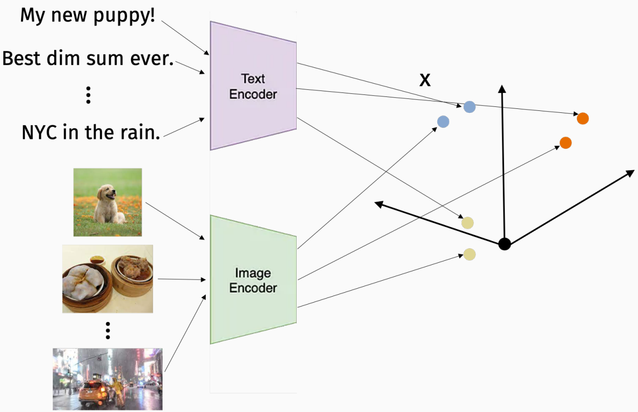
Beam search, navigable graphs, and nearest neighbor search

Christopher Musco, New York University

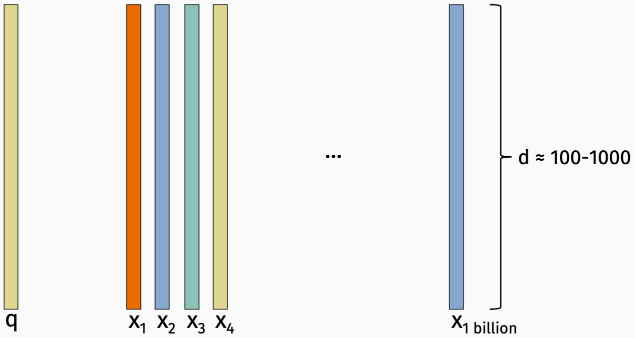
Based on collaborations with: Yousef Al-Jazzazi[1], Haya Diwan[1], Jinrui Gou[1], Cameron Musco[2], Torsten Suel[1]
Alex Conway[3], Laxman Dhulipala[4], Martin Farach-Colton[1], Rob Johnson[5], Ben Landrum[3], Yarin Shechter[1], Richard Wen[4]

[1]NYU [2]UMass [3]Cornell [4]UMD [5]VMWare



Use neural network (BERT, CLIP, etc.) to convert documents, images, and other media to high dimensional vectors. Matching results should have similar vector embeddings.

VECTOR SEARCH

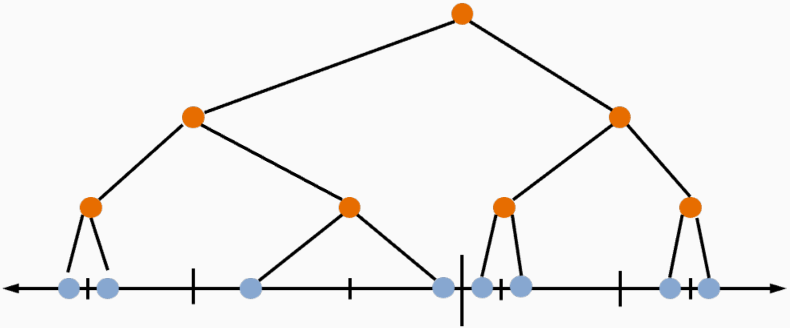


Finding results for a query amounts to finding the closest k vectors in a vector database $\mathcal{X}$. E.g., for $k = 1$, return:

$$\arg\min_{x \in \mathcal{X}} \|x - q\|.$$

Goal: Let $\mathcal{X}$ be a database of n vectors in $\mathbb{R}^d$. Find $\mathbf{x} \in \mathcal{X}$ minimizing $\|\mathbf{x} - \mathbf{q}\|$ for a query $\mathbf{q}$.

- **Naive linear scan:** $O(nd)$ time.
- **Multidimensional Search Trees:** Roughly $O(2^d \log n)$ time.



When d is large, we now have lots of other options available:

- Locality-sensitive hashing [Indyk, Motwani, 1998]
- Spectral hashing [Weiss, Torralba, and Fergus, 2008]
- Vector quantization/IVF data structures [Jégou, Douze, Schmid, 2009]
- Graph-based vector search [Malkov, Yashunin, 2016, Subramanya et al., 2019]

Key ideas behind all of these methods:

1. Do not guarantee exact nearest neighbors.
2. Trade worse space-complexity + preprocessing time for better time-complexity. I.e., preprocess database into data structure that uses $\Omega(n)$ space.

When d is large, we now have lots of other options available:

- Locality-sensitive hashing [Indyk, Motwani, 1998]
- Spectral hashing [Weiss, Torralba, and Fergus, 2008]
- Vector quantization/IVF data structures [Jégou, Douze, Schmid, 2009]
- Graph-based vector search [Malkov, Yashunin, 2016, Subramanya et al., 2019]

Key ideas behind all of these methods:

1. Do not guarantee exact nearest neighbors.
2. Trade worse space-complexity + preprocessing time for better time-complexity. I.e., preprocess database into data structure that uses $\Omega(n)$ space.

Theorem (Andoni, Indyk, FOCS 2006)

*For any approximation factor $c \geq 1$, there is a data structure based on **locality sensitive hashing** that, for any query* $\mathbf{q}$*, returns* $\tilde{\mathbf{x}}$ *satisfying:*

$$\|\tilde{\mathbf{x}} - \mathbf{q}\|_2 \leq c \cdot \min_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - \mathbf{q}\|_2$$

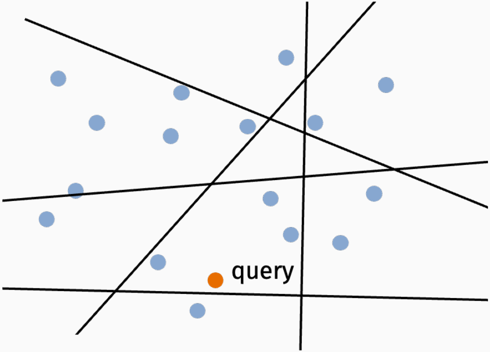
and uses:

- *Query Time:* $\tilde{O}\left(dn^{1/c^2}\right)$.
- *Data Structure Space Complexity:* $\tilde{O}\left(nd + n^{1+1/c^2}\right)$.

For exampe, if $c = 2$, query time scales with $n^{1/4}$, which is pretty amazing, at least in theory.

Rough idea behind LSH:

1. Pick a bunch of random hyperplanes.
2. Check which side of each hyperplane q lies on.
3. Return closest point that lies in the same region as q.
4. Repeat multiple times to avoid missing anything.



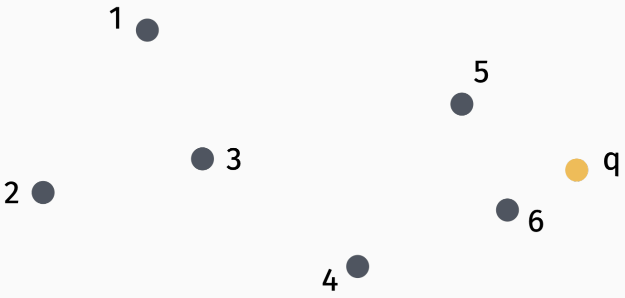
New(ish) kid on the block: Graph-based Search.

- **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]

Similar methods proposed for low-dimensions suggested in the 1990s by Arya, Mount, Kleinberg and others.

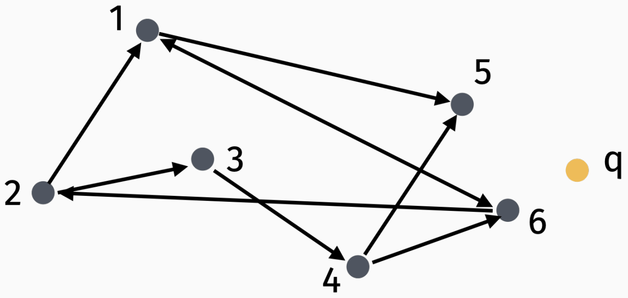
Connections to Milgram's famous "small world" experiments from the 1960s and later work on the small world phenomenon by Watts, Strogatz, Kleinberg, and others.

1. Construct a directed search graph over our dataset.



2. Run some variant of greedy search in the graph.

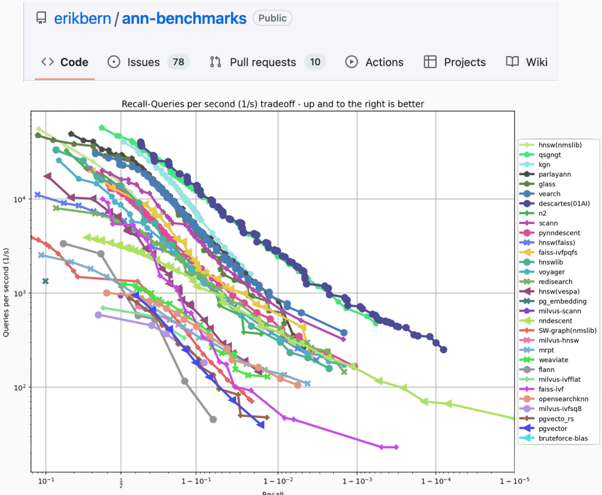
1. Construct a directed search graph over our dataset.



2. Run some variant of greedy search in the graph.

GRAPH-BASED SEARCH IN PRACTICE

Winning all of the competitions! Competitive with or better than the best space-partitioning methods.



Winning all of the competitions! Competitive with or better than the best space-partitioning methods.

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

Open theory challenge: Can we mathematically explain the empirical success of graph-based nearest-neighbor search methods?

1. Are there natural combinations of graph construction + search algorithm that lead to provably accurate approximate nearest neighbors?
2. Can we show these good approximations are returned quickly?

Computational Cost $\approx$ (graph degree) $\times$ (# of search steps).

A major reason to study theoretical guarantees is to provide a foundation for improving on existing methods.

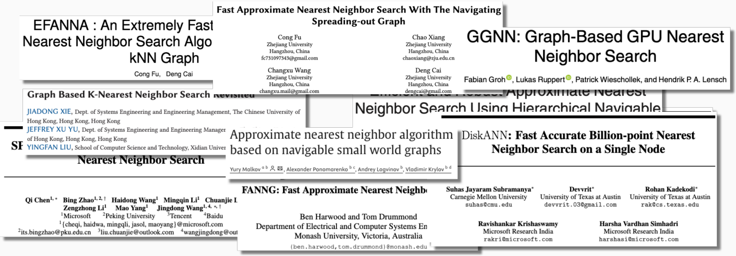
1. Are there natural combinations of graph construction + search algorithm that lead to provably accurate approximate nearest neighbors?
2. Can we show these good approximations are returned quickly?

Computational Cost $\approx$ (graph degree) $\times$ (# of search steps).

A major reason to study theoretical guarantees is to provide a foundation for improving on existing methods.

SEARCH GRAPH PROPERTIES

Dozens of papers on constructing good search graphs.



First Task: Come up with abstraction for what it means for a search graph, G, to be “good”.

Navigability is one of the most commonly listed desirable properties. Lends its name to methods like Navigable Spreading-out and Hierarchical Navigable Small World Graphs.

Definition (Navigable Graph)

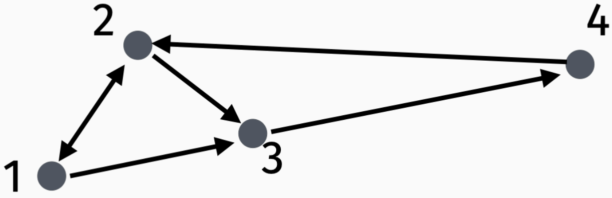
A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$, there is some $k \in \mathcal{N}(i)$ (i's out neighborhood) satisfying:

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

Definition (Navigable Graph)

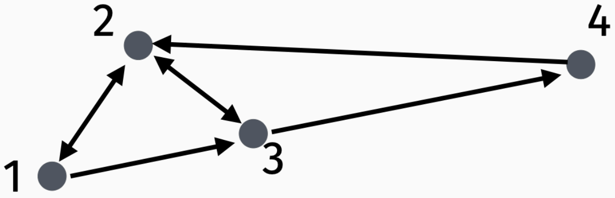
A directed graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot,\cdot)$ if, for all nodes i, for all $j \neq i$, there is some $k \in \mathcal{N}(i)$ (i's out neighborhood) satisfying:

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



Claim (Perfect Recall for Queries in Dataset)

For any starting node i and query q in our dataset, standard greedy search run on a navigable graph G returns q itself.



Moreover, sparse navigable graphs exist!

- **2-dimensional Euclidean space:** The Delaunay graph is navigable. This graph has average degree $O(1)$.
- **d-dimensional Euclidean space:** The Sparse Neighborhod Graph of Arya and Mount [SODA, 1993] is navigable and has average degree $O(2^d)$.
- **Arbitary metric:** Can always construct a navigable graph with average degree $O(\sqrt{n})$ [Diwan, Gou, Musco, Musco, Suel, NeurIPS 2024].

We recently showed that a navigable graph with near-optimal sparsity can be computed in $\tilde{O}(n^2)$ time for any metric and dataset [Conway, Dhulipala, Farach-Colton, Johnson, Landrum, Musco, Shechter, Suel, Wen, 2025].

FAILED APPROXIMATION FOR GENERIC QUERIES

Unfortunately, greedy search on a navigable graph fails to provide any meaningful approximation for general queries, q (that are not exactly in our dataset).

Recall that, if currently at node i, greedy search moves to $j^* = \arg\min_{j \in \mathcal{N}(i)} d(j, q)$, or terminates if $d(j^*, q) > d(i, q)$.

If we want stronger theoretical guarantees, we seemingly have two options:

1. **Consider stronger graph properties.**
2. Consider a stronger search algorithm.

TOWARDS APPROXIMATION GUARANTEES

If we want stronger theoretical guarantees, we seemingly have two options:

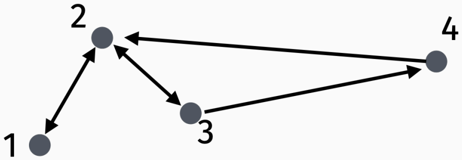
1. Consider stronger graph properties.
2. Consider a stronger search algorithm.

Inspired by Microsoft's DiskANN method, Indyk and Xu recently studied α-shortcut reachability:

Definition (α-shortcut Reachable Graph)

For $\alpha \geq 1$, G is α-shortcut reachable for for a point set $1, \ldots, n$ and distance function d if, for all nodes i, for all $j \neq i$, there is some $k \in \mathcal{N}(i)$ satisfying:

$$d(j, k) < \frac{1}{\alpha} d(j, i).$$



Theorem (Indyk-Xu, NeurIPS 2023)

For <u>any</u> query q, greedy search run on an α-shortcut reachable graph is guaranteed to return x' satisfying

$$d(x', q) \leq \frac{1+\alpha}{1-\alpha} \cdot \min_{x \in \{1,\ldots,n\}} d(x, q).$$

Recently improved to $\frac{\alpha}{\alpha-1}$ for Euclean metrics and extended to k-nearest neighbor search for $k > 1$ by [Gollapudi, Krishnaswamy, Shiragur, Wardhan, ICML 2025].

As expected from our counter example earlier, both bounds are vacuous when $\alpha = 1$, which corresponds to navigability.

If we want stronger theoretical guarantees, we seemingly have two options:

1. Consider stronger graph properties.
2. **Consider a stronger search algorithm (our work).**

- The sparsest α-shortcut reachable graph is always denser than the sparsest navigable graph. A random point set in $O(\log n)$ dimensions requires $\Omega(n)$ degree to be α-shortcut reachable, but $O(\sqrt{n})$ degree to be navigable.[1]
- Since they hold even for standard greedy search, these results do not explain the empirical success of widely used improved greedy search strategies like beam search.

[1] Indyk-Xu show that there is always an α-shortcut reachable graph with degree $\alpha^{O(q)}$, where q is the "double dimension" of our dataset.

Our work considers a class of improved greedy search algorithms that we call generalized beam search methods.

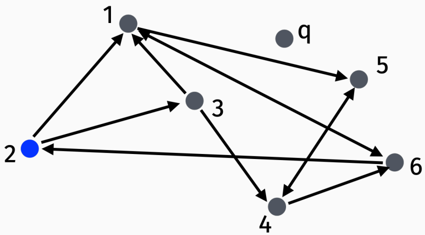
Algorithm 1 Generalized Beam Search

Input: Search graph G over nodes $\{1, \ldots, n\}$, starting node s, distance function d, query q, target number of nearest neighbors k.
Output: Set of k nodes $\mathcal{B} \subset \{1, \ldots, n\}$, where each $x \in \mathcal{B}$ is ideally close to q with respect to d.

Initialize min-priority queues $\mathcal{C}$ and $\mathcal{D}$. ▷ Elements are nodes, priorities are distances to q. $\mathcal{D}$ contains all discovered nodes. $\mathcal{C}$ contains discovered nodes that are not yet expanded.
Insert $(s, d(q, s))$ into $\mathcal{C}$ and $\mathcal{D}$.
while C is not empty **do**
    $(x, d(q, x)) \leftarrow \text{extractMin}(\mathcal{C})$. ▷ Pop min. distance node.
    if x satisfies **[termination condition]** **then**
        break
    For all $y \in \mathcal{N}_G(x)$, if y is not in $\mathcal{D}$, insert $(y, d(q, y))$ into $\mathcal{C}$ and $\mathcal{D}$. ▷ Expand node x.
Obtain $\mathcal{B}$ by running extractMin k times on $\mathcal{D}$, which returns the k elements with the smallest distances from the query, q.

All methods in this class search nodes in the same order but use a different termination rule to decide when to stop.

Every time we "explore" a node, we compute the distance between q and all of the node's neighbors.

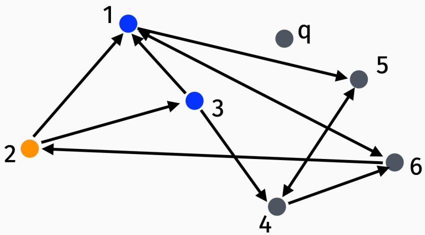


Discovered Nodes = [2]

Unexplored Nodes = [2]

Explored Nodes = []

Every time we "explore" a node, we compute the distance between q and all of the node's neighbors.

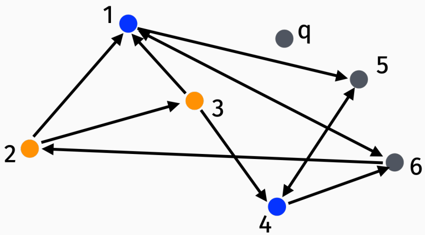


Discovered Nodes = [3, 1, 2]

Unexplored Nodes = [3, 1]

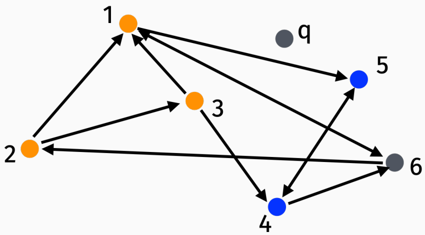
Explored Nodes = [2]

Every time we “explore” a node, we compute the distance between q and all of the node’s neighbors.



Discovered Nodes = [3, 1, 4, 2]
Unexplored Nodes = [1, 4]
Explored Nodes = [3, 2]

Every time we "explore" a node, we compute the distance between q and all of the node's neighbors.

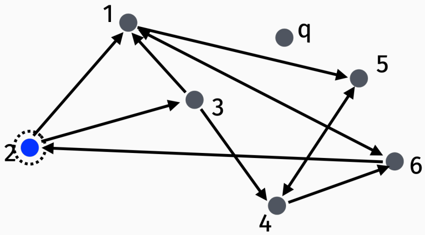


Discovered Nodes = [5, 3, 1, 4, 2]
Unexplored Nodes = [5, 4]
Explored Nodes = [3, 1, 2]

Once the search terminates, return the k best results in the Discovered Nodes list.

Importantly, beam search allows backtracking!

Denote the next node to explore by x.



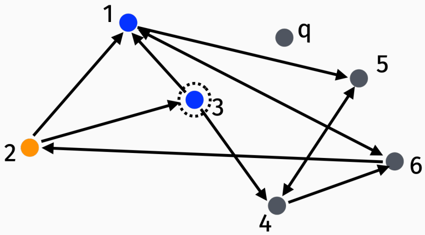
Discovered Nodes = [2]

Unexplored Nodes = [2]

Explored Nodes = []

Greedy stopping condition: Terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with $d(j_i, q) < d(x, q)$.

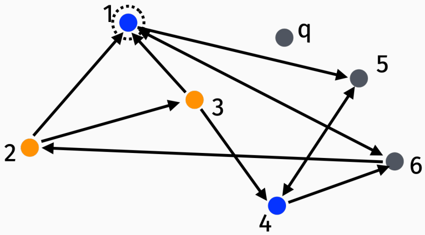
Denote the next node to explore by x.



Discovered Nodes = [3, 1, 2]
Unexplored Nodes = [3, 1]
Explored Nodes = [2]

Greedy stopping condition: Terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with $d(j_i, q) < d(x, q)$.

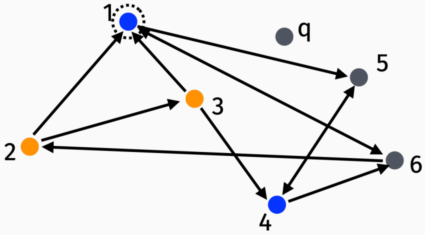
Denote the next node to explore by x.



Discovered Nodes = [3, 1, 4, 2]
Unexplored Nodes = [1, 4]
Explored Nodes = [3, 2]

Greedy stopping condition: Terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with $d(j_i, q) < d(x, q)$.

We will obtain a strictly better result if we allow the search to continue. It is thus natural to relax the greedy stopping rule.

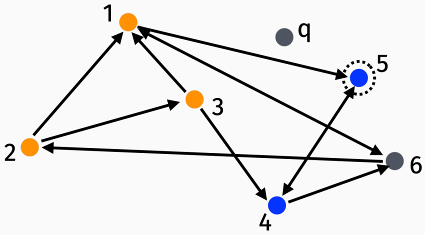


Discovered Nodes = [3, 1, 4, 2]
Unexplored Nodes = [1, 4]
Explored Nodes = [3, 2]

Beam search stopping condition: Terminate if there are $b > k$ Discovered Nodes, $j_1, \ldots, j_b$, with $d(j_i, q) < d(x, q)$.

STANDARD BEAM SEARCH STOPPING CONDITION

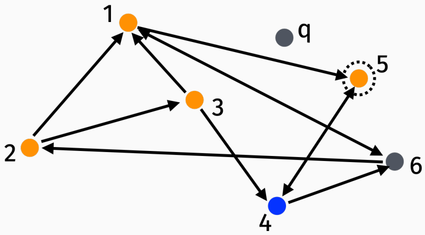
We will obtain a strictly better result if we allow the search to continue. It is thus natural to relax the greedy stopping rule.



Discovered Nodes = [5, 3, 1, 4, 2]
Unexplored Nodes = [5, 4]
Explored Nodes = [3, 1, 2]

Beam search stopping condition: Terminate if there are $b > k$ Discovered Nodes, $j_1, \ldots, j_b$, with $d(j_i, q) < d(x, q)$.

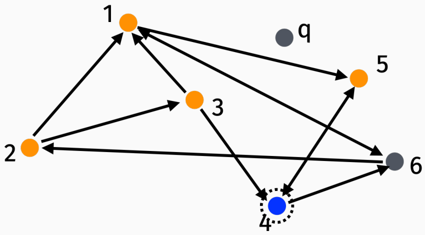
We will obtain a strictly better result if we allow the search to continue. It is thus natural to relax the greedy stopping rule.



Discovered Nodes = [5, 3, 1, 4, 2]
Unexplored Nodes = [4]
Explored Nodes = [5, 3, 1, 2]

Beam search stopping condition: Terminate if there are $b > k$ Discovered Nodes, $j_1, \ldots, j_b$, with $d(j_i, q) < d(x, q)$.

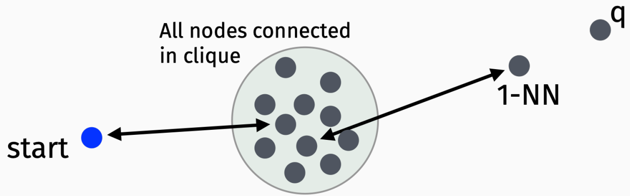
We will obtain a strictly better result if we allow the search to continue. It is thus natural to relax the greedy stopping rule.



Discovered Nodes = [5, 3, 1, 4, 2]
Unexplored Nodes = [4]
Explored Nodes = [5, 3, 1, 2]

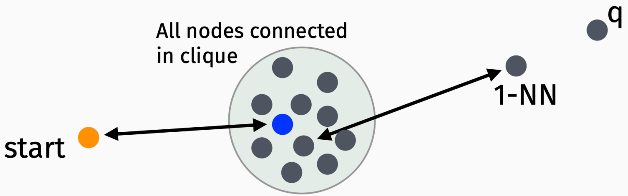
Beam search stopping condition: Terminate if there are $b > k$ Discovered Nodes, $j_1, \ldots, j_b$, with $d(j_i, q) < d(x, q)$.

Unfortunately, even if we set $b = O(n)$, beam search does not yield provably accurate nearest neighbors when run on a navigable graph.



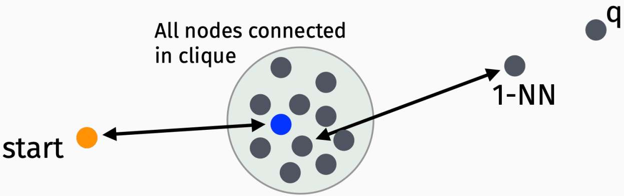
Can have a large cluster of “local minimums” that beam search will never escape.

Unfortunately, even if we set $b = O(n)$, beam search does not yield provably accurate nearest neighbors when run on a navigable graph.



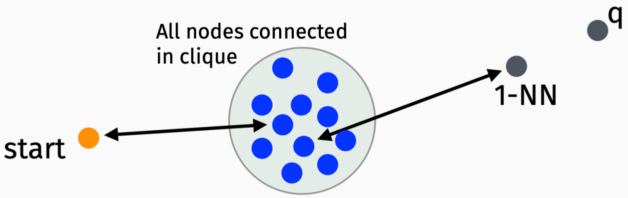
Can have a large cluster of "local minimums" that beam search will never escape.

Unfortunately, even if we set $b = O(n)$, beam search does not yield provably accurate nearest neighbors when run on a navigable graph.



Can have a large cluster of “local minimums” that beam search will never escape.

Our idea: Relax by distance instead of number of closer nodes.

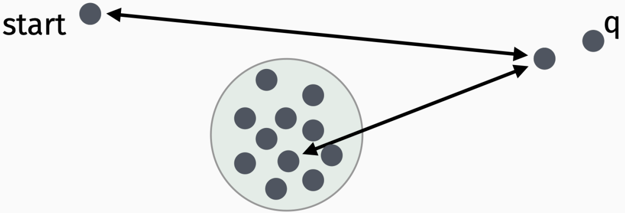


Adaptive Beam Search stopping condition: For $\gamma > 0$, terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with

$$(1+\gamma) \cdot d(j_i, q) < d(x, q).$$

Adaptive Beam Search seems to more naturally "adapt" to query difficulty. Spends more time when there are many similar local minima, less time when there are not.

For "easy" queries, Adaptive Beam Search can terminate faster than standard beam search.

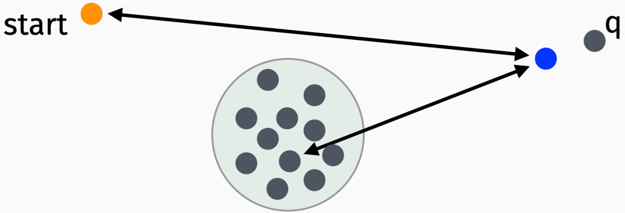


Adaptive Beam Search stopping condition: For $\gamma > 0$, terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with

$$(1+\gamma) \cdot d(j_i, q) < d(x, q).$$

In fact, the rule has been previously used as an "early termination condition" for beam search [ParlayANN, Dobson Manohar et al., 2024]!

For "easy" queries, Adaptive Beam Search can terminate faster than standard beam search.

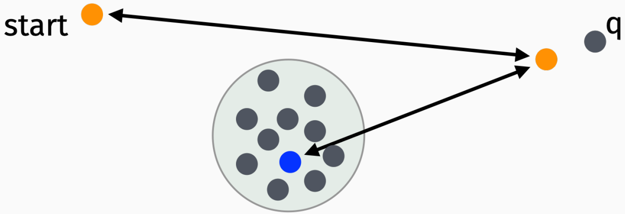


Adaptive Beam Search stopping condition: For $\gamma > 0$, terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with

$$(1+\gamma) \cdot d(j_i, q) < d(x, q).$$

In fact, the rule has been previously used as an "early termination condition" for beam search [ParlayANN, Dobson Manohar et al., 2024]!

For "easy" queries, Adaptive Beam Search can terminate faster than standard beam search.



Adaptive Beam Search stopping condition: For $\gamma > 0$, terminate if there are k Discovered Nodes, $j_1, \ldots, j_k$, with

$$(1+\gamma) \cdot d(j_i, q) < d(x, q).$$

In fact, the rule has been previously used as an "early termination condition" for beam search [ParlayANN, Dobson Manohar et al., 2024]!

Theorem (Al-Jazzazi, Diwan, Gou, Musco, Musco, Suel, 2025)

Let $k = 1$. For any metric d and for any $\gamma \leq 2$, Adaptive Beam Search run on a navigable graph returns x' satisfying:

$$d(x', q) \leq \frac{2}{\gamma} \cdot \min_{x \in \{1,\ldots,n\}} d(x, q).$$

- When $\gamma = 2$, we obtain the exact nearest neighbor. Smaller values of γ yield worse approximations.
- The same bound holds for $k > 1$: no point that is not returned by the method can be more than $\frac{\gamma}{2}\times$ closer than any of the k points returned.

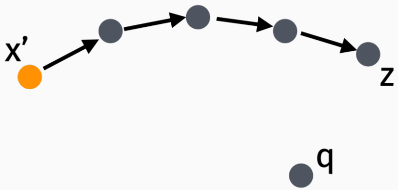
Need to show that any <u>undiscovered</u> z has $d(z, q) \geq \frac{\gamma}{2} \cdot d(x', q)$.

Discovered Nodes = [x’, ...]

Unexplored Nodes = [....]

Explored Nodes = [x’, ...]

Need to show that any <u>undiscovered</u> z has $d(z, q) \geq \frac{\gamma}{2} \cdot d(x', q)$.



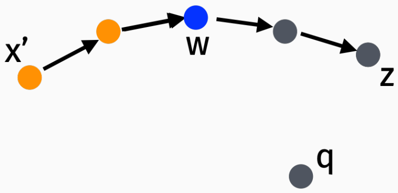
Discovered Nodes = [x', ...]

Unexplored Nodes = [....]

Explored Nodes = [x', ...]

Existence of monotonically improving path from x' to z follows from navigability.

Need to show that any undiscovered z has $d(z, q) \geq \frac{\gamma}{2} \cdot d(x', q)$.

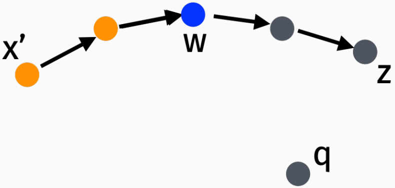


Discovered Nodes = [x', ...]

Unexplored Nodes = [....]

Explored Nodes = [x', ...]

Need to show that any undiscovered z has $d(z, q) \geq \frac{\gamma}{2} \cdot d(x', q)$.



Discovered Nodes = [x', ...]

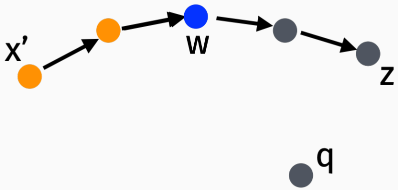
Unexplored Nodes = [....]

Explored Nodes = [x', ...]

Since w is unexplored, it must be that $d(w, q) \geq (1 + \gamma)d(x', q)$.

PROOF BY CONTRADICTION

Suppose by way of contradiction that $d(z,q) < \frac{\gamma}{2} \cdot d(x', q)$.



Discovered Nodes = [x’, ...]

Unexplored Nodes = [....]

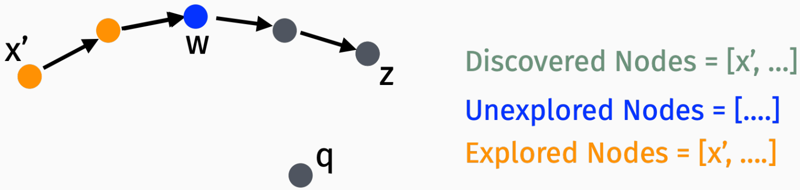
Explored Nodes = [x’, ...]

Since w is unexplored, it must be that $d(w,q) \geq (1+\gamma)d(x',q)$.

At the same time we have via triangle inequality:

$$\begin{aligned} d(w,q) &\leq d(w,z) + d(z,q) \leq d(x',z) + d(z,q) \\ &\leq d(x',q) + d(z,q) + d(z,q) \\ &< (1+\gamma) \cdot d(x',q). \end{aligned}$$

Suppose by way of contradiction that $d(z, q) < \frac{\gamma}{2} \cdot d(x', q)$.



Since w is unexplored, it must be that $d(w, q) \geq (1 + \gamma)d(x', q)$.

At the same time we have via triangle inequality:

$$\begin{aligned} d(w, q) &\leq d(w, z) + d(z, q) \leq d(x', z) + d(z, q) \\ &\leq d(x', q) + d(z, q) + d(z, q) \\ &< (1 + \gamma) \cdot d(x', q). \end{aligned}$$

We have thus reach a contridiction! So $d(z, q) \geq \frac{\gamma}{2} \cdot d(x', q)$.

Theorem (Al-Jazzazi, Diwan, Gou, Musco, Musco, Suel, 2025)

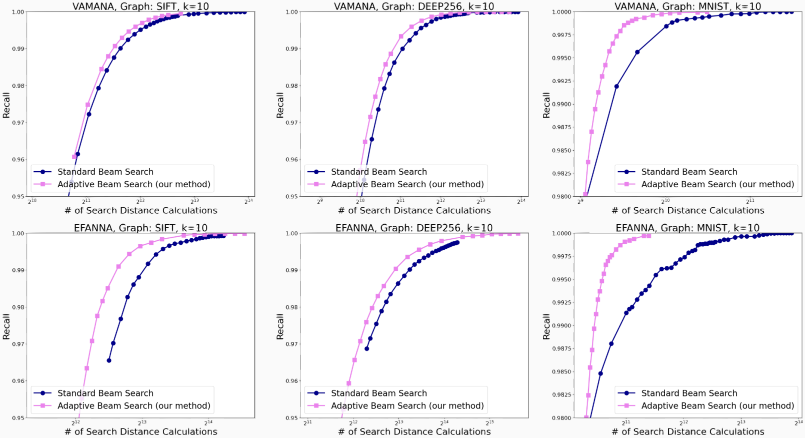
Let $k = 1$. For any metric d and for any $\gamma \leq 2$, Adaptive Beam Search run on a navigable graph returns x' satisfying:

$$d(x', q) \leq \frac{2}{\gamma} \cdot \min_{x \in \{x, \ldots, n\}} d(i, q).$$

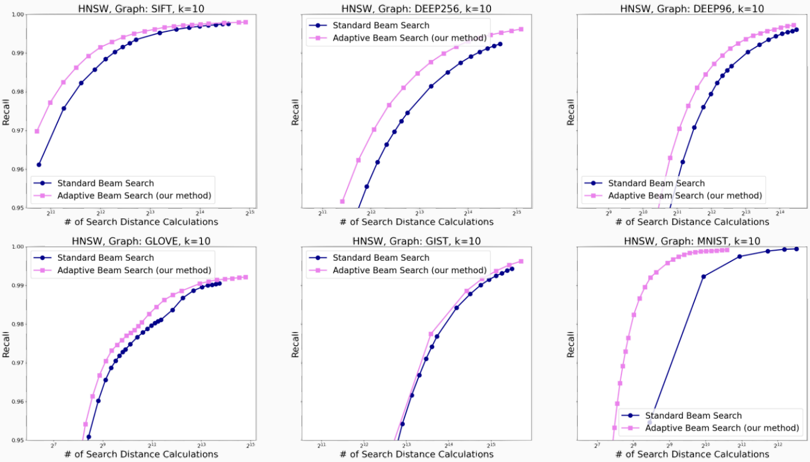
The theoretical benefit of Adaptive Beam Search seems to translate to practice.

- Experimented on 6 datasets, 5 graph constructions, and a variety of k values.
- Ran Standard Beam Search and Adaptive Beam Search, varying b and γ to obtain curves trading off between recall and # of distance computations.

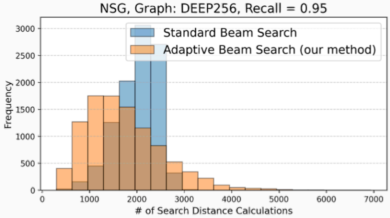
Adaptive Beam Search never performs worse than Standard Beam Search. Typically, 10% – 50% better.

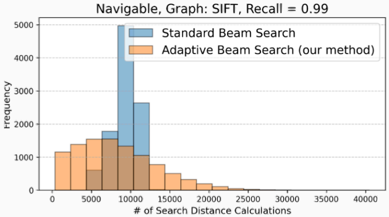


Adaptive Beam Search never performs worse than Standard Beam Search. Typically, 10% – 50% better.



As expected, improvement seems to come from better "adapting" to query hardness.





For Adaptive Beam Search, we see greater variance in the number of distance computations used per query.

IMPORTANT NOTE ABOUT EXPERIMENTS

Despite their names, many of the graphs we run on (NSG, HNSW, etc.) are not actually navigable!

Microsoft's DiskANN can be configured to return a navigable graph, but this is not how it is typically used.

Still reasonable to expect theoretical results to be meaningful. But we wanted to also test on actually navigable graphs.

For the paper we use a fast heuristic for constructing sparse navigable graphs: basically start with the $O(\sqrt{n})$ degree construction of Diwan at al. then "prune" edges as in DiskANN.

Result is provably navigable graphs with degree ≈ 50 for datasets we tested.

How quickly can we construct the sparsest navigable graph for a given dataset?

Theorem (Conway, Dhulipala, Farach-Colton, Johnson, Landrum, Musco, Shechter, Suel, Wen, 2025)

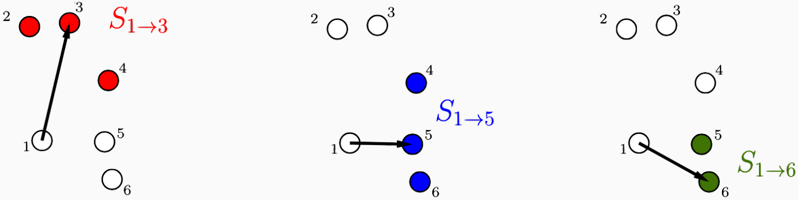
For any dataset and any distance function, we can construct a navigable graph with at most $O(\log n)$ times as many edges as the sparsest navigable graph in $\tilde{O}(n^2)$ time.

- Also give results for α-shortcut reachable graphs. Similar results obtained by [Khanna, Padaki, Waingarten, 2025].
- $O(n^2)$ time is optimal even for points in $O(\log n)$ dimensional Euclidean space assuming Strong Exponential Time Hypothesis.
- $O(\log n)$ approximation factor is optimal assuming $P \neq NP$

Navigable graph construction is n different minimum set cover problems!

Definition (Navigable Graph)

A graph G is navigable for a point set $1, \ldots, n$ and distance function $d(\cdot, \cdot)$ if, for all nodes i, for all $j \neq i$, there is some $k \in \mathcal{N}(i)$ (i's out neighborhood) satisfying $d(j, k) < d(j, i)$.



One problem for each node i. One set for all possible out neighbors. Elements to cover are all $j \neq i$.

Baseline: Explicitly write down each set cover problem, which takes $n \times O(n^2)$ time. Standard greedy algorithm obtains a $O(\log n)$ approximation in $n \times O(n^2) = O(n^3)$ time.

Key Observation: In $O(n^2 \log n)$ time, we can implement and oracle to access information about the set cover instances, without every writing them down explicitly.

Distance Based Permutation Matrix

Node 1	1	10	2	3	5	9	6	8	4	7
Node 2	2	4	6	1	10	5	3	9	7	8
					⋮					
Node n	10	9	1	5	7	2	4	8	3	6

NAVIGABILITY AS SET COVER

Concretely, after $O(n^2 \log n)$ time preprocessing, can implement **Membership** and **SetOf** queries in $O(1)$ time.

Distance Based Permutation Matrix

Node 1	1	10	2	3	5	9	6	8	4	7
Node 2	2	4	6	1	10	5	3	9	7	8
⋮										
Node n	10	9	1	5	7	2	4	8	3	6

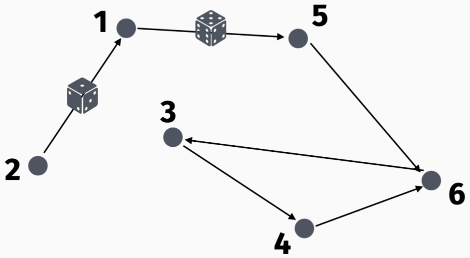
Sets that contain 2 in instance 10

- **Membership:** Is j in set k in instance i? I.e., is $d(j,k) < d(j,i)$?
- **SetOf:** What is the k^{th} set containing j in instance i?

Applying off-the-shelf "sublinear time" set cover algorithms, we can obtain runtime $\tilde{O}(n \cdot OPT) \leq \tilde{O}(n^{2.5})$.

[Indyk, Mahabadi, Rubinfeld, Vakilian, Yodpinyanee, SODA 2018].

Obtaining $\tilde{O}(n^2)$ time requires a few more tricks. Basically, we reduce the size of our set cover instances via additional preprocessing.



These techniques do not apply to general set cover problems. They reduce the size of the average instance in our n instances, not the maximum size. See paper for more details!

QUESTIONS?