Almost Navigable Graphs

2nd Workshop on Vector Databases at VLDB 2026

Pratyush Avi, Christopher Musco

New York University

Problem Setup

Let $P = \{x_1, ..., x_n\} \subset \mathbb{R}^r$ be a set of vectors and let $d : \mathbb{R}^r \times \mathbb{R}^r \mapsto \mathbb{R}$ be some distance function between vectors.

Nearest Neighbor Search Objective:

Given any query vector $q \in \mathbb{R}^r$, we want to return $x^* \in P$ s.t.

$$x^* = \underset{x \in P}{\operatorname{argmin}}\, d(x, q)$$

Problem Setup

Let $P = \{x_1, ..., x_n\} \subset \mathbb{R}^r$ be a set of vectors and let $d : \mathbb{R}^r \times \mathbb{R}^r \mapsto \mathbb{R}$ be some distance function between vectors.

Nearest Neighbor Search Objective:

Given any query vector $q \in \mathbb{R}^r$, we want to return $x^* \in P$ s.t.

$$x^* = \underset{x \in P}{\operatorname{argmin}}\, d(x, q)$$

Common choices for d are:

- Euclidean distance: $d(x, y) = \|x - y\|_2$
- Cosine Similarity: $d(x, y) = -\langle x, y \rangle$

Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

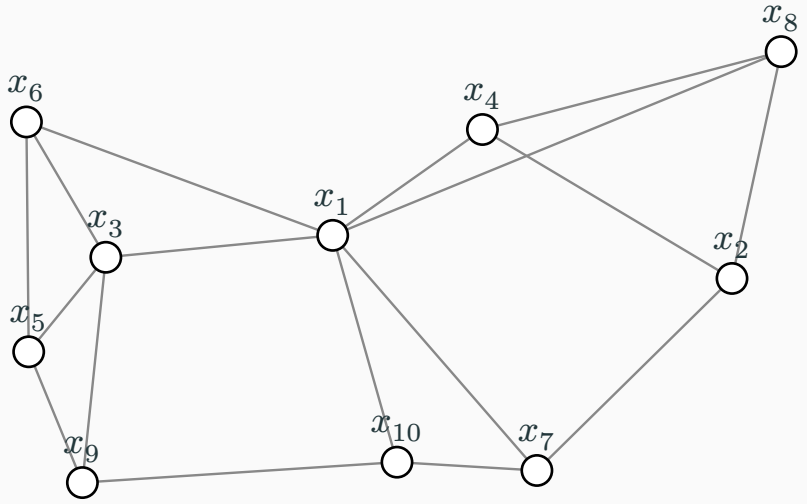
Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

Key Idea

Pre-process the dataset into a graph. Then, run greedy search on the graph.



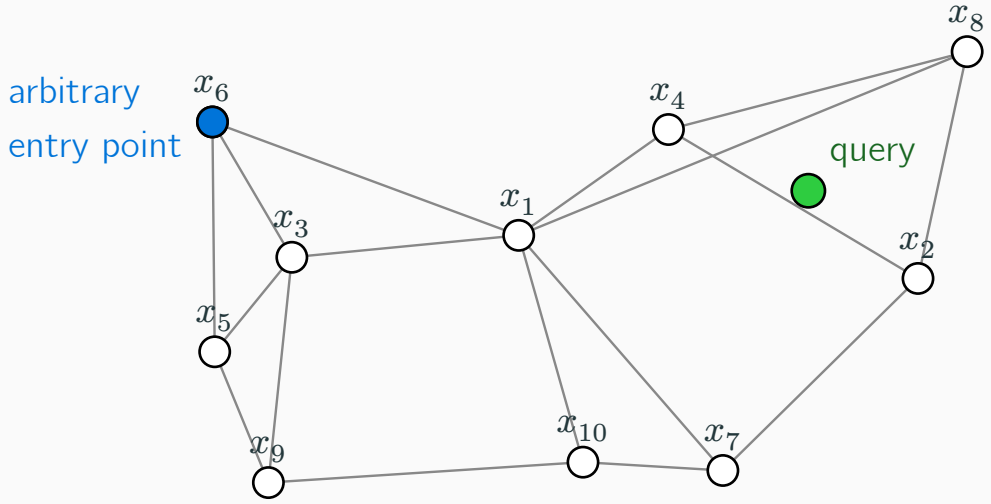
Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

Key Idea

Pre-process the dataset into a graph. Then, run greedy search on the graph.



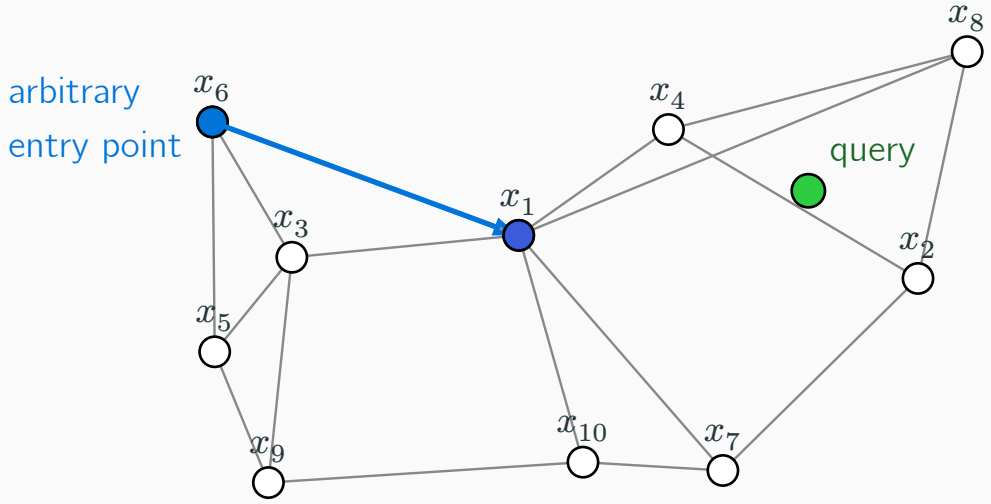
Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

Key Idea

Pre-process the dataset into a graph. Then, run greedy search on the graph.



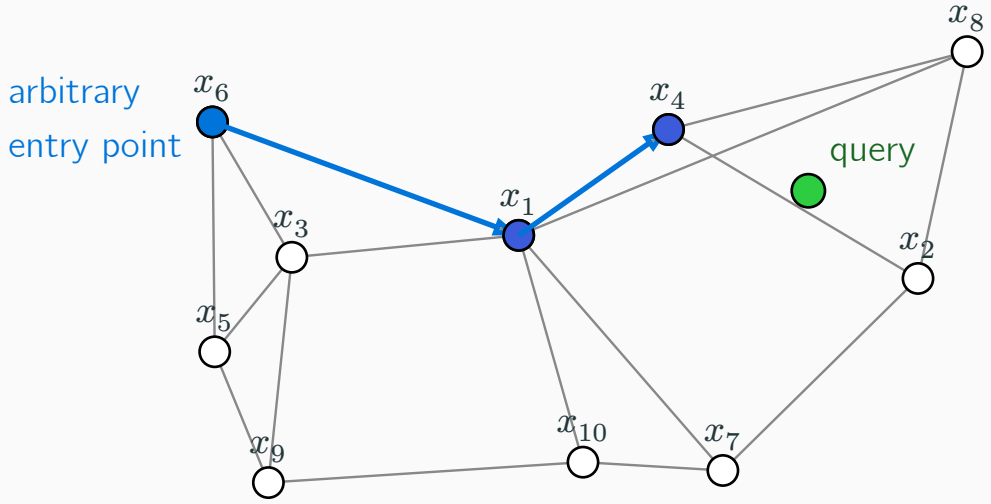
Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

Key Idea

Pre-process the dataset into a graph. Then, run greedy search on the graph.



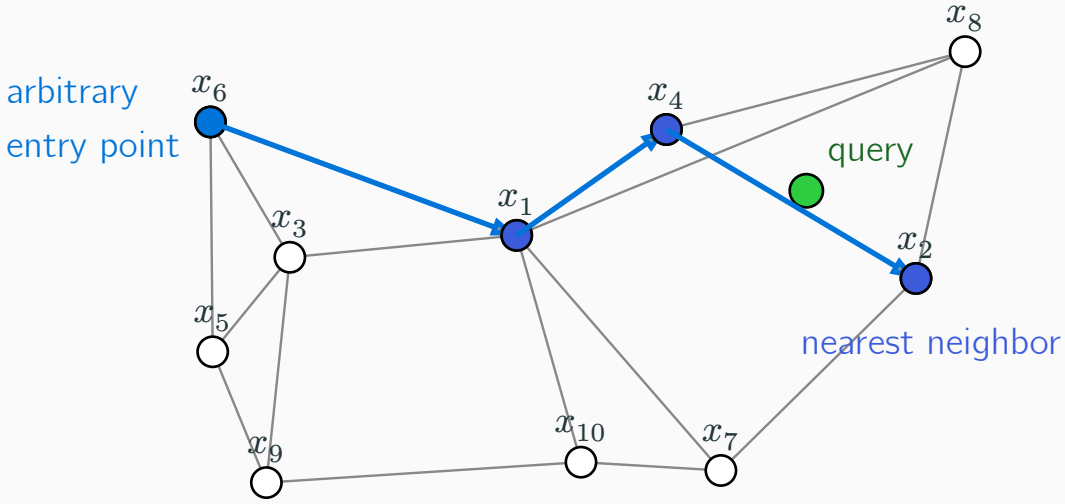
Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

Key Idea

Pre-process the dataset into a graph. Then, run greedy search on the graph.



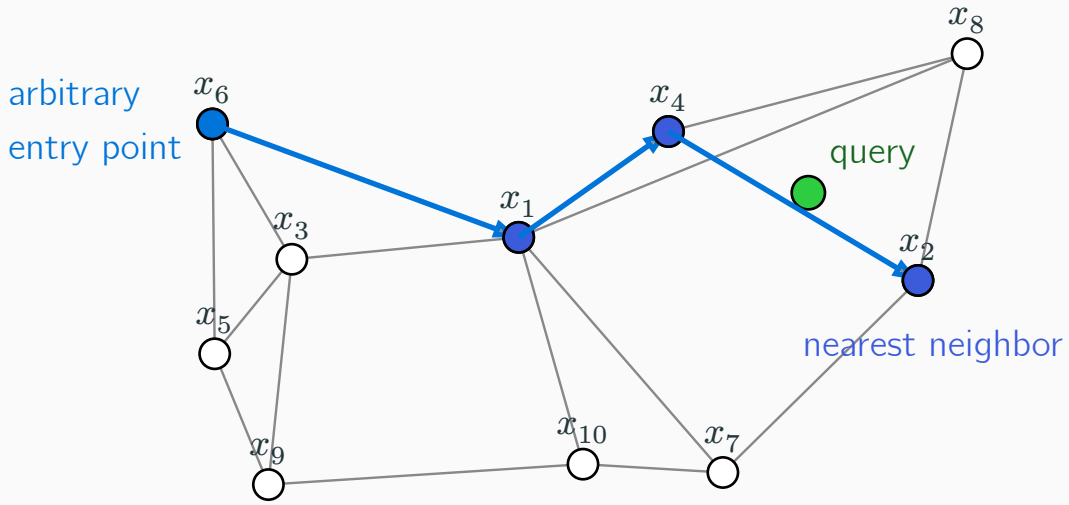
Graph-Based Approximate Nearest Neighbor Search

Popular approaches include: locality sensitive hashing, clustering, tree-based methods.

Graph-based methods for vector search have gained popularity in recent times.

Key Idea

Pre-process the dataset into a graph. Then, run greedy search on the graph.



E.g. Microsoft DiskANN (Subramanya, Devvrit, Kadekodi, Krishaswamy, and Simhadri 2019), NSG (Fu, Xiang, Wang, and Cai 2019), and HNSW (Malkov and Yashunin 2020).

Building Search Graphs

Question:

What makes a graph good for vector search?

Building Search Graphs

Question:

What makes a graph good for vector search?

Graph Navigability has been identified as a desirable property.

Navigability

Informally: Every node in the graph has an out-edge that gets closer to any other node.

Navigability

Informally: Every node in the graph has an out-edge that gets closer to any other node.

Definition (Navigable Graph):
A graph $G = (P, E)$ is navigable if, for all pairs $x_i, x_j \in P$, there is a directed edge $(x_i, u) \in E$ such that

$$d(u, x_j) < d(x_i, x_j).$$

Navigability

Informally: Every node in the graph has an out-edge that gets closer to any other node.

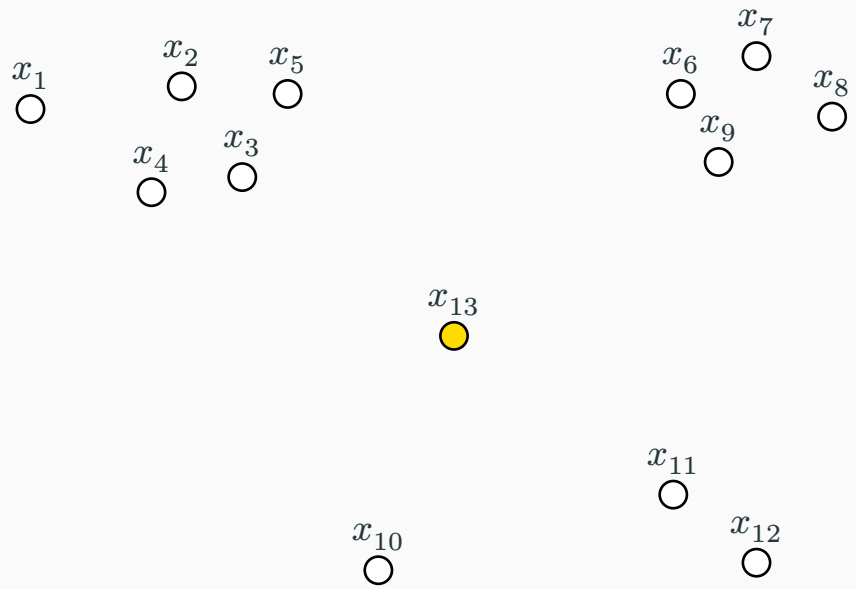
Definition (Navigable Graph):
A graph $G = (P, E)$ is navigable if, for all pairs $x_i, x_j \in P$, there is a directed edge $(x_i, u) \in E$ such that

$$d(u, x_j) < d(x_i, x_j).$$

Consequence: Greedy search on navigable graphs always succeeds on *in-distribution* queries, i.e., if we query $x_j \in P$, it exactly returns x_j no matter where the search begins.

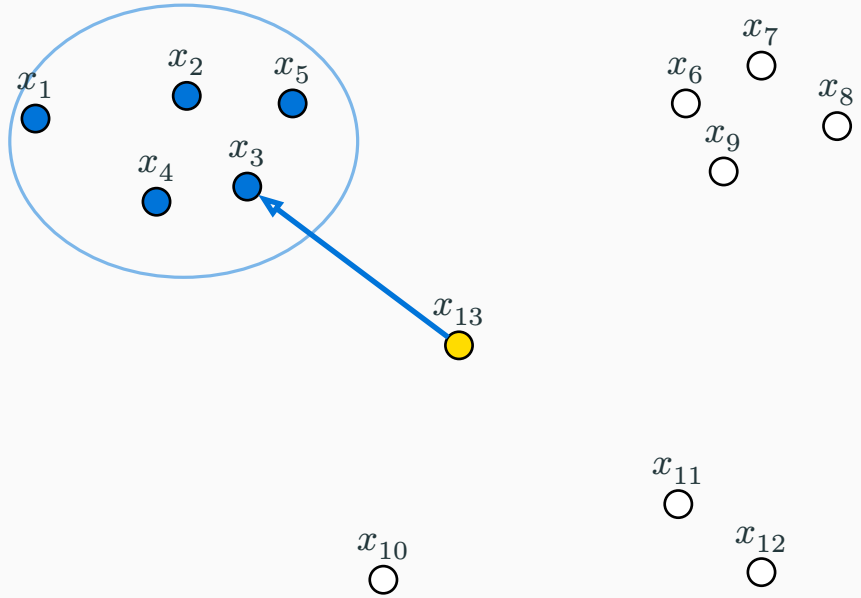
Visualizing Navigability

Consider the following pointset and focus on x_{13}'s out-neighborhood.



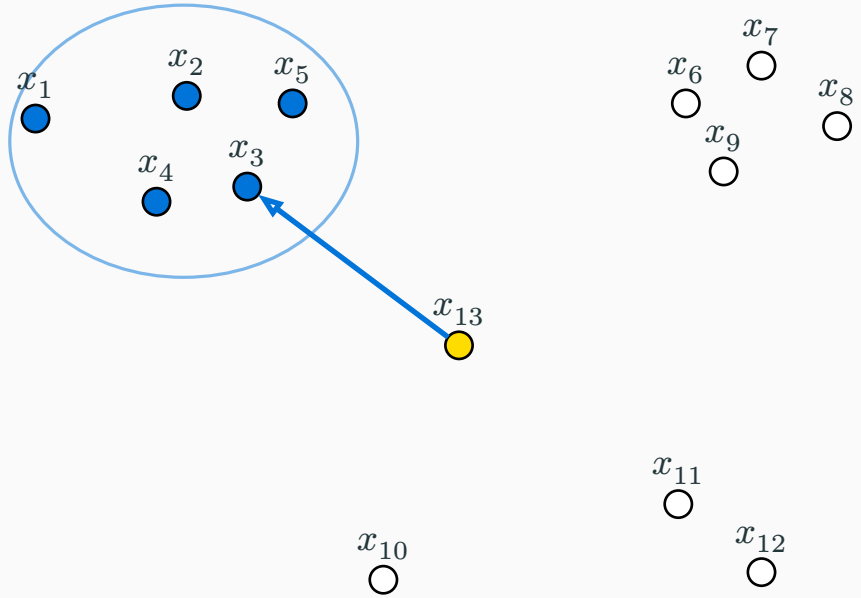
Visualizing Navigability

The edge (x_{13}, x_3) allows greedy navigation to points x_1, x_2, x_3, x_4, x_5.



Visualizing Navigability

The edge (x_{13}, x_3) allows greedy navigation to points x_1, x_2, x_3, x_4, x_5.

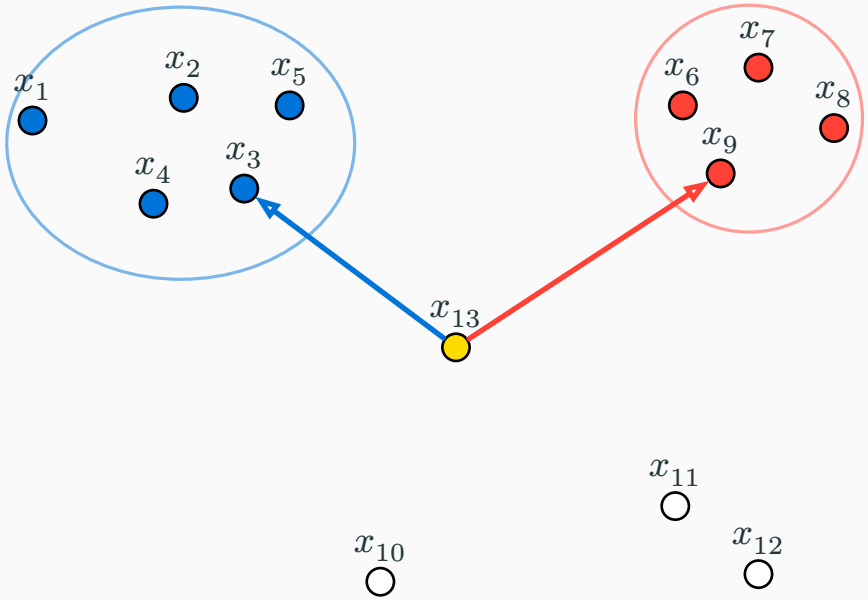


Remainder of the talk: An edge covers all the points it gets closer to.

E.g. (x_{13}, x_3) covers x_1, x_2, x_3, x_4, x_5

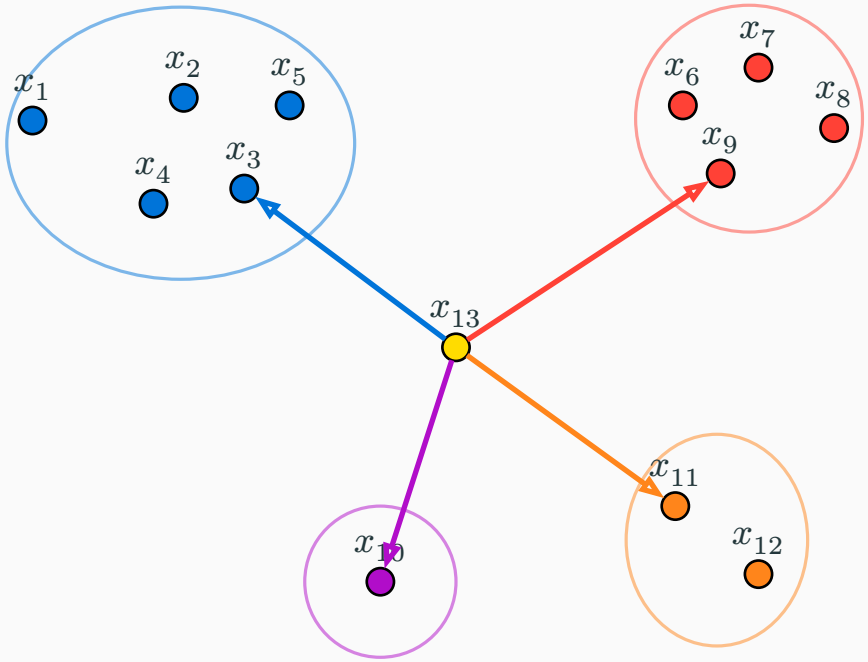
Visualizing Navigability

The edge (x_{13}, x_9) covers points x_6, x_7, x_8, x_9.



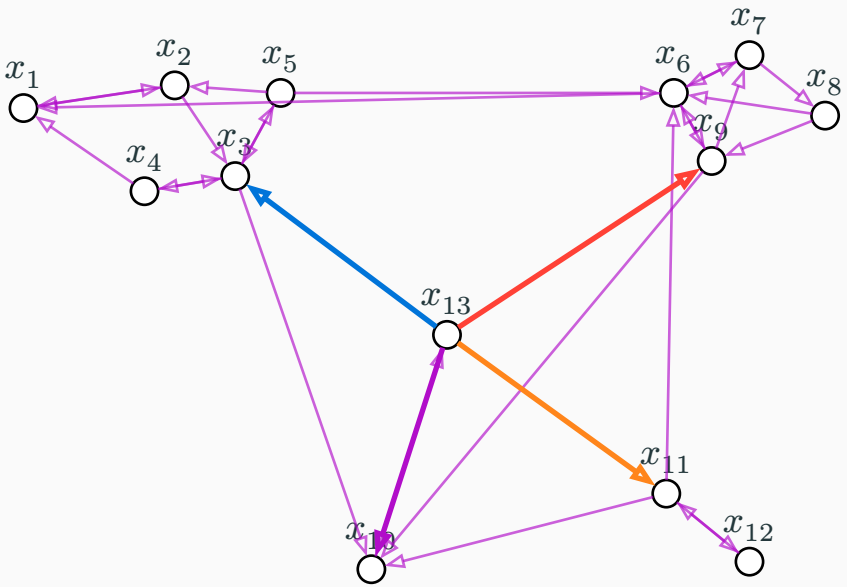
Visualizing Navigability

Adding edges (x_{13}, x_{10}) and (x_{13}, x_{11}) covers the remaining points.



Visualizing Navigability

Doing this for every point yields a **navigable graph**.



Theory of Navigability

So what do we know about navigability from a theoretical standpoint?

Theory of Navigability

So what do we know about navigability from a theoretical standpoint?

- Unfortunately, some datasets require dense graphs to achieve navigability: $\Omega(n^{1.5})$ **edges in the worst case** (Diwan, Gou, Musco, Musco, and Suel 2024)
  - Even true for random datapoints in $\Theta(\log n)$ dimensional Euclidean space

Theory of Navigability

So what do we know about navigability from a theoretical standpoint?

- Unfortunately, some datasets require dense graphs to achieve navigability: $\Omega(n^{1.5})$ **edges in the worst case** (Diwan, Gou, Musco, Musco, and Suel 2024)
  - Even true for random datapoints in $\Theta(\log n)$ dimensional Euclidean space
- Moreover, constructing navigable graphs requires $\Omega(n^2)$ **time** (Conway, Dhulipala, Farach-Colton, Johnson, Landrum, Musco, Shechter, Suel, and Wen 2026; Khanna, Padaki, and Waingarten 2026)

Navigability in Practice

Many billion-scale datasets require high-degree graphs to achieve navigability.

Dataset	Dimensions	# of points	Median Out Degree
BIGANN	128	1B	95
Yandex DEEP	96	1B	138
SPACEV1B	100	1.4B	500
FB SimSearchNet++	256	1B	1577

In practice, we often hope for graphs with degree 64 and 128.

Navigability in Practice: Hope?

Key Research Question

Are there relaxations of navigability that allow for sparser graphs and faster construction time?

Navigability in Practice: Hope?

Key Research Question

Are there relaxations of navigability that allow for sparser graphs and faster construction time?

Most practical graph-based methods build *close to navigable* graphs, usually in some heuristic sense.

Our goal is to develop a more formal theory.

Almost Navigable Graphs

Definition (γ-almost navigability):
A graph $G = (P, E)$ is γ-almost navigable if every point $p \in P$ has an out-neighborhood that covers at least γ fraction of the dataset.

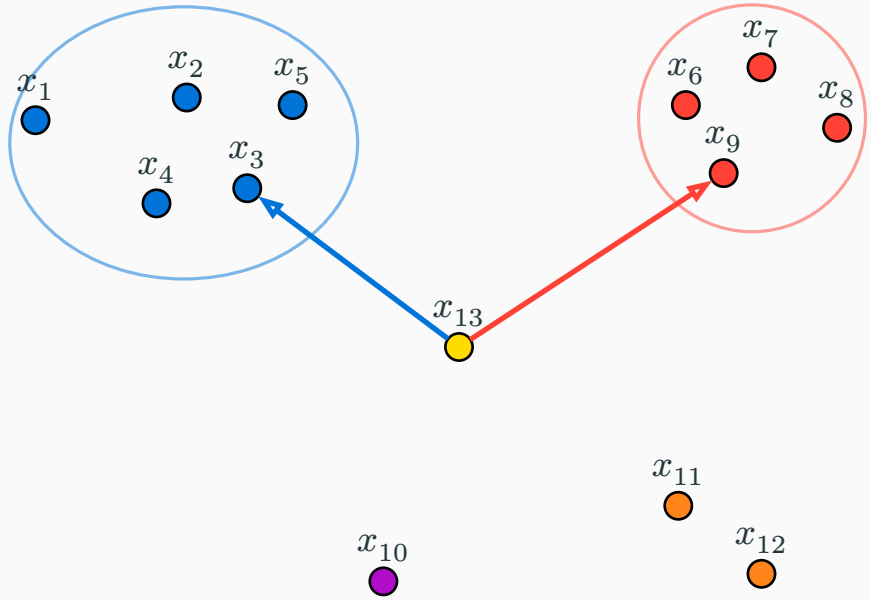
Almost Navigable Graphs

Definition (γ-almost navigability):
A graph $G = (P, E)$ is γ-almost navigable if every point $p \in P$ has an out-neighborhood that covers at least γ fraction of the dataset.

Where true navigability requires that the out-neighborhood covers all $n - 1$ nodes, γ-almost navigability only requires that we cover $\geq \gamma \cdot (n - 1)$ nodes.

Almost Navigable Graphs: Example

For example, just selecting the edges (x_{13}, x_3) and (x_{13}, x_9) results in a $\frac{3}{4}$-almost navigable out-neighborhood for x_{13}.



Selecting 2 edges covers *most* points.

Need 4 for full navigability.

Promising Initial Experiments

We constructed navigable graphs on real-world billion-scale datasets using the Robust Prune algorithm.

We tracked how many nodes were covered as one additional out-neighbor was added to each node.

Promising Initial Experiments

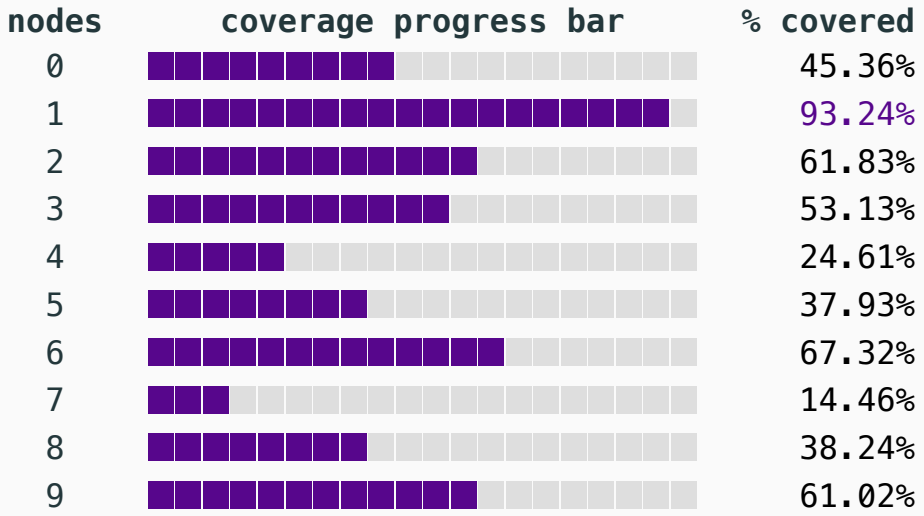
Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).

nodes	coverage progress bar	% covered
0		0.00%
1		0.00%
2		0.00%
3		0.00%
4		0.00%
5		0.00%
6		0.00%
7		0.00%
8		0.00%
9		0.00%

Every point starts with an empty out-neighborhood.

Promising Initial Experiments

Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).



nodes	coverage progress bar	% covered
0		45.36%
1		93.24%
2		61.83%
3		53.13%
4		24.61%
5		37.93%
6		67.32%
7		14.46%
8		38.24%
9		61.02%

```
Degree       : 1
Min-Coverage : 14.46%
```

Promising Initial Experiments

Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).

nodes	coverage progress bar	% covered
0		75.56%
1		97.52%
2		69.08%
3		85.20%
4		55.80%
5		83.96%
6		80.11%
7		53.41%
8		61.20%
9		78.00%

```
Degree       : 2
Min-Coverage : 53.41%
```

Promising Initial Experiments

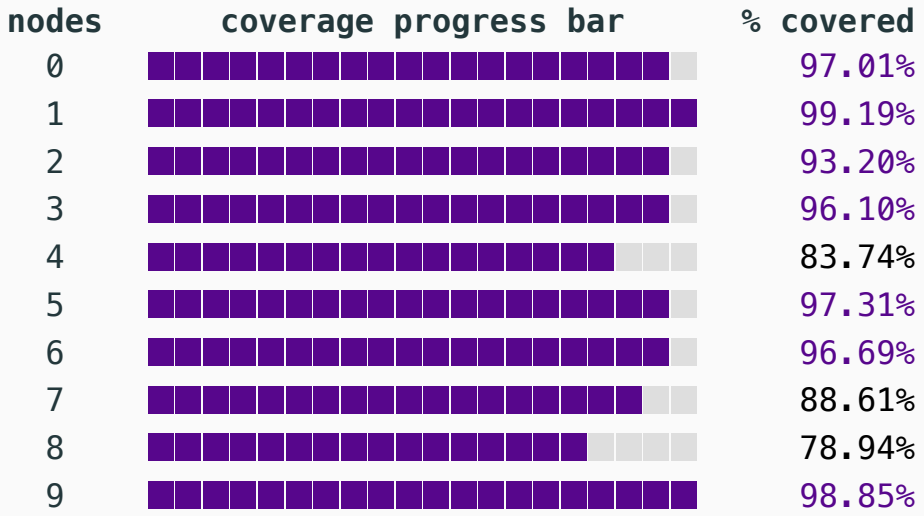
Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).

nodes	coverage progress bar	% covered
0		82.13%
1		98.82%
2		89.04%
3		95.85%
4		79.74%
5		90.19%
6		88.97%
7		71.46%
8		74.23%
9		89.68%

```
Degree       : 3
Min-Coverage : 71.46%
```

Promising Initial Experiments

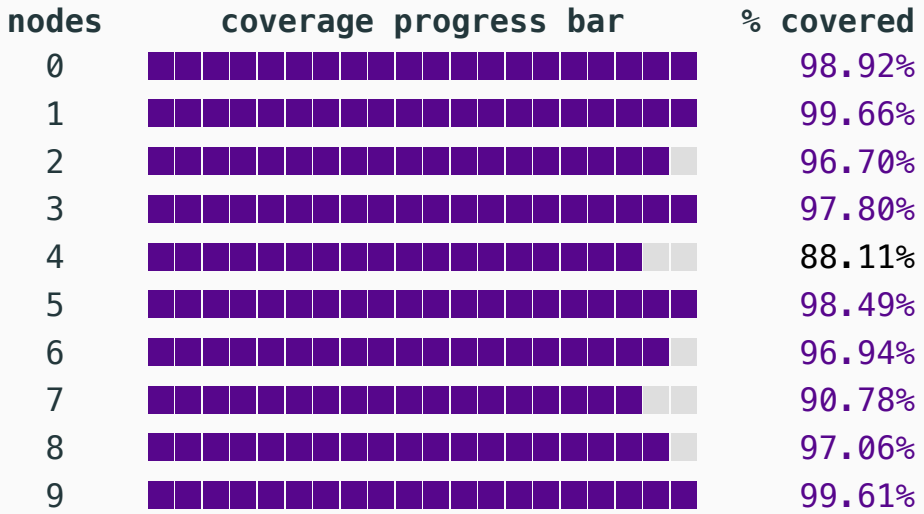
Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).




```
Degree       : 4
Min-Coverage : 78.94%
```

Promising Initial Experiments

Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).

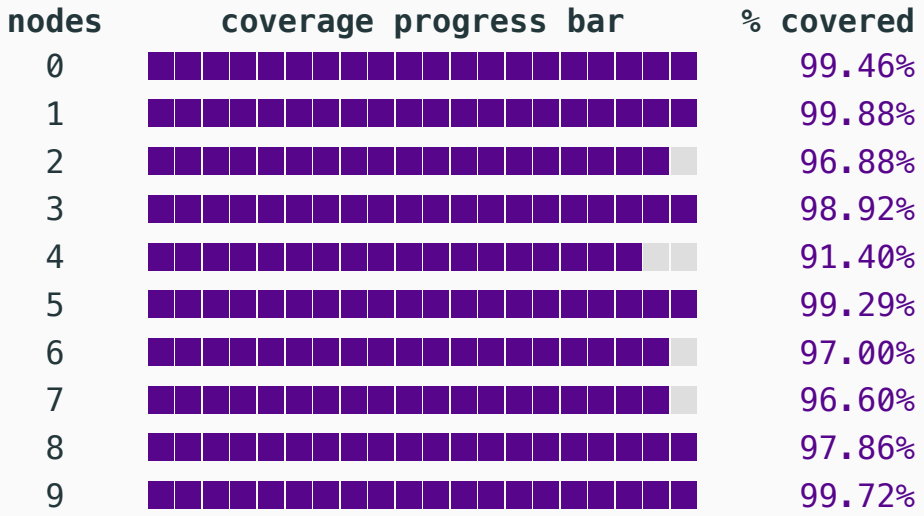


nodes	coverage progress bar	% covered
0		98.92%
1		99.66%
2		96.70%
3		97.80%
4		88.11%
5		98.49%
6		96.94%
7		90.78%
8		97.06%
9		99.61%

```
Degree       : 5
Min-Coverage : 88.11%
```

Promising Initial Experiments

Building out-neighborhoods for 10 random points from the BIGANN dataset (128 dimensions, 1 billion points).



Degree : 6
Min-Coverage : 91.40%

Promising Initial Experiments

It only took 6 edges per node to cover 90% of the dataset!!

Promising Initial Experiments

It only took **6 edges** per node to cover **90%** of the dataset!!

Let's keep adding edges to get to 100%.

- 14 edges per node, 99.72% coverage
- 22 edges per node, 99.99% coverage
- ...

Promising Initial Experiments

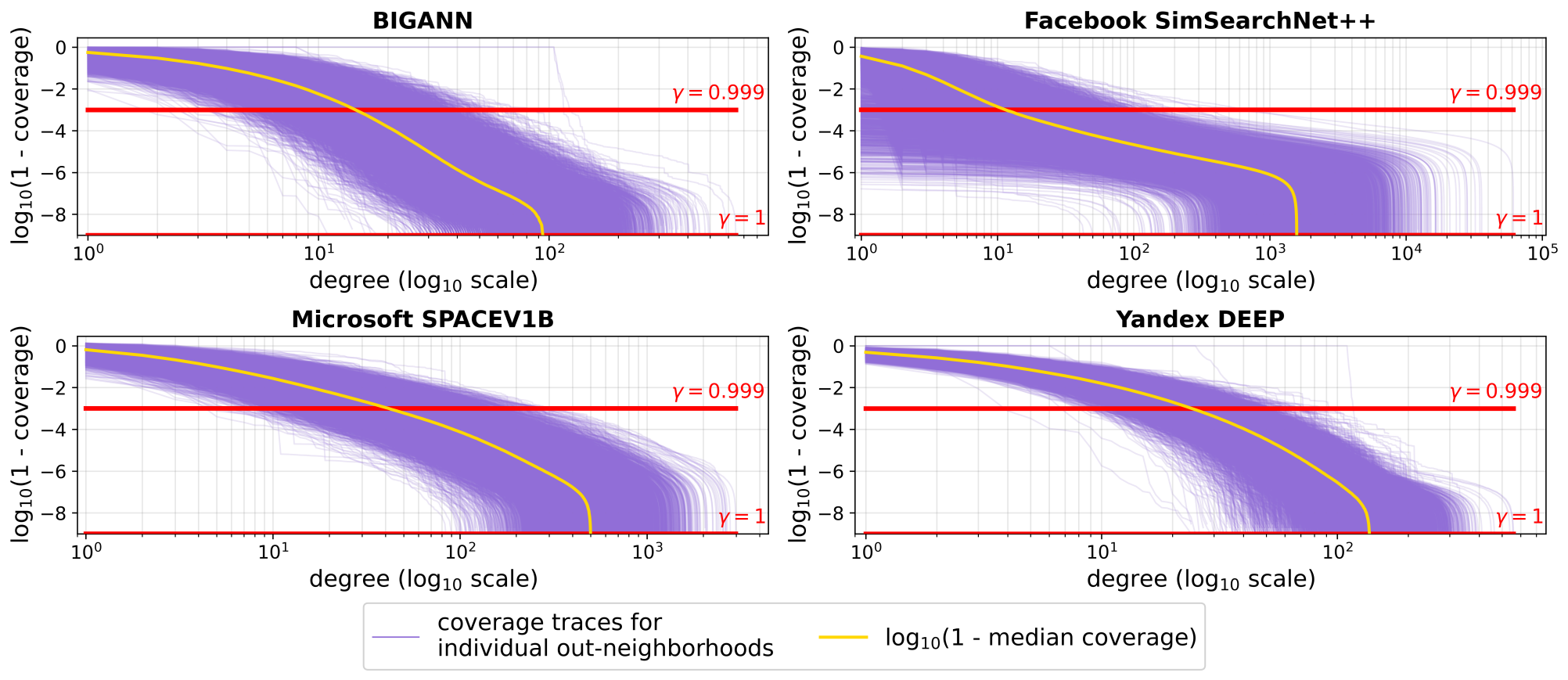
It only took 6 edges per node to cover 90% of the dataset!!

Let's keep adding edges to get to 100%.

- 14 edges per node, 99.72% coverage
- 22 edges per node, 99.99% coverage
- ...

As we saw earlier, we need 95 edges per node to achieve 100% coverage.

Trend Across Billion-point Datasets



99.9% coverage requires an order of magnitude fewer edges than 100% coverage!

Theoretical Results

Moreover, we have theory to support these findings!

Theorem (Avi and Musco (2026)):
Every point set P admits a γ-almost navigable graph with $\frac{4}{1-\gamma}$ average degree.

Theoretical Results

Moreover, we have theory to support these findings!

Theorem (Avi and Musco (2026)):
Every point set P admits a γ-almost navigable graph with $\frac{4}{1-\gamma}$ average degree.

E.g. we can achieve $\gamma = 99\%$-almost navigability with worst case average degree 400.

Theoretical Results

Further, we can construct these bounded-degree graphs in near-linear time!

Theorem (Avi and Musco (2026)):
Every point set P admits a γ-almost navigable graph with $O\left(\frac{1}{1-\gamma}\right)$ average degree that can be constructed in $\tilde{O}\left(\frac{n}{1-\gamma}\right)$ time.

Theoretical Results

Further, we can construct these bounded-degree graphs in near-linear time!

Theorem (Avi and Musco (2026)):
Every point set P admits a γ-almost navigable graph with $O\left(\frac{1}{1-\gamma}\right)$ average degree that can be constructed in $\tilde{O}\left(\frac{n}{1-\gamma}\right)$ time.

I.e. for constant γ, we achieve $O(1)$ average degree and $\tilde{O}(n)$ construction time.

Theoretical Results

Further, we can construct these bounded-degree graphs in near-linear time!

Theorem (Avi and Musco (2026)):
Every point set P admits a γ-almost navigable graph with $O\left(\frac{1}{1-\gamma}\right)$ average degree that can be constructed in $\tilde{O}\left(\frac{n}{1-\gamma}\right)$ time.

I.e. for constant γ, we achieve $O(1)$ average degree and $\tilde{O}(n)$ construction time.

Full navigability requires $\Omega(\sqrt{n})$ average degree and $\Omega(n^2)$ construction time.

Practical Performance

We performed beam-search on almost navigable graphs for various choices of γ and beam-width.

We compared search accuracy (**recall@k**) and search cost (**# of distance computations**).

Practical Performance

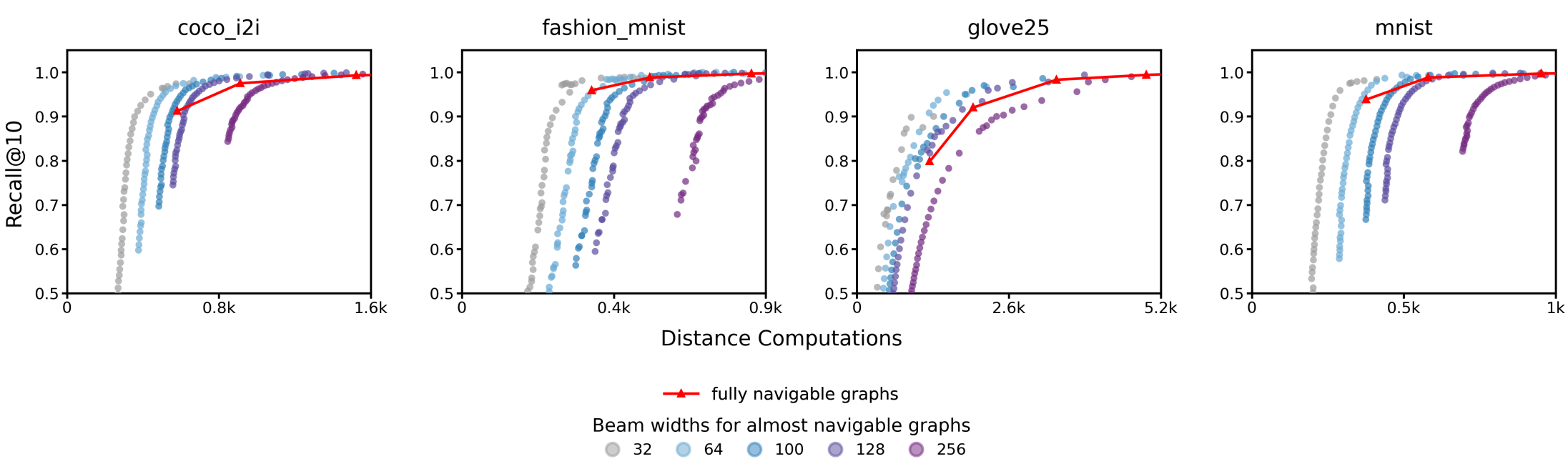
We performed beam-search on almost navigable graphs for various choices of γ and beam-width.

We compared search accuracy (**recall@k**) and search cost (**# of distance computations**).

Key Takeaway

Nearly 25-47% reduction in search cost at 46-55% smaller graph sizes to achieve similar or better recall@k, when compared to fully navigable graphs.

Practical Performance: recall@10



Thank you!

Paper link!

References

Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. “DiskANN: Fast Accurate Billion-Point Nearest Neighbor Search on a Single Node”. In: *Advances in Neural Information Processing Systems (NeurIPS) 2019*. 2019.

Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. “Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph”. In: *Proceedings of the VLDB Endowment* 12.5 (2019), Data acccessed at: url{https://github.com/ZJULearning/nsg}, pages 461–474.

Yu A. Malkov and D. A. Yashunin. “Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (2020), pages 824–836.

Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. “Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits”. In: *Advances in Neural Information Processing Systems (NeurIPS) 2024*. 2024.

Alex Conway, Laxman Dhulipala, Martin Farach-Colton, Rob Johnson, Ben Landrum, Christopher Musco, Yarin Shechter, Torsten Suel, and Richard Wen. “Efficiently Constructing Sparse Navigable Graphs”. In: *ACM-SIAM Symposium on Discrete Algorithms (SODA) 2026*. 2026.

Sanjeev Khanna, Ashwin Padaki, and Erik Waingarten. “Sparse Navigable Graphs for Nearest Neighbor Search: Algorithms and Hardness”. In: *ACM-SIAM Symposium on Discrete Algorithms (SODA) 2026*. 2026.

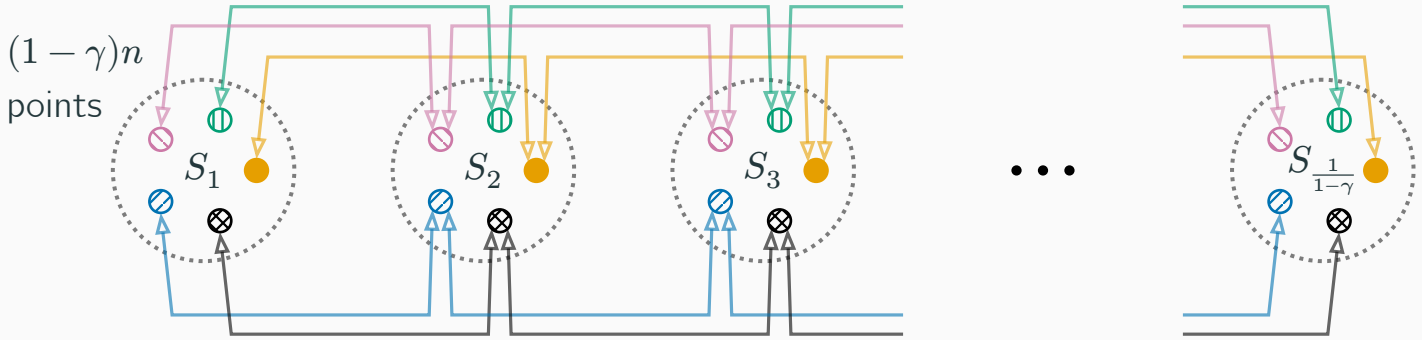
Pratyush Avi and Christopher Musco. “Almost Navigable Graphs”. In: *The 2nd Workshop on Vector Databases*. 2026.

AImost Navigable Graphs in the wild

Dataset	# of Points	Dim.	Median Out-Degree			% Change
			$\gamma = 1$	$\gamma = 0.995$	$\gamma = 0.990$	
Fashion-MNIST	60K	784	12	6	5	–58.3%
MNIST	60K	784	19	10	9	–52.6%
COCO-i2i	113.3K	512	27	12	10	–63%
Glove25	1.2M	25	50	7	6	–88%
Yandex DEEP	1B	96	138	16	12	–91.3%
BIGANN	1B	128	95	11	9	–90.5%
Facebook SimSearchNet++	1B	256	1577	7	6	–99.6%
Microsoft SPACEV1B	1.4B	100	500	19	14	–97.2%

Hard Instance

However, γ-almost navigable graphs don't enjoy the same theoretical guarantees as standard navigability.



Although this graph is γ-almost navigable, for any source point, greedy search will fail to return q for $n - \frac{1}{1-\gamma}$ in-distribution queries $q \in P$.

Hard Instance

Theorem (Avi and Musco (2026)):
For any n and $\gamma \in [0, 1)$, there is a set of n points P with a γ-almost navigable graph G, such that greedy search originating at any source $s \in P$ will not correctly return q for $n - \frac{1}{1-\gamma}$ in-distribution queries $q \in P$.

Near-Linear Time Construction Algorithm

Deterministic algorithm outline:

1. Arbitrarily partition the dataset into subsets of $O\left(\frac{1}{1-\gamma}\right)$ points
2. Add a clique to each subset
3. Let P' be the set of points with an *insufficient* out-neighborhood
4. Repeat step 1 with P' until $|P'| < O\left(\frac{1}{1-\gamma}\right)$
5. Connect any remaining points in P' with the entire dataset

Near-Linear Time Construction Algorithm

Deterministic algorithm outline:

1. Arbitrarily partition the dataset into subsets of $O\left(\frac{1}{1-\gamma}\right)$ points
2. Add a clique to each subset
3. Let P' be the set of points with an *insufficient* out-neighborhood
4. Repeat step 1 with P' until $|P'| < O\left(\frac{1}{1-\gamma}\right)$
5. Connect any remaining points in P' with the entire dataset

This algorithm can be made practical by using a random estimator in Step 3.