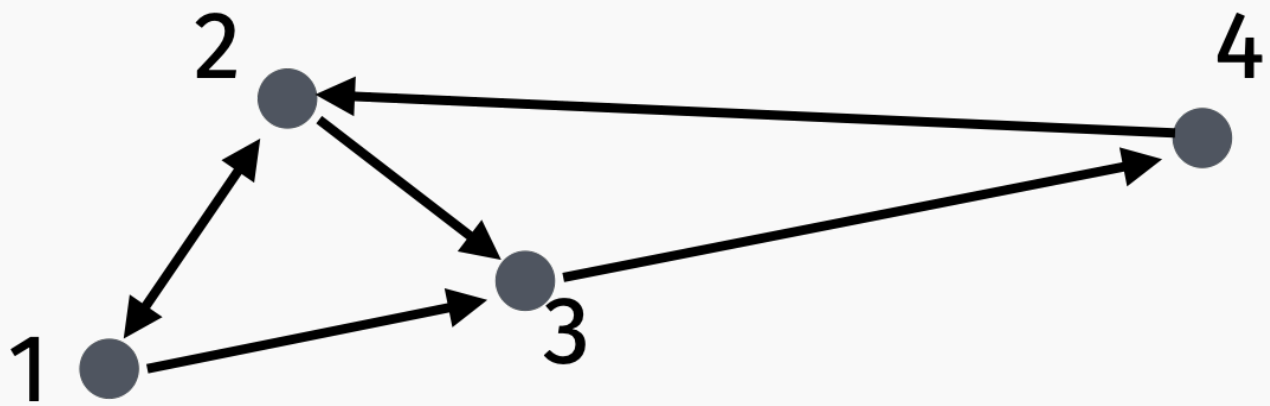
$k = j$

Definition (Navigable Graph)

A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$,

$$d(j, k) < d(j, i).$$

for some k in i's out neighborhood, $\mathcal{N}(i)$

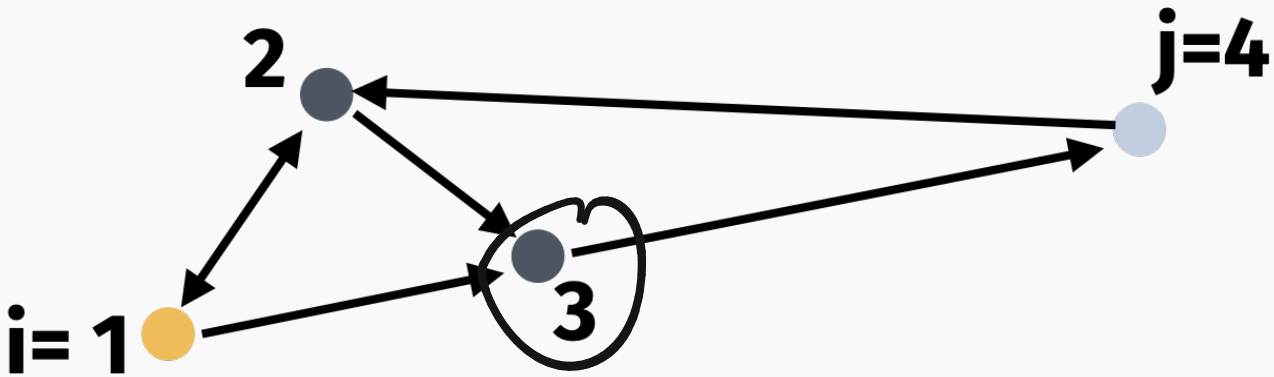


Definition (Navigable Graph)

A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$,

$$d(j, k) < d(j, i).$$

for some k in i's out neighborhood, $\mathcal{N}(i)$

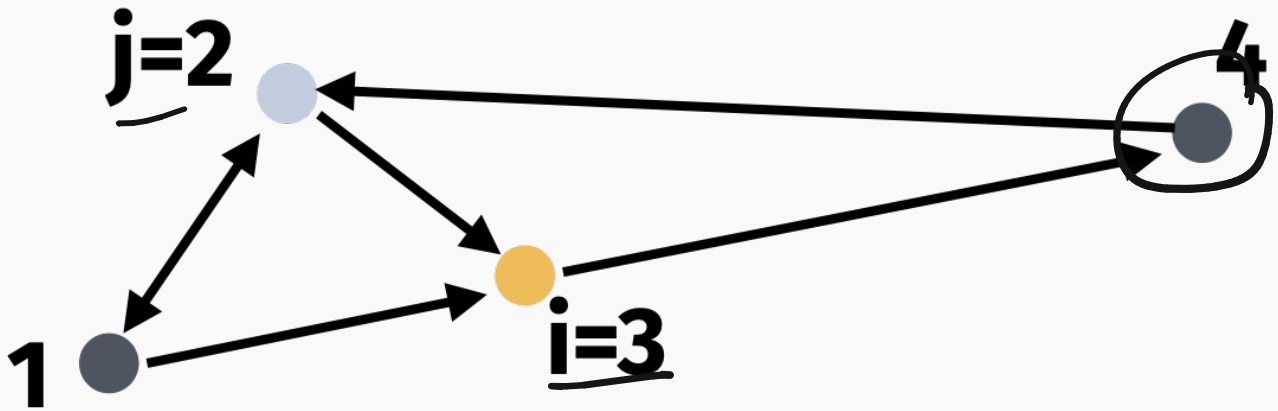


Definition (Navigable Graph)

A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$,

$$d(j, k) < d(j, i).$$

for some k in i's out neighborhood, $\mathcal{N}(i)$

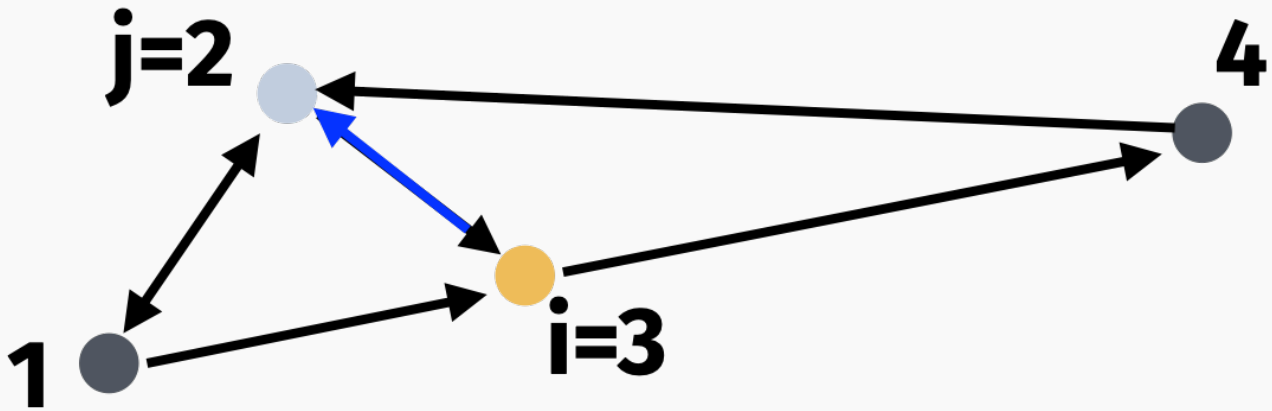


Definition (Navigable Graph)

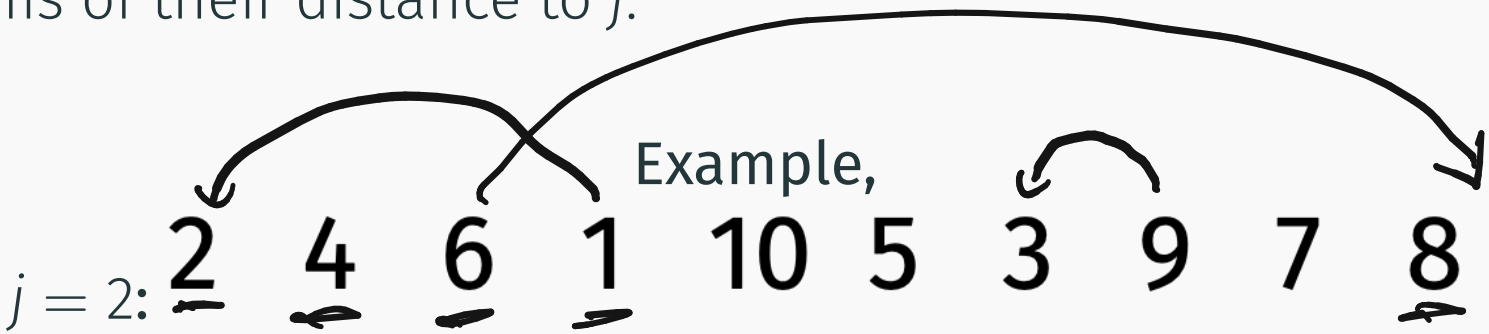
A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$,

$$d(j, k) < d(j, i).$$

for some k in i's out neighborhood, $\mathcal{N}(i)$



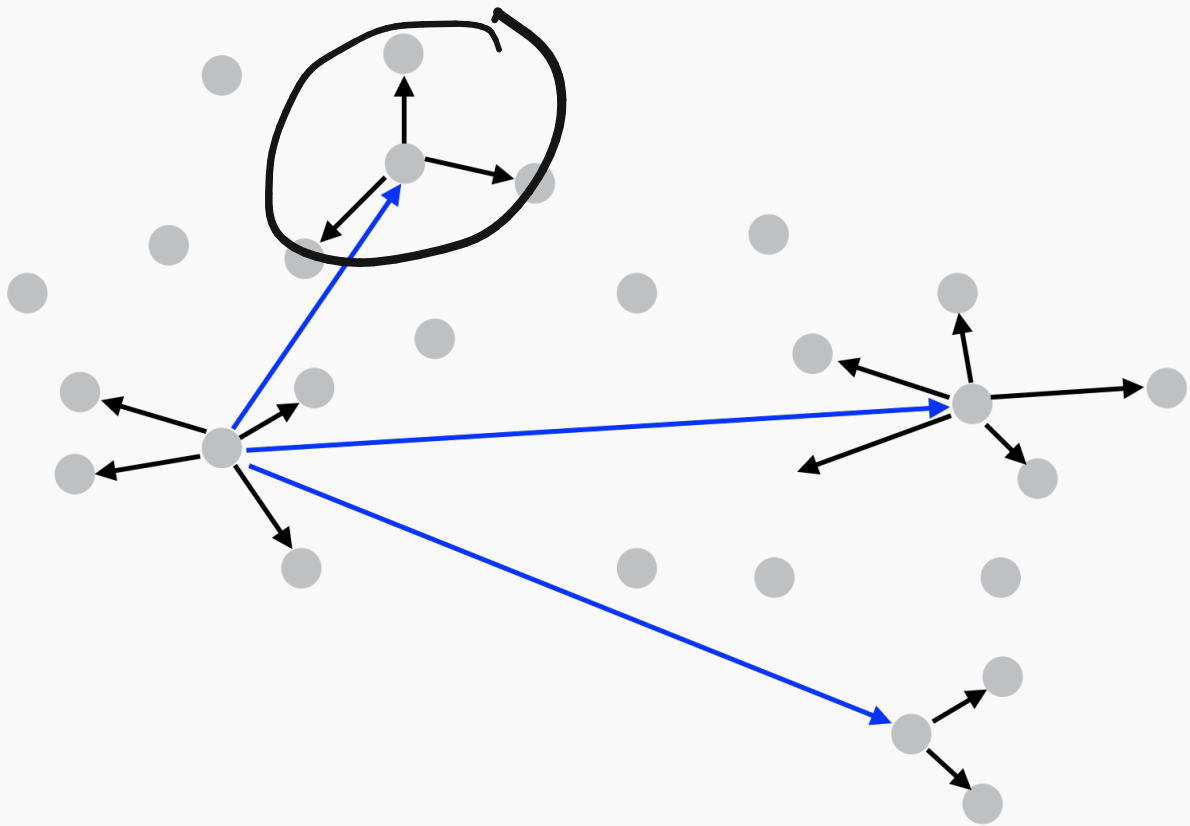
Consider a single target node j. Write down all other points in terms of their distance to j.



Navigability Requirement: Need at least one "left pointing" edge for every node in the list.

Claim

Any dataset $\mathbf{x}_1, \ldots, \mathbf{x}_n$ admits a navigable graph with average degree $O(\sqrt{n \log n})$, under any distance function.



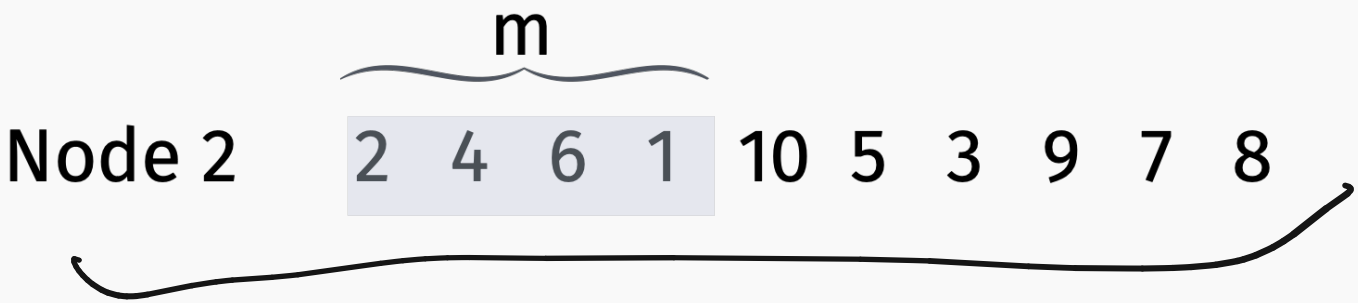
Idea: Combine long- and short-range edges.

Construction: Choose $m < n$.

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.

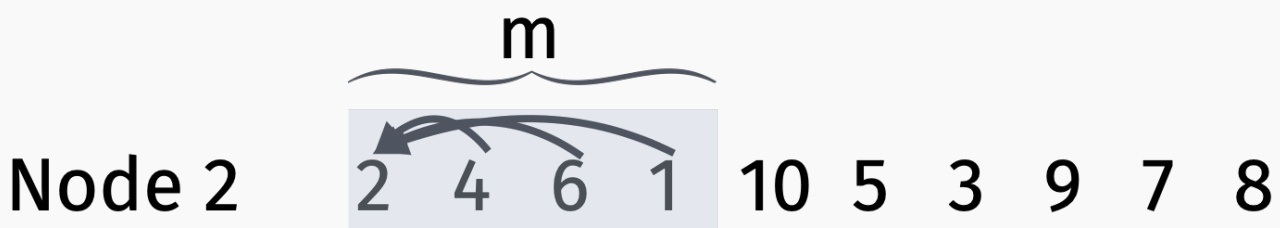
Construction: Choose $m < n$.

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.



Construction: Choose $m < n$.

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.



Construction: Choose $m < n$.

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.



LEFT REGION

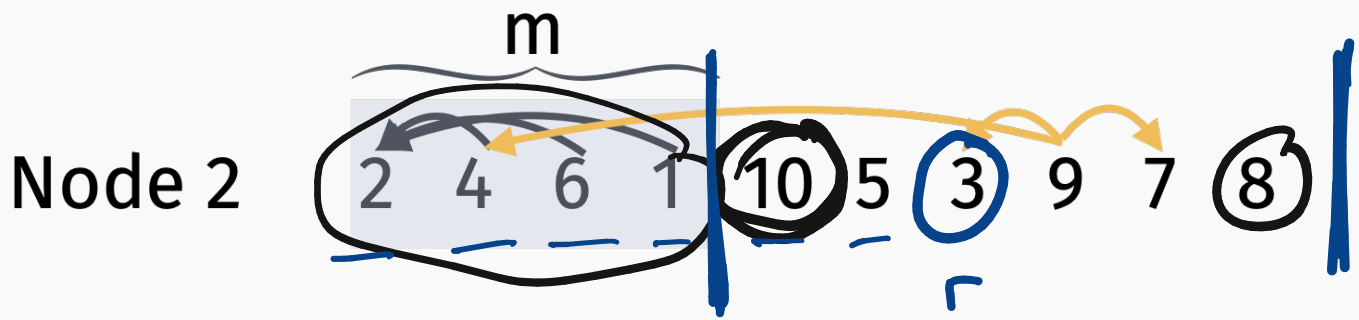
1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.



Analysis: All constraints in the left shaded region are trivially satisfied.

RIGHT REGION

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.



Analysis: Consider any node outside the left region. The probability a random edge points left is at least m/n

$\frac{r-1}{n}$ $\quad r \geq m+1 \quad$ $\frac{r-1}{n} \geq \frac{m}{n}$

RIGHT REGION

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m}\log n$ uniformly random out-edges to every node.



Analysis: Consider any node outside the left region. The probability a random edge points left is at least $\frac{m}{n}$.

RIGHT REGION

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.



Analysis: Consider any node outside the left region. The probability a random edge points left is at least $\frac{m}{n}$. So the probability that none point left is:

$$\left(1 - \frac{m}{n}\right)^{3\frac{n}{m}\log(n)} = \left(\left(1 - \frac{m}{n}\right)^{n/m}\right)^{3\log(n)}$$

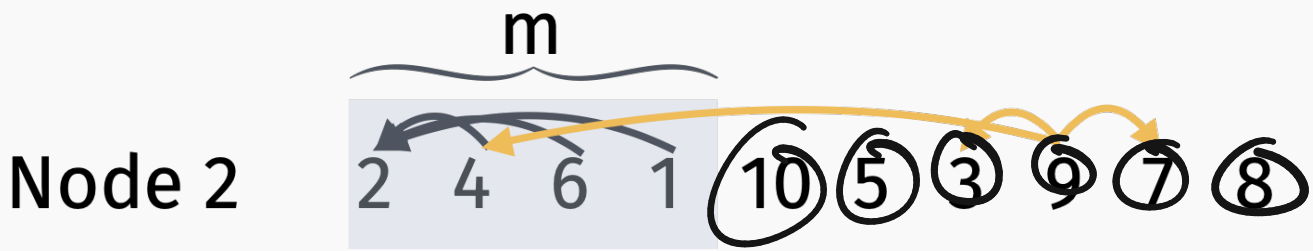
1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m}\log n$ uniformly random out-edges to every node.



Analysis: Consider any node outside the left region. The probability a random edge points left is at least $\frac{m}{n}$. So the probability that none point left is:

$$\leq \left(1-\frac{m}{n}\right)^{3\frac{n}{m}\log n} \leq \left(\frac{1}{e}\right)^{3\log n} \leq \frac{1}{n^3}.$$

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m}\log n$ uniformly random out-edges to every node.



Analysis: Consider any node outside the left region. The probability a random edge points left is at least $\frac{m}{n}$. So the probability that none point left is:

$$\leq \left(1-\frac{m}{n}\right)^{3\frac{n}{m}\log n} \leq \left(\frac{1}{e}\right)^{3\log n} \leq \frac{1}{n^3}.$$

By a union bound, we have at least one left-pointing edge, for all i, j, with probability at least

1. Add an edge from i to j if i is one of j's m closest neighbors.
2. Add $3\frac{n}{m}\log n$ uniformly random out-edges to every node.



Analysis: Consider any node outside the left region. The probability a random edge points left is at least $\frac{m}{n}$. So the probability that none point left is:

$$\leq \left(1 - \frac{m}{n}\right)^{3\frac{n}{m}\log n} \leq \left(\frac{1}{e}\right)^{3\log n} \leq \frac{1}{n^3}.$$

By a union bound, we have at least one left-pointing edge, for all i, j, with probability at least $\left(1 - \frac{1}{n}\right)$.

Construction: Choose $m < n$.

1. Add an edge from j to i if j is one of i's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.

Total number of edges: $nm + n \cdot 3\frac{n}{m} \log n$.

Minimize by setting $m =$.

$$nm = n \cdot 3\frac{n}{m} \log(n)$$

Construction: Choose $m < n$.

1. Add an edge from j to i if j is one of i's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.

Total number of edges: $nm + n \cdot 3\frac{n}{m} \log n$.

Minimize by setting $m = \sqrt{3n \log n}$.

Construction: Choose $m < n$.

1. Add an edge from j to i if j is one of i's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.

Total number of edges: $nm + n \cdot 3\frac{n}{m} \log n$.

Minimize by setting $m = \sqrt{3n \log n}$.

Average out-degree: $O(\sqrt{n \log n})$.

Construction: Choose $m < n$.

1. Add an edge from j to i if j is one of i's m closest neighbors.
2. Add $3\frac{n}{m} \log n$ uniformly random out-edges to every node.

Total number of edges: $nm + n \cdot 3\frac{n}{m} \log n$.

Minimize by setting $m = \sqrt{3n \log n}$.

Average out-degree: $O(\sqrt{n \log n})$.



Moreover, the constructed graph has "depth" 2: greedy search will converge in 2 steps for any query in the dataset.

Claim (Nearly Matching Lower Bound)

Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *be random vectors in* $\{-1, 1\}^m$ *where* $m = O(\log n)$. *With high probability, any navigable graph for* $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *under Euclidean distance requires average out-degree* $\Omega(n^{1/2-\epsilon})$ *for any fixed constant* ϵ.

Claim (Nearly Matching Lower Bound)

Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *be random vectors in* $\{-1, 1\}^m$ *where* $m = O(\log n)$. *With high probability, any navigable graph for* $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *under Euclidean distance requires average out-degree* $\Omega(n^{1/2-\epsilon})$ *for any fixed constant* ϵ.

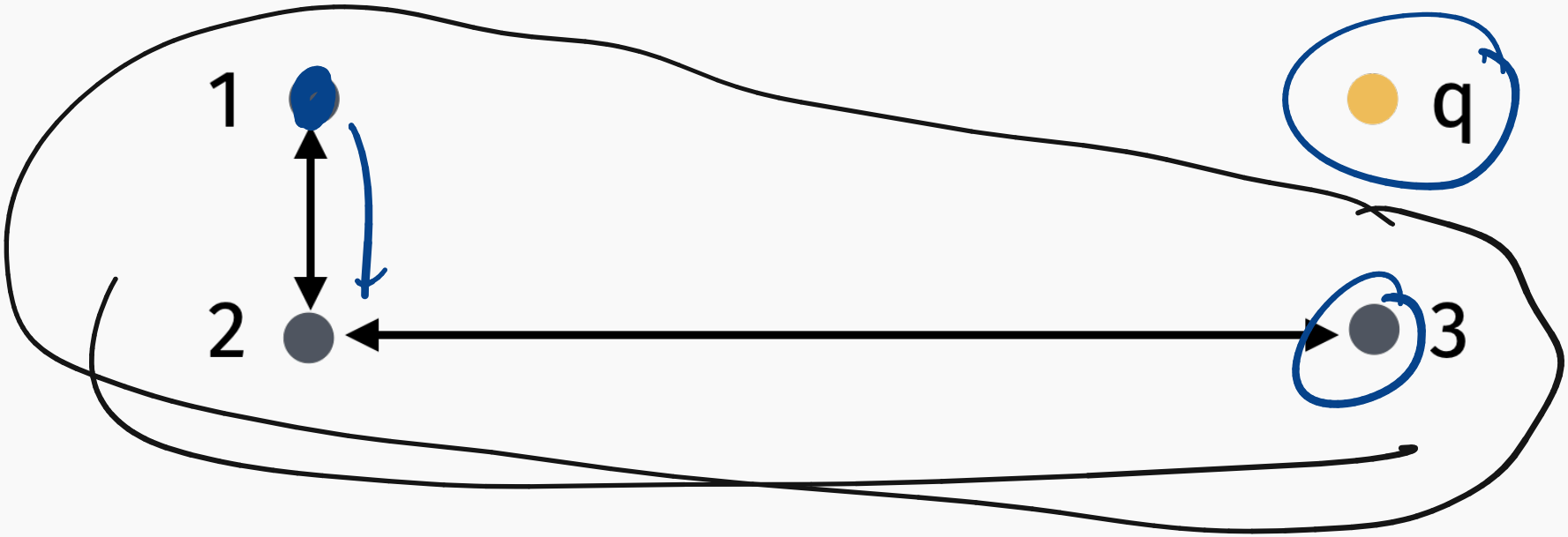
In practice, it seems like many fewer edges are required. Often 50-100 suffice even for data sets with billions of points.

Tons of open questions:

- (Why do some datasets admit sparser navigable graphs?)
- Can we find the sparsest navigable graph efficiently.
- (Can we translate guarantees on navigability to worst-case runtime/approximation guarantees?

Tons of open questions:

- Why do some datasets admit sparser navigable graphs?
- Can we find the sparsest navigable graph efficiently.
- Can we translate guarantees on navigability to worst-case runtime/approximation guarantees?



QUESTIONS?