Graph-based Indices for ANNS

Harsha Simhadri (Microsoft)

Why Graph Indices?

Pros:

- Query efficiency
- Versatile

Cons:

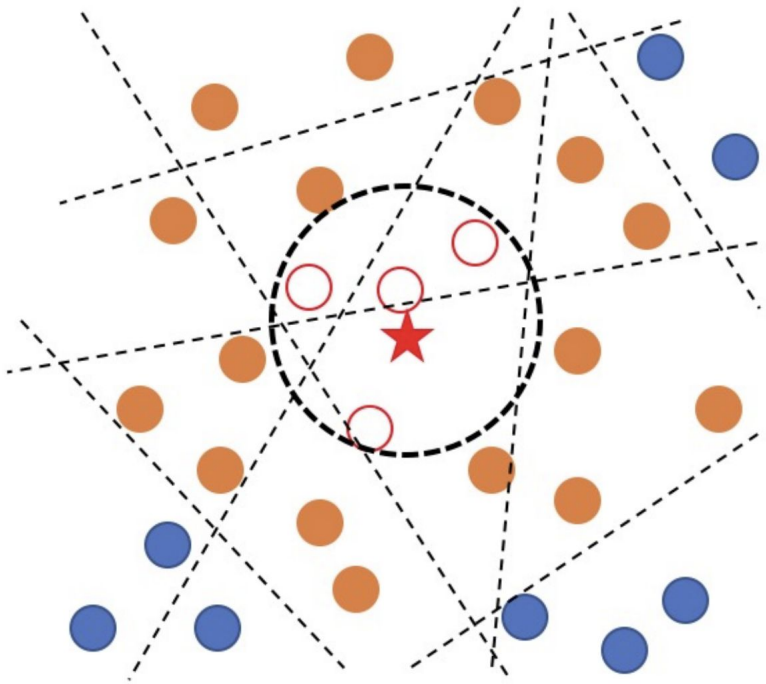
- Slow index construction
- Little theoretical understanding
- Harder to integrate into databases

Outline

- Part 1 [Efficiency]
  - Graph indices: search, construction and empirical analysis
    - (H)NSW,
    - NSG, Vamana/DiskANN
    - HCNNG
  - (Limited) analysis of convergence properties

- Part 2 [Versatility]
  - Disk-based indices
  - Streaming indices
  - Filtered search
  - Out of Distribution queries

Locality Sensitive Hashing (LSH)



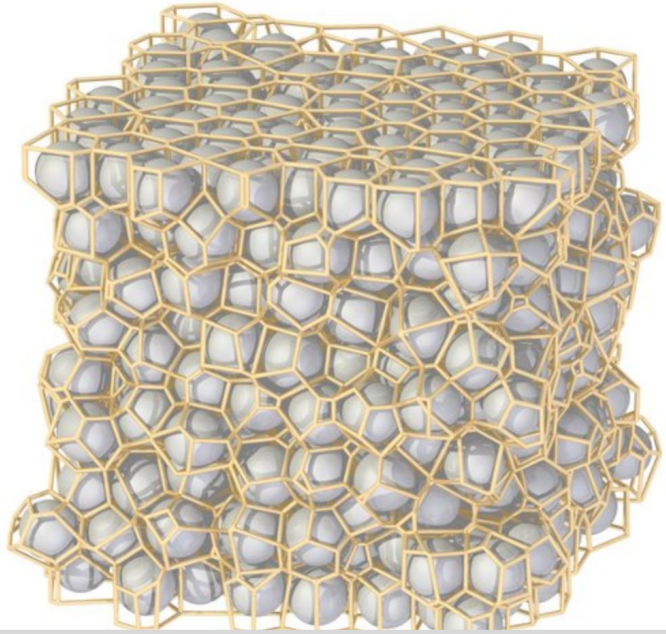
- Hash space into buckets using many random hyperplanes
  - When query arrives, fetch points from nearby buckets (all orange points) and return top K vectors

- First and only theoretically sound algorithm for ANNS
  - $O(n^{1+1/c})$-space and $O(d\ n^{1/c})$-query time for c-approximate NNS

Clustering based algorithms [Matthijs' talk]

Popularized by Facebook AI Similarity Search [FAISS IVF]

- Index build: Cluster points into $k=n^{1/2}$ clusters
  - (k=32000 for n=1B)
- Query time
  - Find ***w*** closest clusters to query
  - Retrieve all points from these ***w*** clusters
  - Output top-k based on approximate distances using compressed vectors
- Compress each vector into 32 bytes so **bil**
  into 32 GB RAM using product quantizati

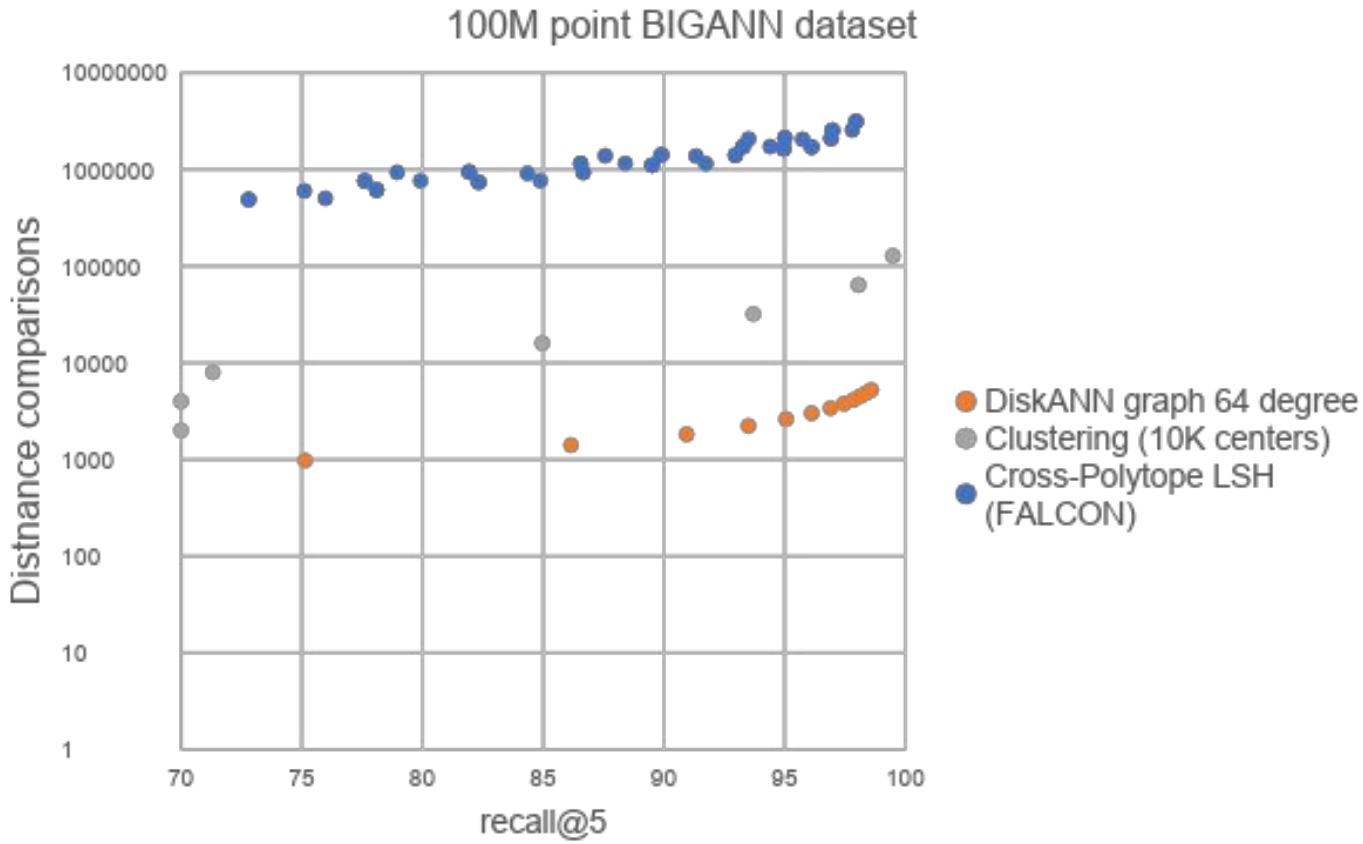


Main drawback: compressed vector distances are lossy and susceptible to noise

✓ High density of points/machine

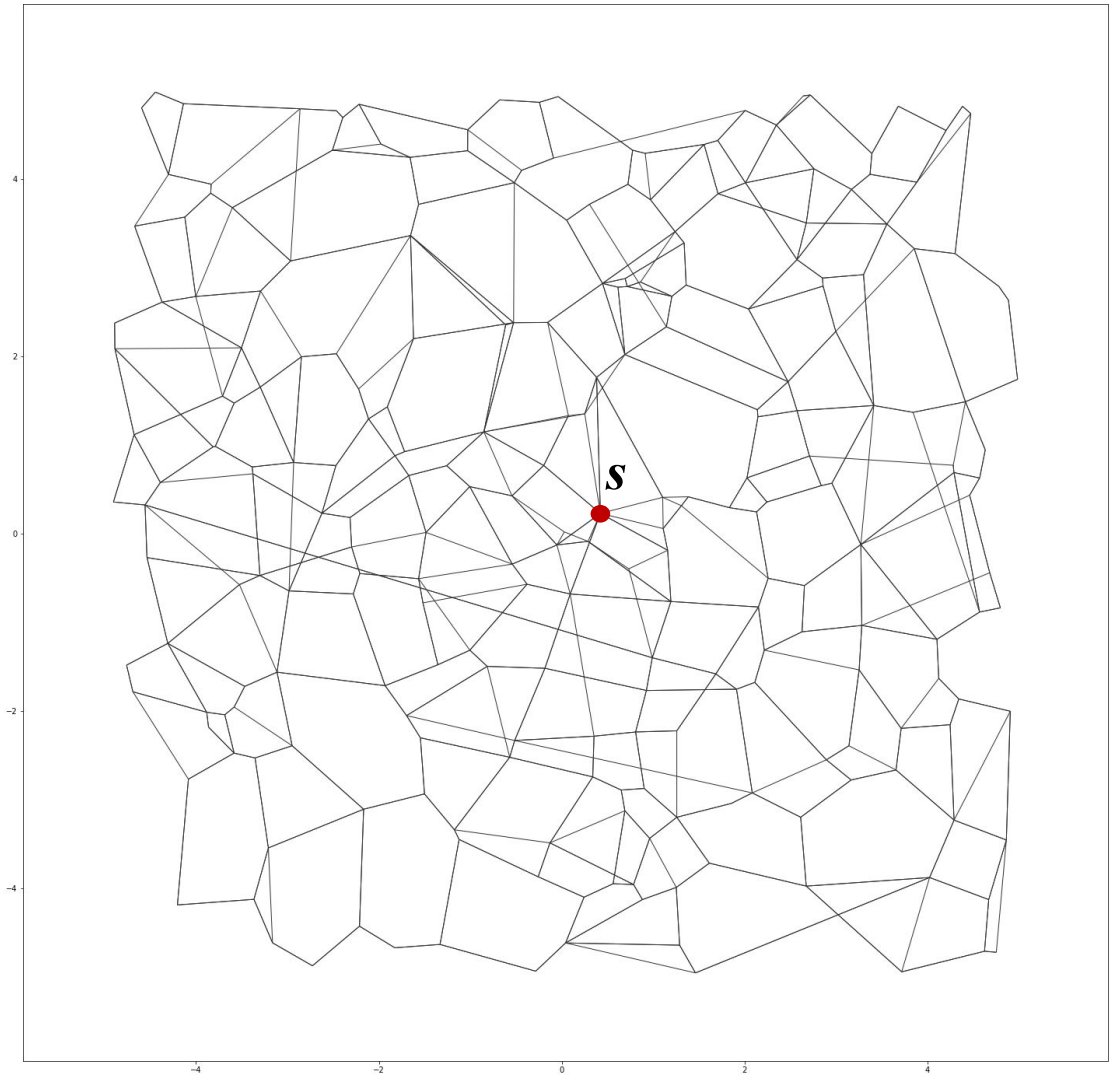
✗ Low recalls

LSH, Clustering, Graph at 100 million scale



Cross-polytope LSH from FALCONN library [Razenshteyn'16]
number of tables = {10 15}, probe width = {25 50 100 200}, dimension of last polytope = {4 8 16 32}, cross polytope number = 4, number of rotations = 3

Graph-based ANN indices: Data Structure

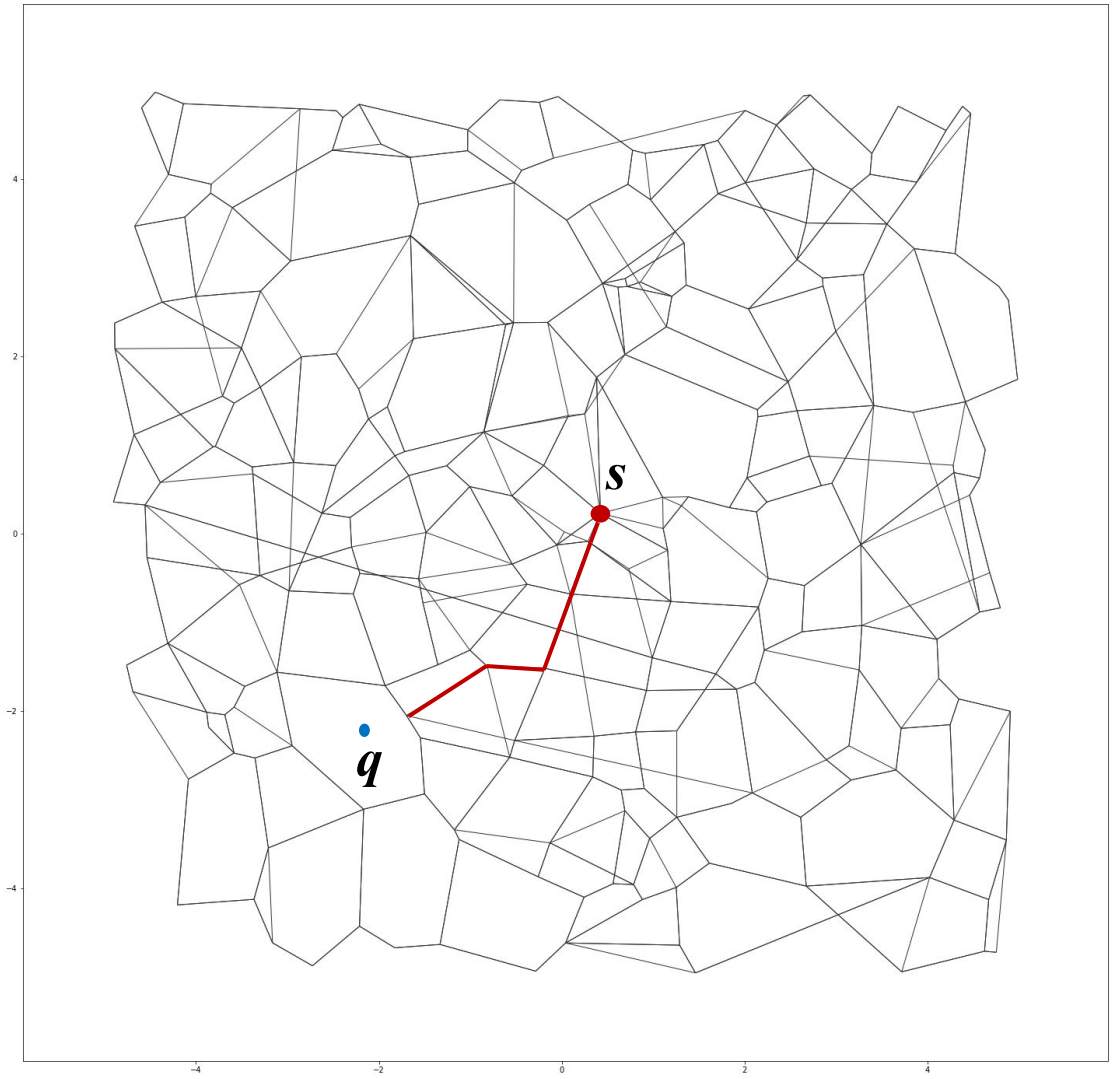


Index Data Structure:

- One vertex per vector/embedding.
- Directed edges between vertices
  - Typically O(log n) degree.
  - Store out-neighbors
- A designated starting point, s
- Size: n*(vector_size + 4Bytes x degree)

 Example in two dimensions with randomly distributed data.

Graph-based ANN indices: Query Path (simplified)



Greedy search (simplified) for query *q*:
Start at designated start ***s***, and iterate:

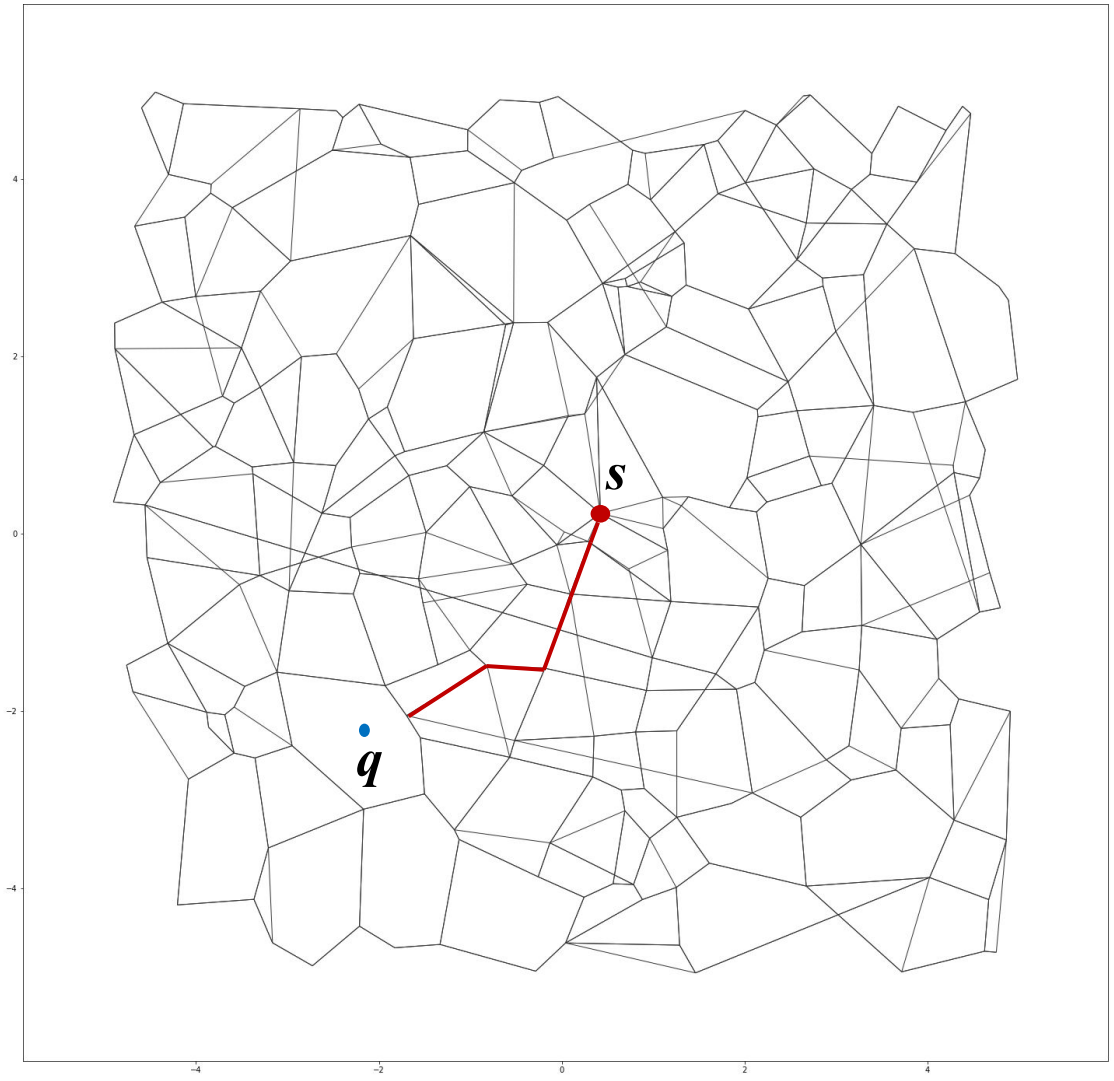
1. compute dist. from ***q*** to neighbors
2. hop to node closest to ***q***,

as long as distance improves

Pitfalls: Local minima. No backtracking.

Let us add backtracking

Graph-based ANN indices: Beam Search with Priority Queue



See Black Board

- ✓ High recall and low latency
- ✗ Data and graph in memory
- Only ~100M points/machine

Graph Construction

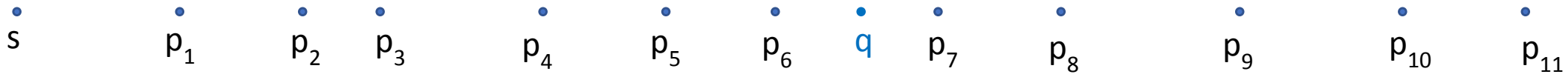
How to build sparse graphs where the greedy search algorithm

- A) converges to good nearest neighbor candidates
- B) In as few “hops” as possible

Examples in 1 dimension

One-dimensional dataset: points on a line

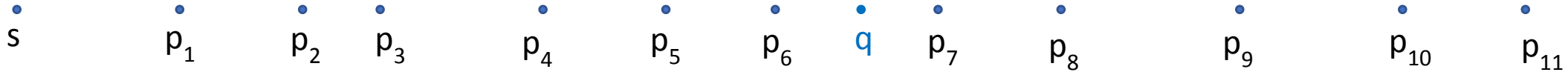
Bad Graph: Not even navigable



Examples in 1 dimension

One-dimensional dataset: points on a line

Navigable Graph, but needs many hops to convergence



Examples in 1 dimension

One-dimensional dataset: points on a line

Navigable Graph with few hops

Main Algorithmic Challenge:

Sparse + Navigable + Low-diameter
Graphs for High-Dimensional Vectors

Navigable Small World (NSW) graphs

[Malkov, Ponomarenko, Logvinov, Krylov, Information systems, vol 45(61-68)]

G.insert(p) {

N(p) ?G. search(p)

Connect p with N(p)

}

- Randomize order of insertion
- Start search anywhere
- No prune of insertions

- Empirical claims:
  - Degree ~ max (dimension, log n) is sufficient
  - #hops to convergence proportional to log n (with some function of d as a constant multiplicative factor)

Navigable Small World (NSW) graphs

[Malkov, Ponomarenko, Logvinov, Krylov, Information systems, vol 45(61-68)]

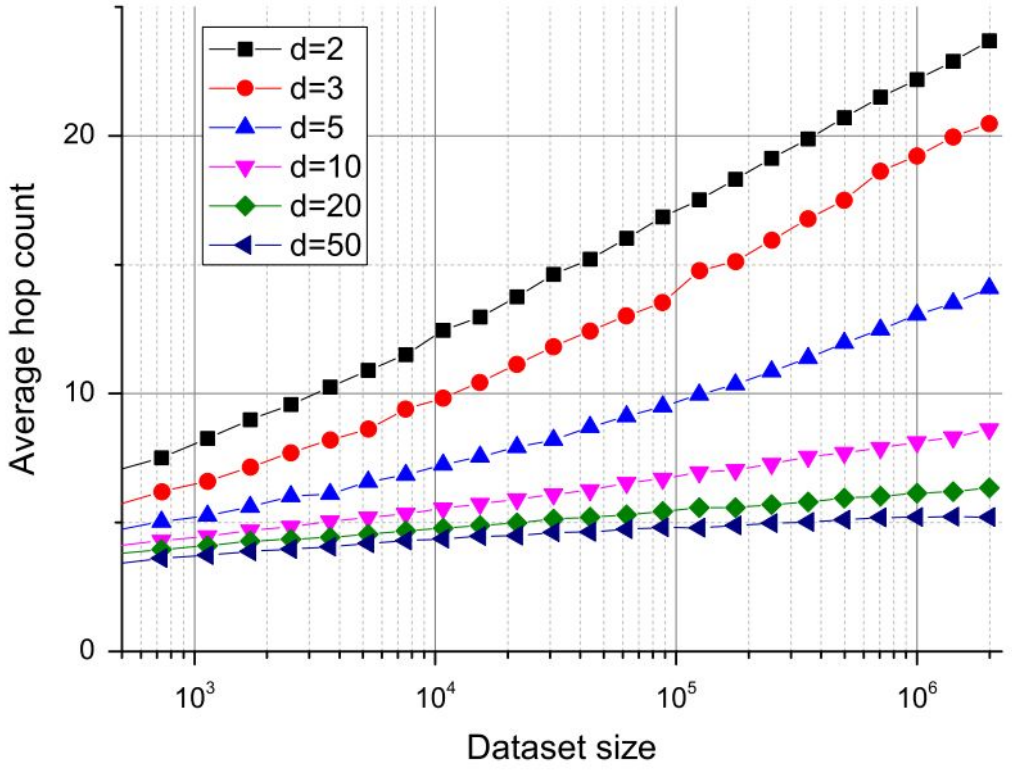


Fig. 2. The average hop count induced by a greedy search algorithm for different dimensionality Euclidean data (k=10, w=20). The navigable small world properties are evident from the logarithmic scaling.

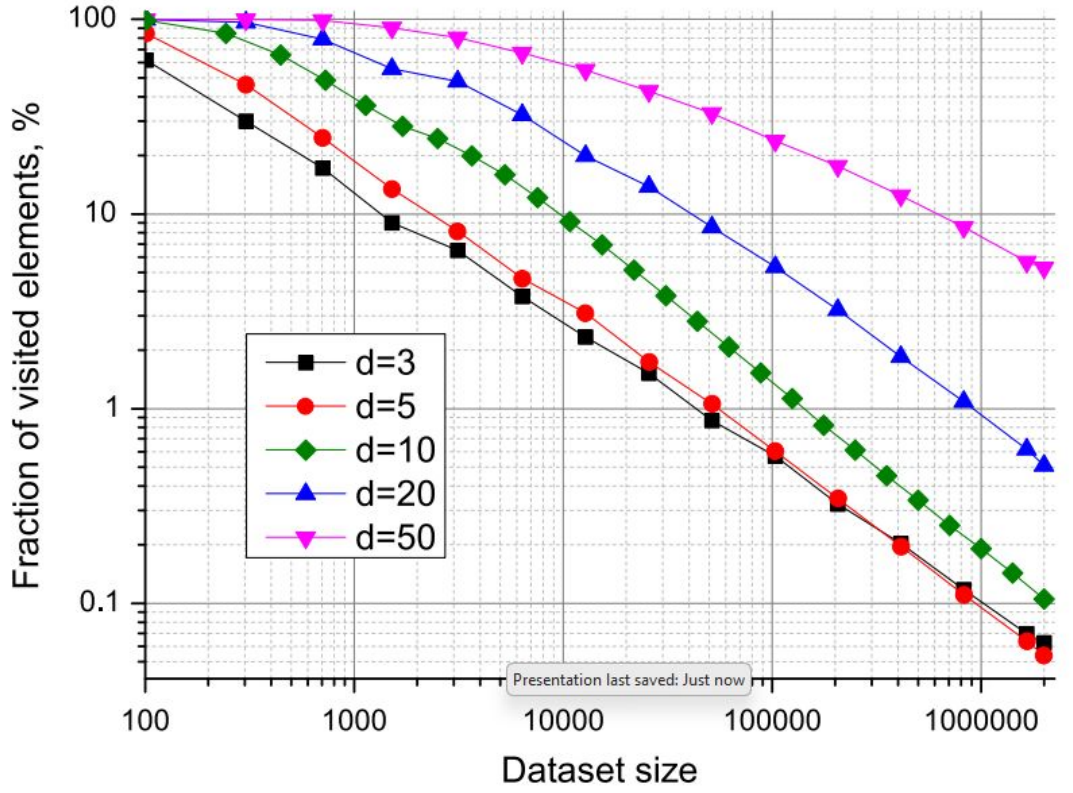


Fig. 3. Average fraction of visited elements within a single 10-NN-search with 0.999 recall versus the size of the dataset for different dimensionality.

Navigable Small World (NSW) graphs

[Malkov, Ponomarenko, Logvinov, Krylov, Information systems, vol 45(61-68)]

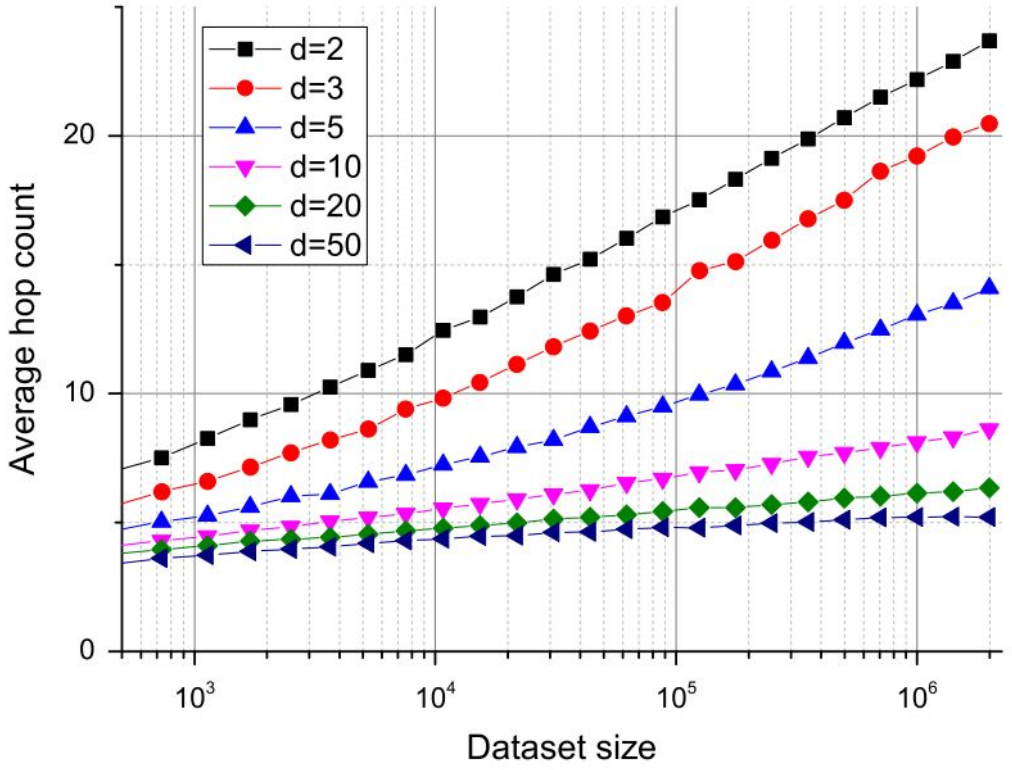


Fig. 2. The average hop count induced by a greedy search algorithm for different dimensionality Euclidean data (k=10, w=20). The navigable small world properties are evident from the logarithmic scaling.

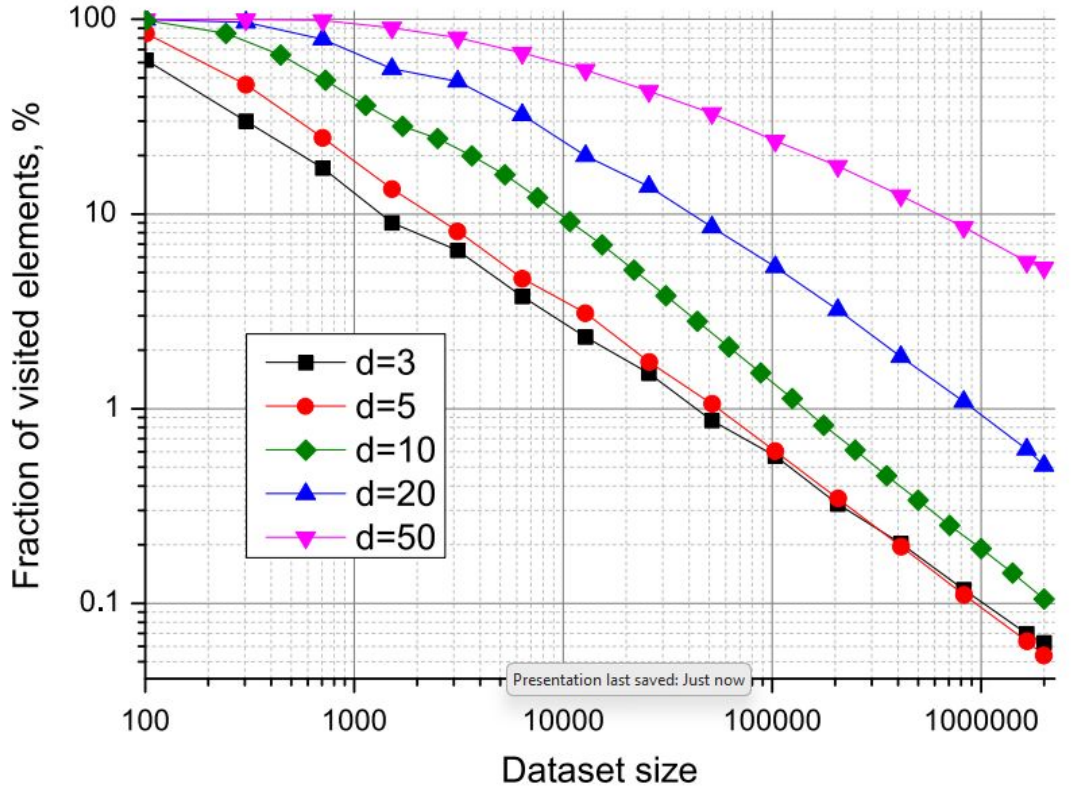


Fig. 3. Average fraction of visited elements within a single 10-NN-search with 0.999 recall versus the size of the dataset for different dimensionality.

Hierarchnical NSW [Malkov, Yashunin, T.PAMI'18]

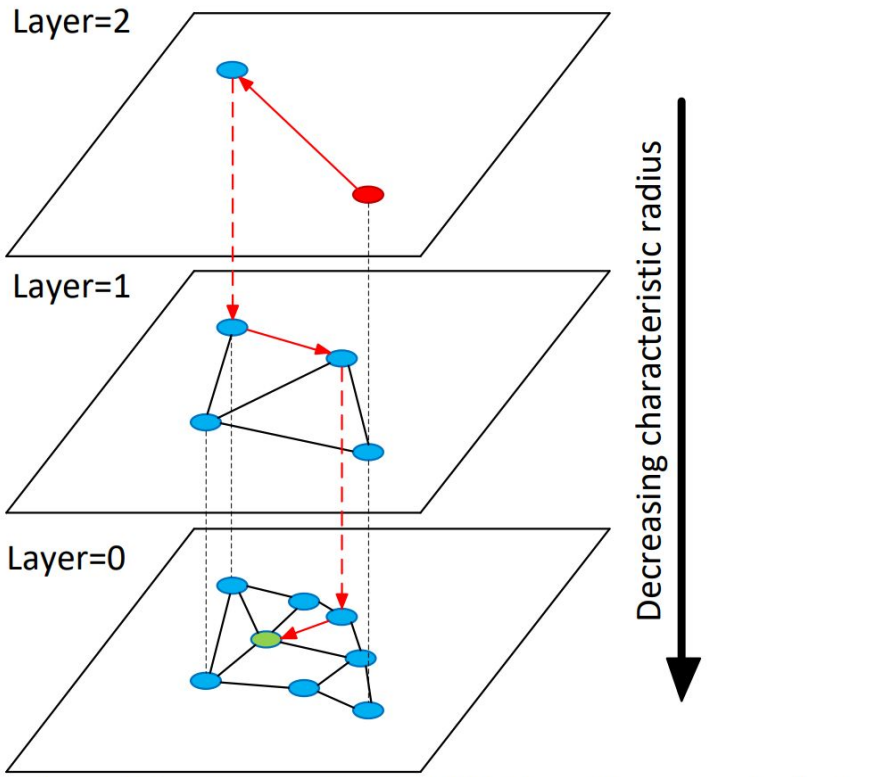
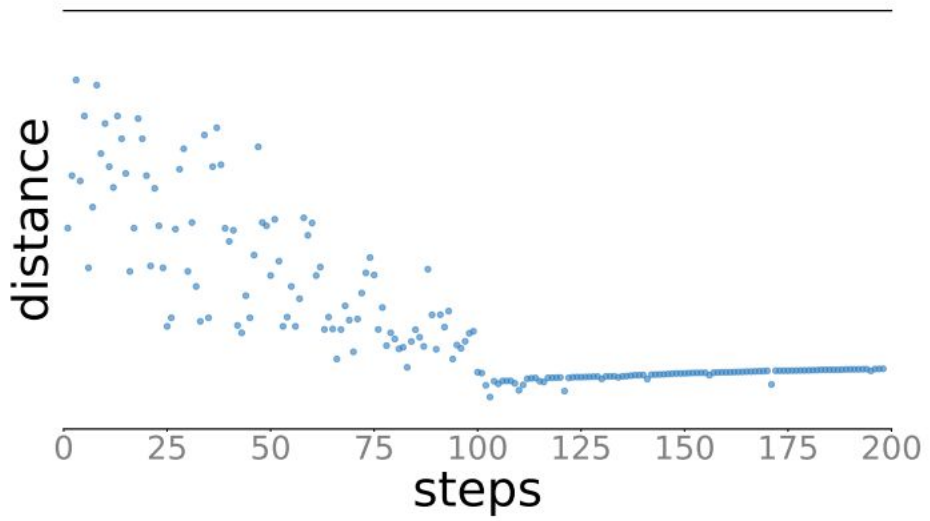


Fig. 1. Illustration of the Hierarchical NSW idea. The search starts from an element from the top layer (shown red). Red arrows show direction of the greedy algorithm from the entry point to the query (shown green).

HNSW: Construction and Search

Comparison with NSW

- No shuffle: Randomization via sampling
- Designated start point
- Prune candidates for selecting neighbors
- Degree bound



HNSW: Impact of construction parameters

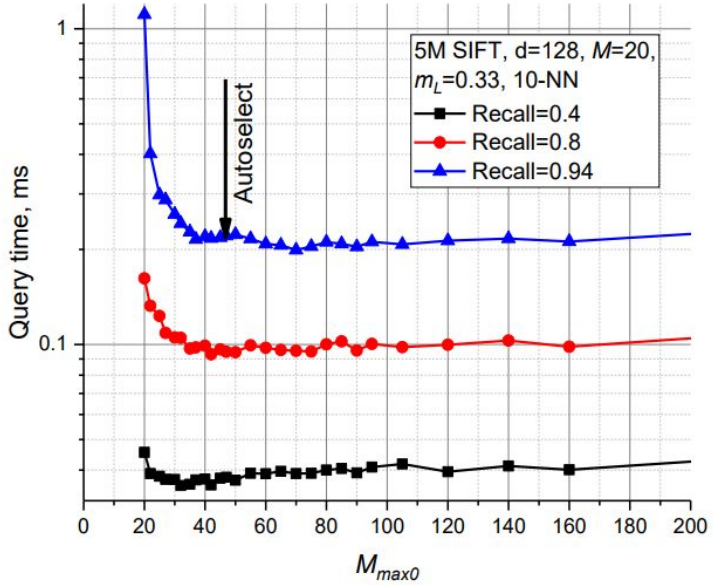


Fig. 6. Plots for query time vs M_{max0} parameter for 5M SIFT learn dataset. The autoselected value $2 \cdot M$ for M_{max0} is shown by an arrow.

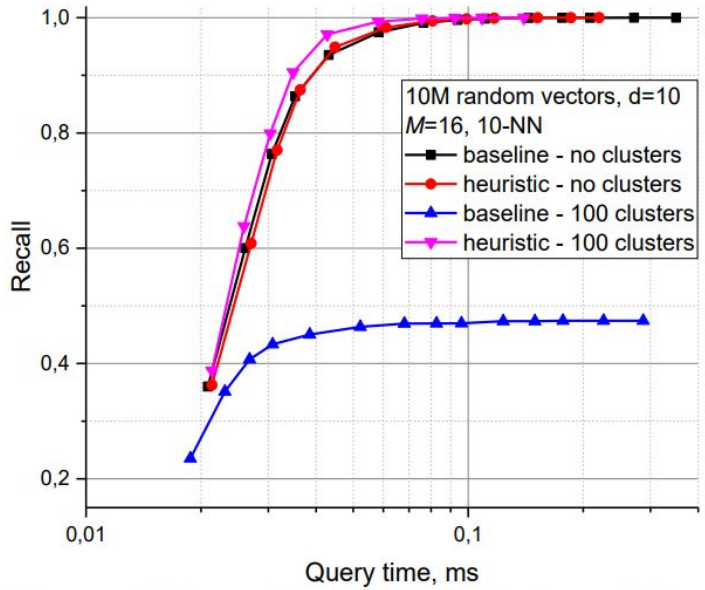


Fig. 7. Effect of the method of neighbor selections (baseline corresponds to alg. 3, heuristic to alg. 4) on clustered (100 random isolated clusters) and non-clustered d=10 random vector data.

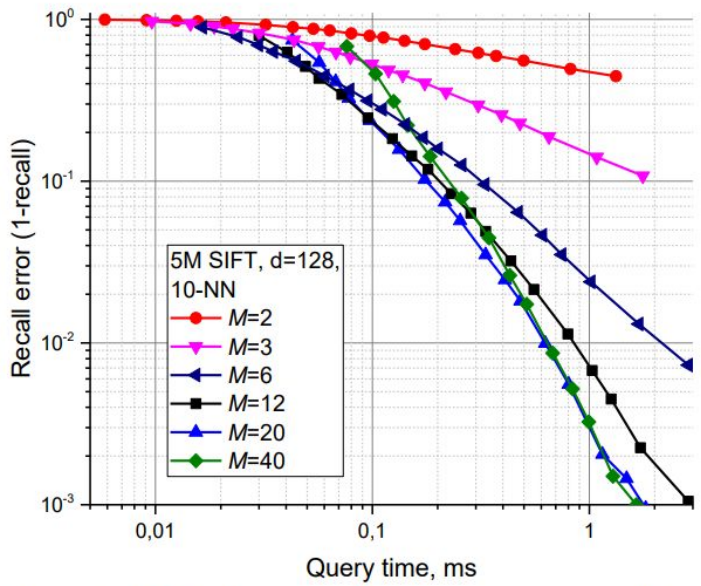


Fig. 8. Plots for recall error vs query time for different parameters of M for Hierarchical NSW on 5M SIFT learn dataset.

HNSW: Comparison with NSW

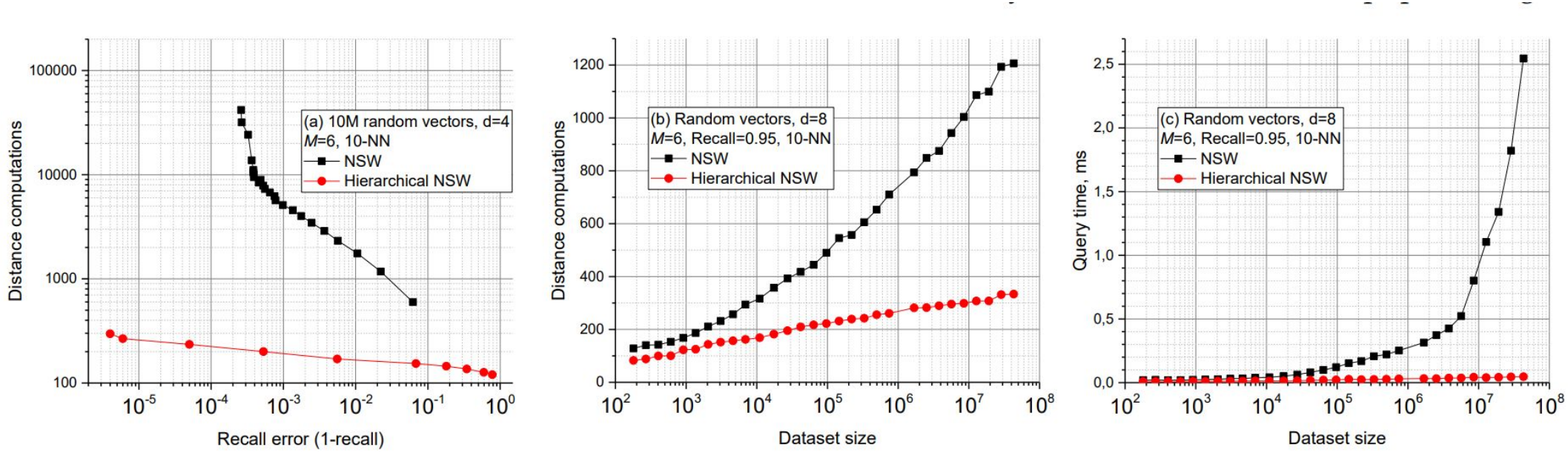


Fig. 12. Comparison between NSW and Hierarchical NSW: (a) distance calculation number vs accuracy tradeoff for a 10 million 4-dimensional random vectors dataset; (b-c) performance scaling in terms of number of distance calculations (b) and raw query(c) time on a 8-dimensional random vectors dataset.

HNSW: Comparison with other algorithms

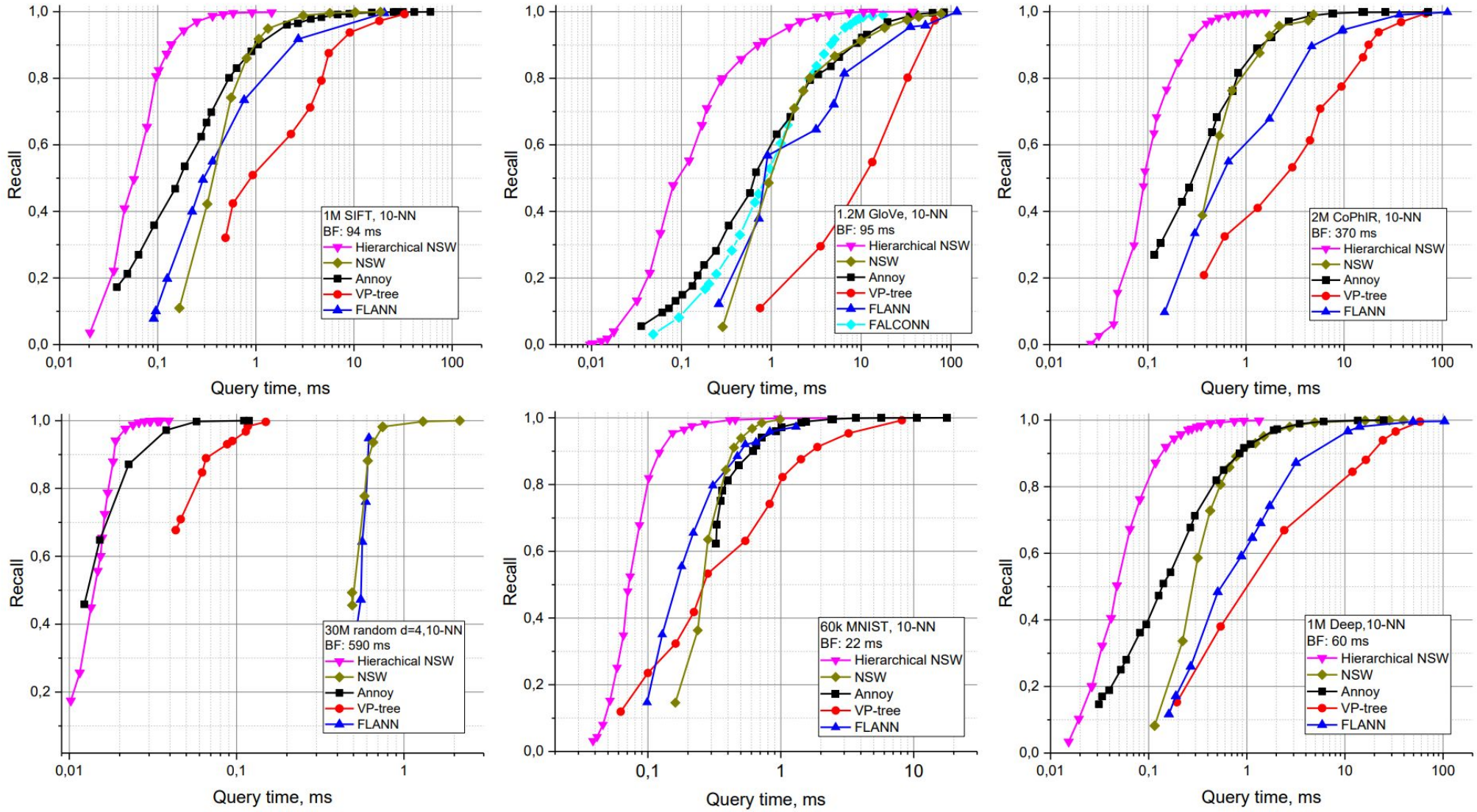


Fig. 13. Results of the comparison of Hierarchical NSW with open source implementations of K-ANNS algorithms on five datasets for 10-NN searches. The time of a brute-force search is denoted as the BF.

Is a hierarchical graph necessary?

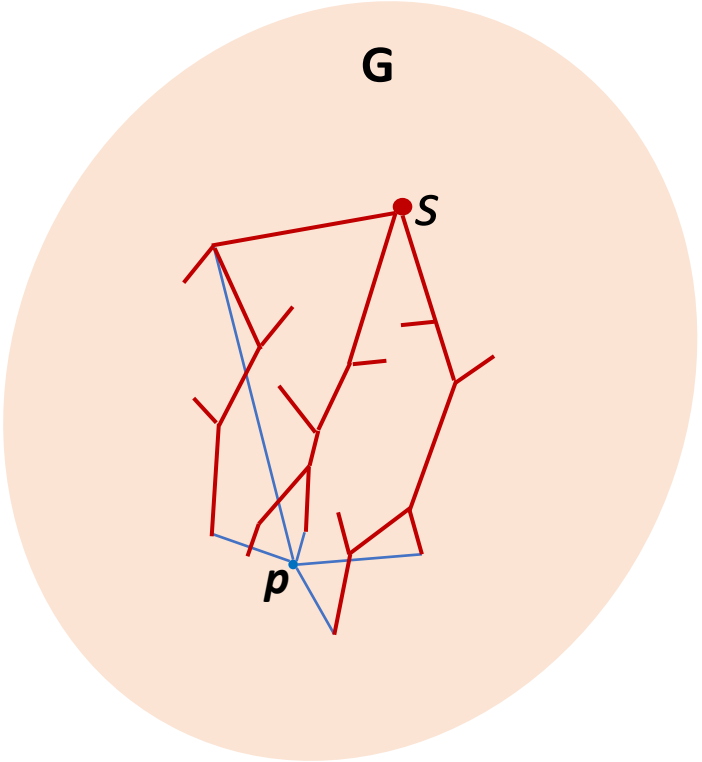
- NSG: Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph [Fu, Xiang, Wang, Cai, Proc. VLDB vol. 12]

- DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node [Subramanya, Kadekodi, Devvrit, Krishnaswamy, Simhadri, NeurIPS'19]

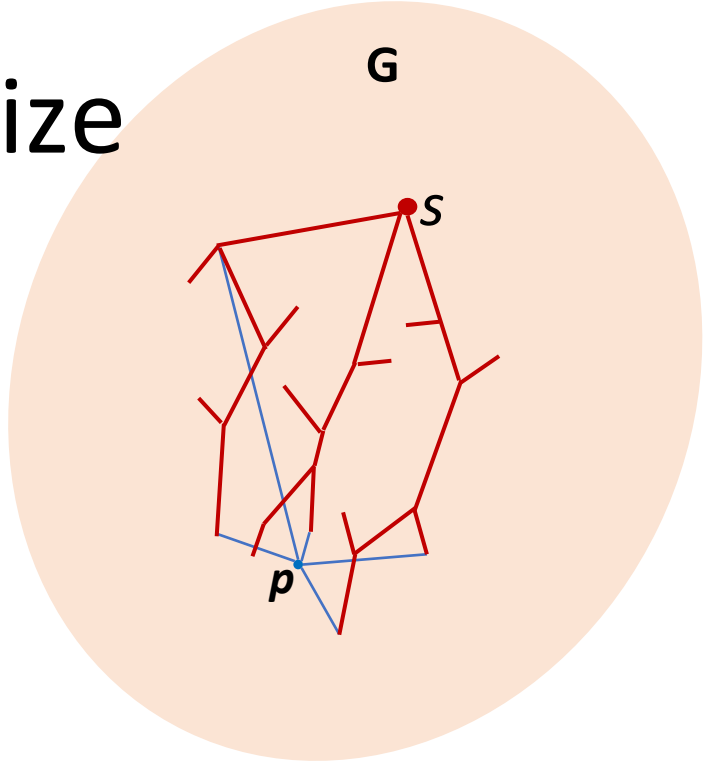
Index build, point by point

G.insert(p, R, α) // R is degree bound, α>=1 constant

- V ← vertices visited by G.search(p) // |V|~5-10K
- V' ← **Prune(p, V, R, α)**
- For all v ∈ V'
  - add edges (p,v) and (v,p) to G
  - if $|N_{out}(v)| > R$
    $N_{out}(v)$ ← **Prune(v, $N_{out}(v)$, R, α)**

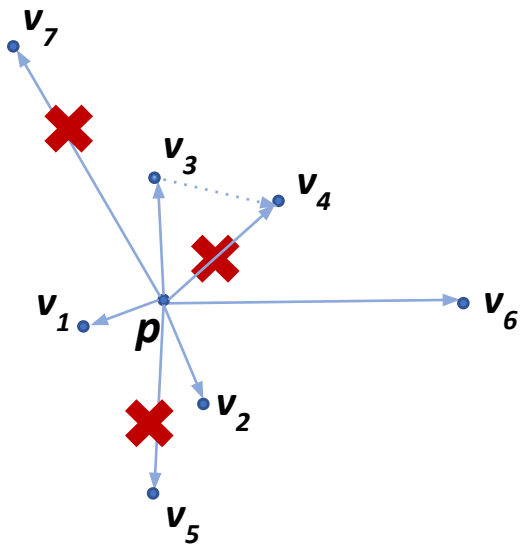


Index build: prune candidates to size



G.insert(p, R, α) // R is degree bound, α>=1 constant

- V ← vertices visited by G.search(p) // |V|~5-10K
- V' ← **Prune(p, V, R, α)**
- For all v ∈ V'
  - add edges (p,v) and (v,p) to G
  - if $|N_{out}(v)| > R$
    $N_{out}(v)$ ← **Prune(v, $N_{out}(v)$, R, α)**

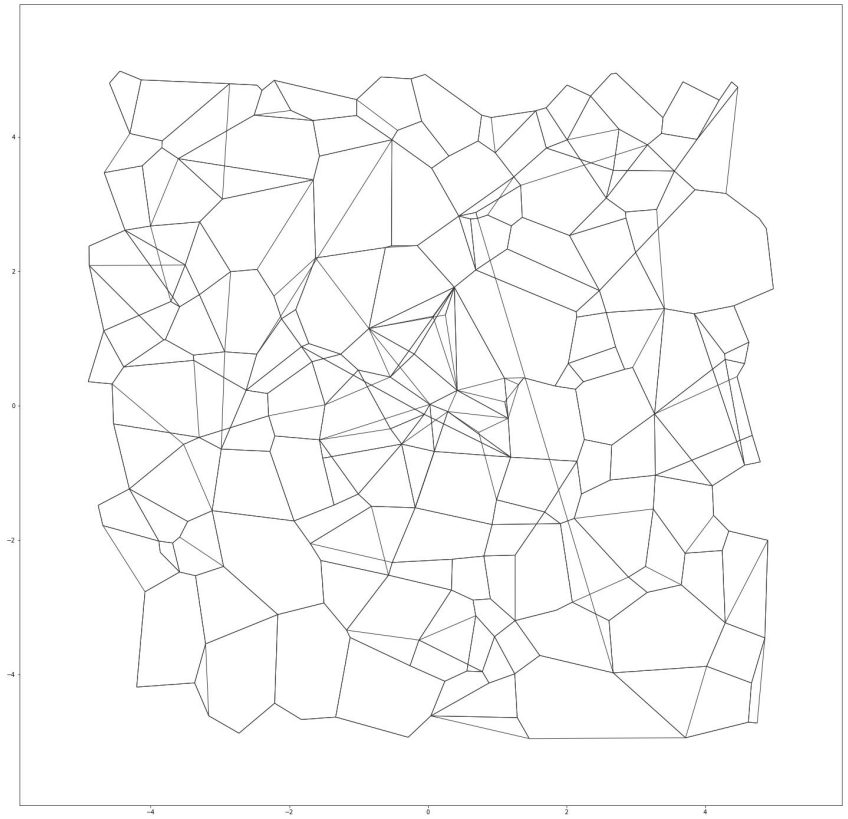


Prune(p, V, R, α) returns V' // |V'|<R is degree, α typically 1.2

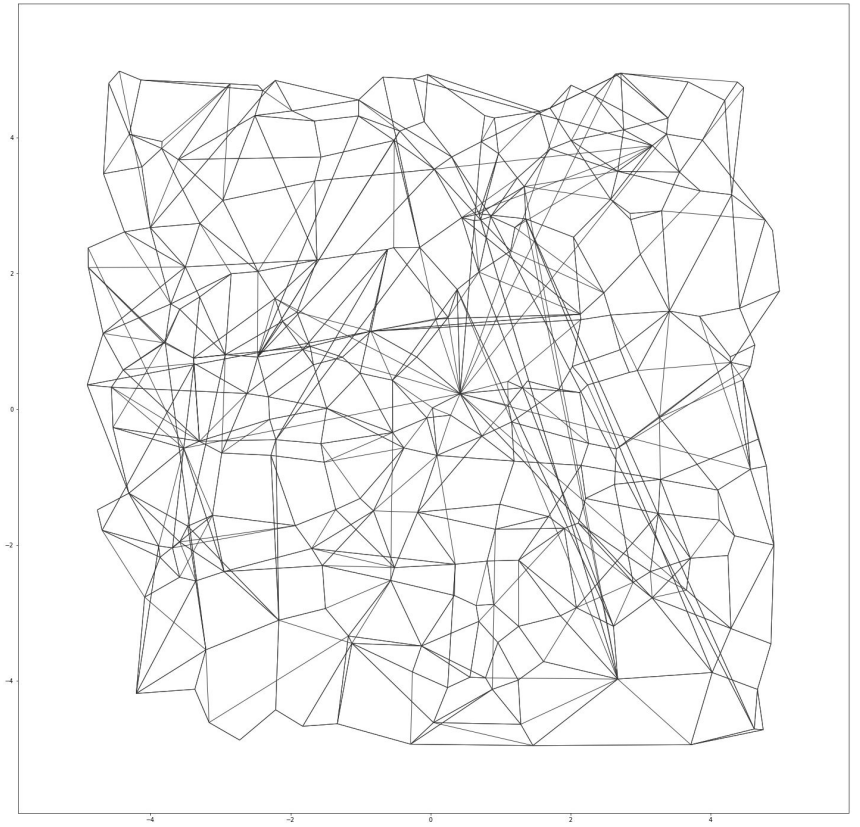
- Sort V in increasing order of distance from p
- V' ← {v_1}
- for i ∈ {2, …, |V|} while |V'| < R
  - if for any v ∈ V', $d(v,v_i) < d(p,v_i)/\alpha$, skip i & continue
  - else, add v_i to V'

Index properties: dependence on alpha

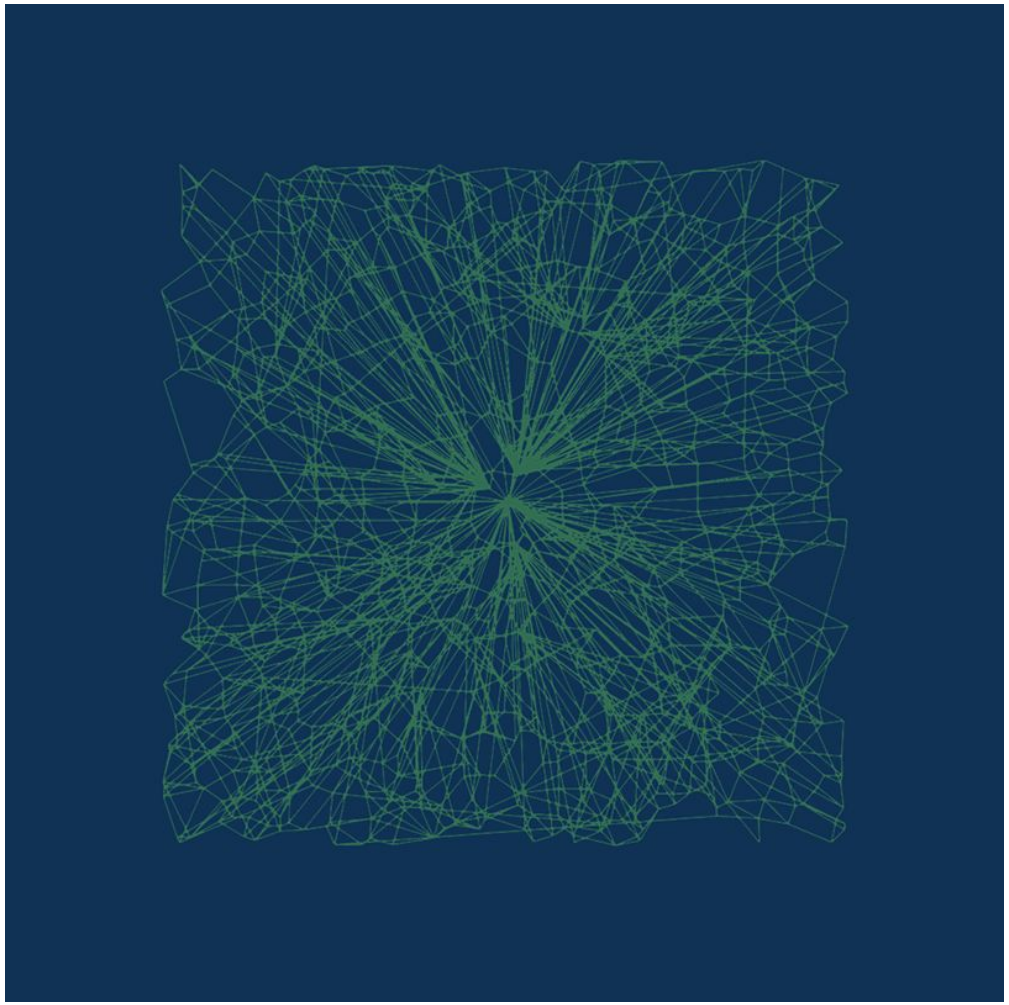
α=1



α=1.2



DiskANN BFS Animation [credit: Weaviate]



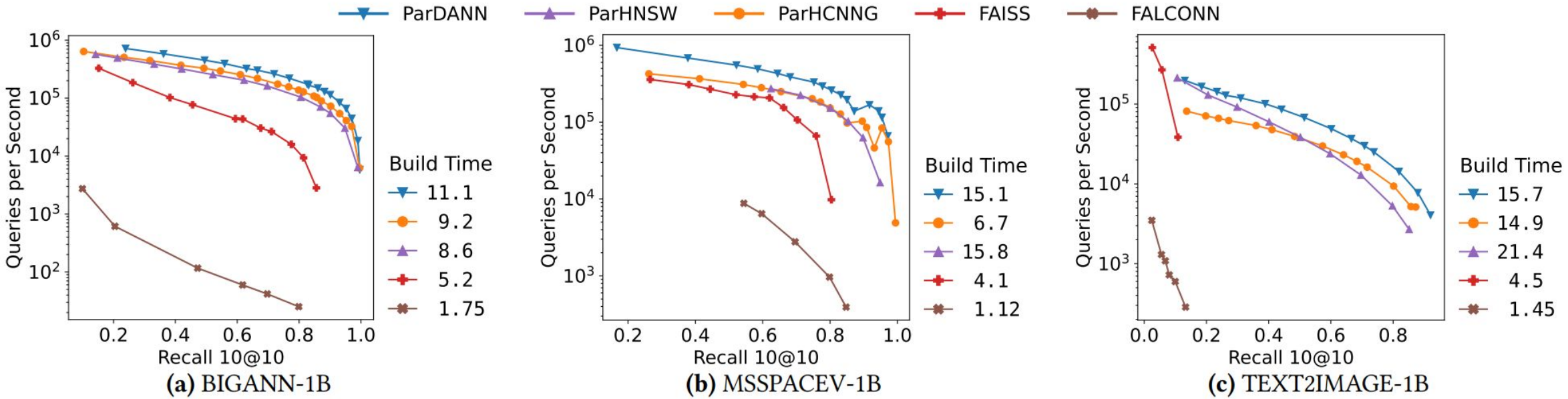
HCNNG [Muñoz, Gonçalves, Dias, Torres, Pattern Recognition vol. 96]

- Repeat d~50 times (for a average d-degree graph)
  - Pick random points p_1 and p_2
  - Divide the space into two with the perpendicular bisector of p_1 --- p_2
  - Recursively divide both half-spaces similarly to create a space-partitioning tree
  - Leaf node: stop when fewer than (say) 1000 points in the partition.
  - In each leaf, consider the complete graph between points weighted by their distance.
  - Compute Minimum Spanning Tree with max-degree 3 (using distance as weights).
- Final Edge set Union of MSTs in all leaves in all iterations!

Parlay ANN library [Dobson, Shen, Blelloch, Dhulipala, Gu, Simhadri, Sun, PPoPP'24]

- High quality parallel implementations that scale to 100+ threads on up to 1B sized dataset
- Link: ParlayANN/algorithms at main · cmuparlay/ParlayANN (github.com)
- Various optimizations
  - Incremental batch construction (doubling range at the beginning)
  - Memory optimizations

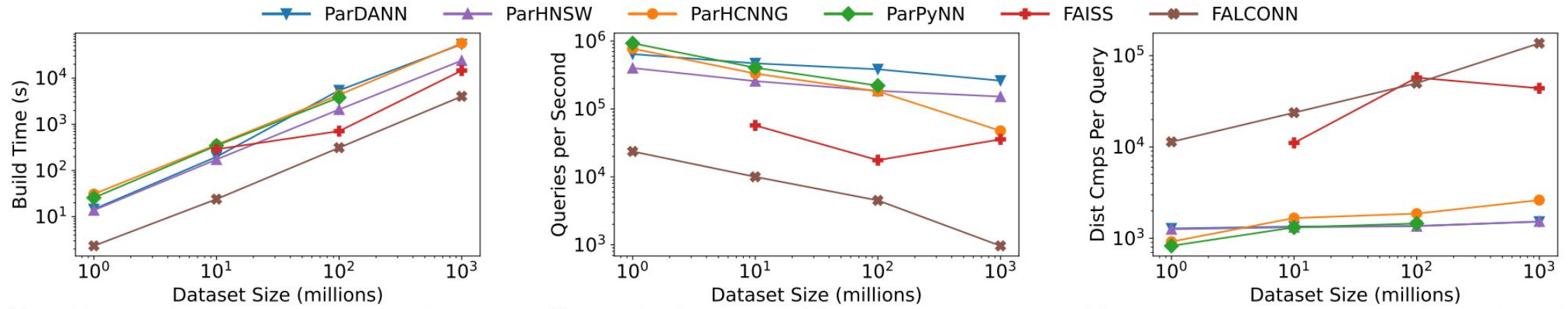
Build time and QPS comparison



(a) BIGANN-1B **(b)** MSSPACEV-1B **(c)** TEXT2IMAGE-1B

Figure 3. Build time (hours), QPS and recall for all algorithms on billion-size datasets.

Scaling properties



(a) Build times shown on MSSPACEV as dataset size increases.

(b) QPS for fixed recall (.8) on MSSPACEV as dataset size increases.

(c) Dist computations per query for fixed recall (.8) on MSSPACEV as dataset size increases.

Figure 4. Figures showing the effect of dataset size on different metrics using the MSSPACEVdataset.

Analysis [Xu, Indyk, NeurIPS'23]

- Convergence analysis of (simplified) DiskANN in terms of doubling dimension and Diameter of dataset

Outline

- Part 1 [Efficiency]
  - Graph indices: Search, Construction and empirical analysis
    - (H)NSW,
    - NSG, Vamana/DiskANN
    - HCNNG
  - (Limited) analysis of convergence properties

- Part 2 [Versatility]
  - Disk-based indices
  - Streaming indices
  - Filtered search
  - Out of Distribution queries

ANNS at scale: Size, Speed and Freshness

	Web Search & Reco	Email Search	Enterprise search
Index Size	~1 trillion pages	100s of trillions of sentences	Trillions of paragraphs across documents
Update Rate (latency <1s)	~10K/sec	Ingest new email, Purge deletes	~1% change/day
Search latency/QPS	<10ms 10-100K+ Queries/sec	100s of ms	10-100ms

Problem 1: Existing algorithms use in-memory indices for <10ms latency and high throughput

DiskANN [NeurIPS'19]: Index 5-10x more points/machine using inexpensive SSDs; serve with <10ms latency and 10000+ QPS.

→ 10,000s of machines (100GB DRAM) to serve a trillion-point index for web search

Problem 2: High-quality indices are graph-based, and hard to update. Rebuilt from scratch periodically.

Fresh-DiskANN [arXiv:2105.09613]:
DiskANN + Real-time freshness + 1000s of updates/sec

→ 10,000s of machines to periodically rebuild indices every 6/12/24 hours

ANNS Challenges: Not all queries are simple

Problem 3: “Predicated” or “filtered” queries, e.g.,

- Best URLs/doc/Ads for query that match user’s region/language/site
- Image matching query with license X

→ Low recall for most predicates, especially infrequent ones

Filtered DiskANN [WWW’23]: Order of magnitude higher QPS and lower query latency; high recall even for rare predicates

Problem 4: Out-of-distribution queries, e.g.,

- Image index, text queries
- Long text index, short queries

→ Low recall as algorithms overfit to index data distribution

OOD-DiskANN [arXiv:2211.12850] Indices that can adapt to query distribution

DiskANN [NeurIPS'19]: High recall, low latency via hybrid DRAM+SSD index

DRAM

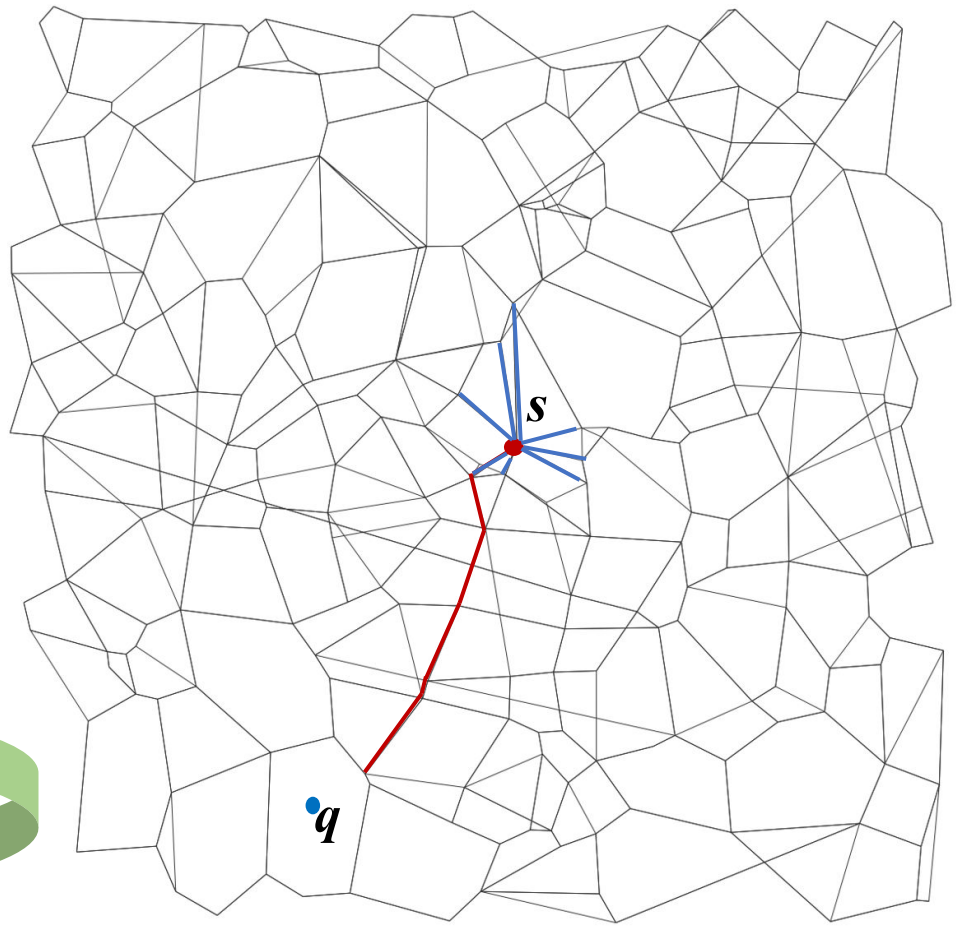
Compressed vectors (~32B)

SSD

1B Vectors (100-1000d) +
Graph(~100 degree) ~ 500GB-1TB
Low Diameter (<10 hops)

GreedySearch(***q***)

- Let ***p := s*** (start node)
- Fetch neighbors of p from SSD
- Use **compressed representation of points** to find neighbor ***p*** closest to ***q***



Index properties: BFS and graph diameter (α=1.2)

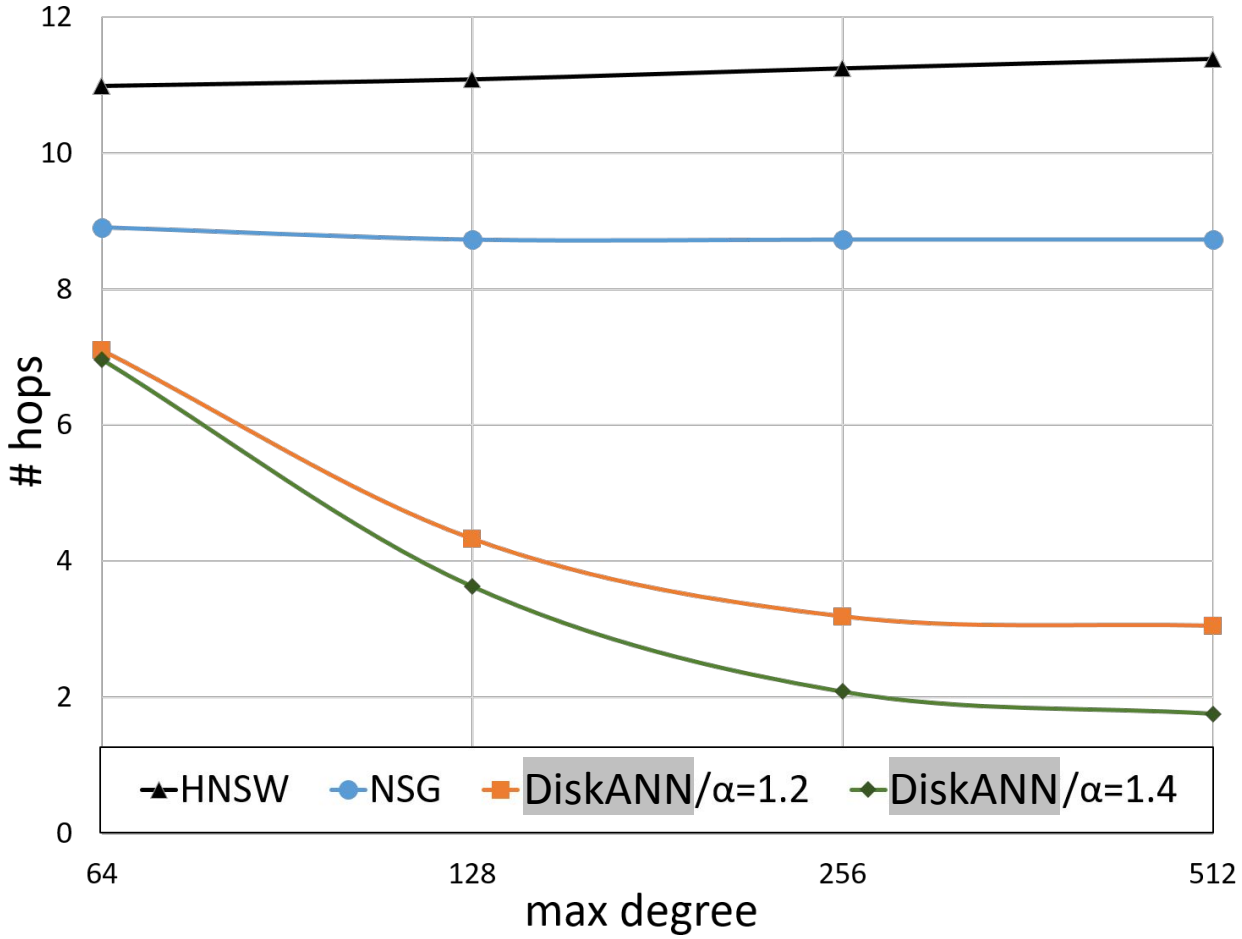
BFS level (from designated start point)	# points at BFS level *i* *(1billion point MS SPACEV dataset)*
1	100
2	9946
3	891273
4	52050789
5	682738849
6	252099714
7	11835439
8	190225
9	104202
10	47656
11	11095
12	1536
13	212
14	24
15	1

98% of points <= 6 hops from start

BFS level (from designated start point)	# points at BFS level *i* *(1billion point MS BIGANN dataset)*
1	73
2	3868
3	213058
4	10864635
5	226069439
6	673168120
7	88167475
8	1487493
9	6645
10	798
11	131
12	16
13	1

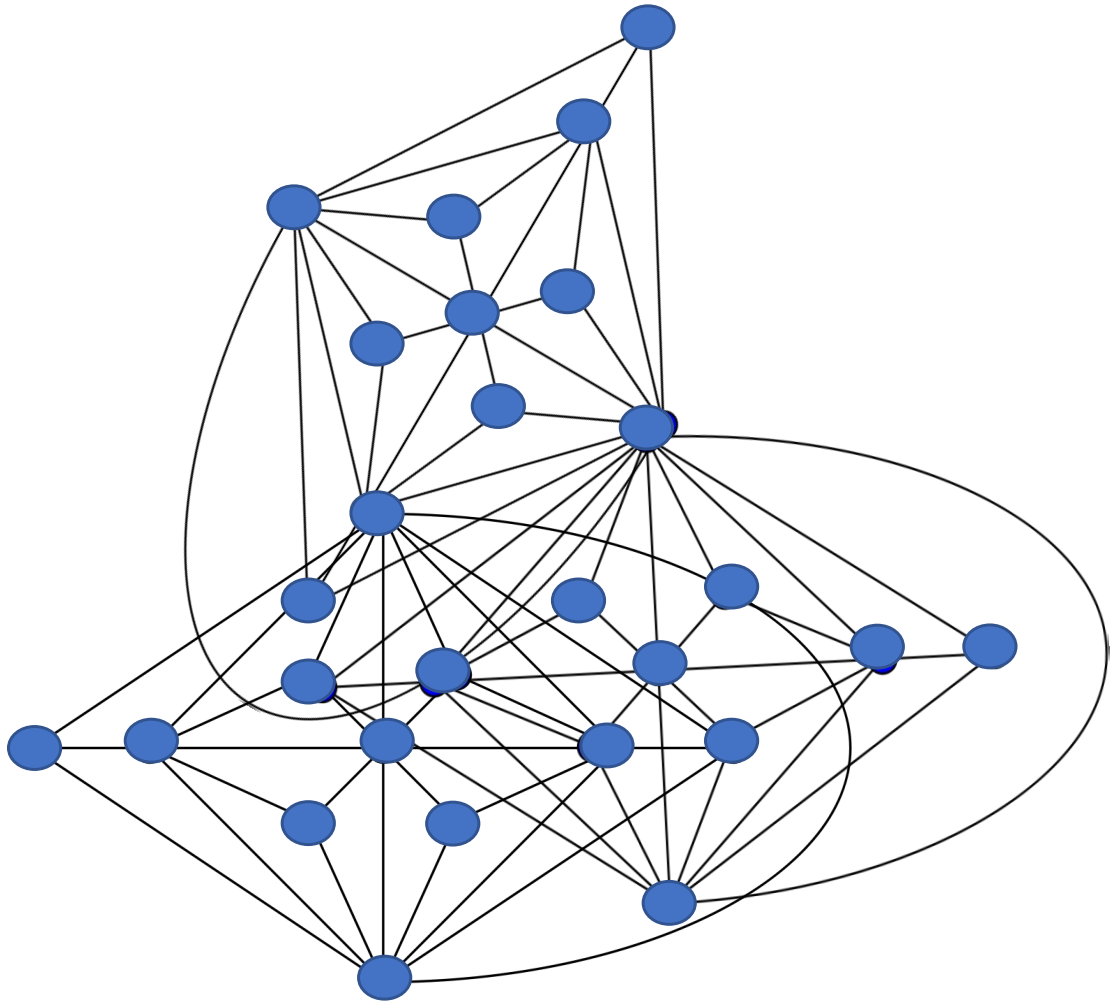
90% of points <= 6 hops from start

Round-trips to SSD: Comparison with other graph algorithms



Search Parameters
PQ size 32, Ls=30
Beam-width, W = 4

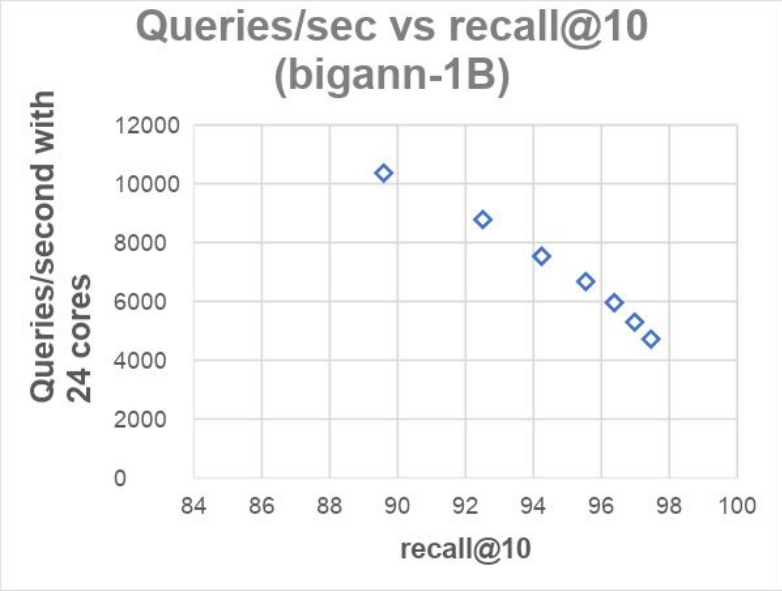
Sharded construction for datasets larger than memory

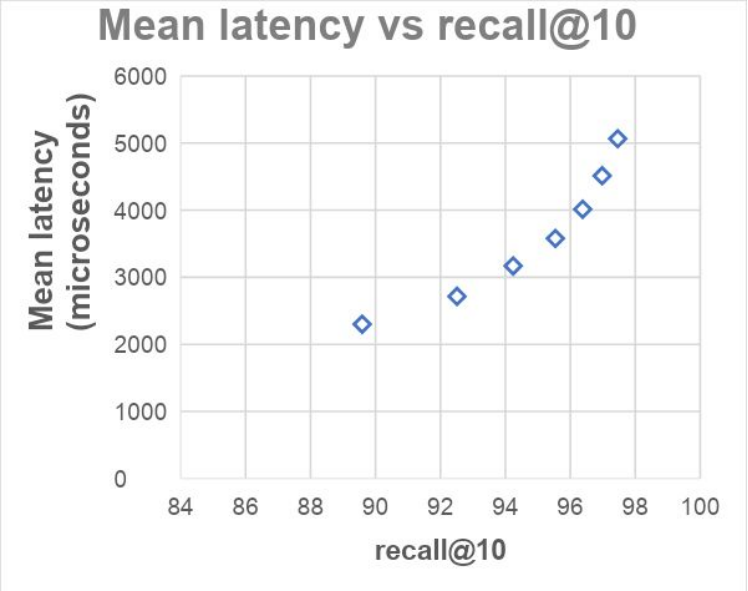


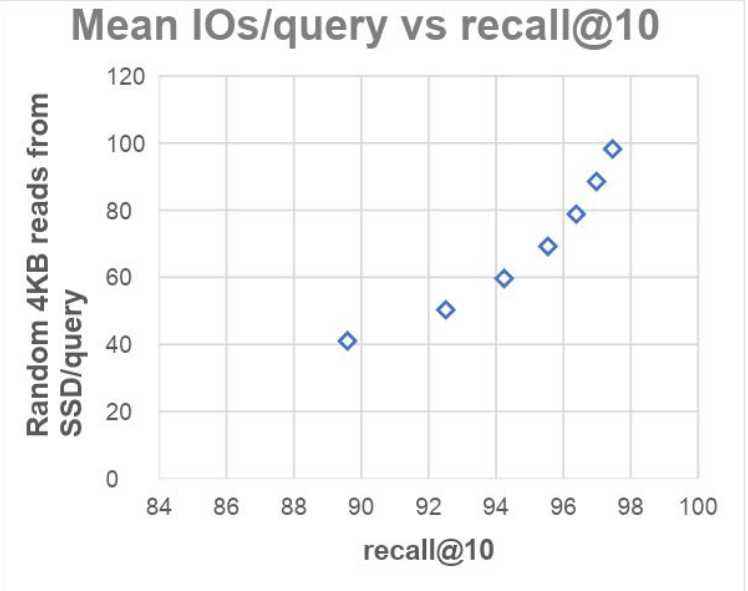
Divide & Conquer:

- Cluster data into partitions/clusters that fit into memory (1B points + 100GB RAM limit -> 10 to 20 shards)
- send each point to be indexed to 2 nearest clusters
- Build graph on each shard
- Final graph is the edge union of smaller graphs
- Cleanly avoids “boundary effects” which arise due to k-means partitioning

Recall, latency, QPS and IO/s for 100-degree graph







BIGANN dataset: 1Billion points in 128 dimensions

Memory footprint = 32GB + 250K adjacency lists cached in memory ~ 33GB

In comparison, in-memory graph indices (e.g., HNSW) would need 500GB+ DRAM

Can use as in-memory index, better than existing algorithms

DiskANN++ [Ni, Xu, Wang, Li, Yao, Xiao, Zhang, [2310.00402]]

Improved query performance with

- Better Start points
- Reordered disk layout with more IO locality

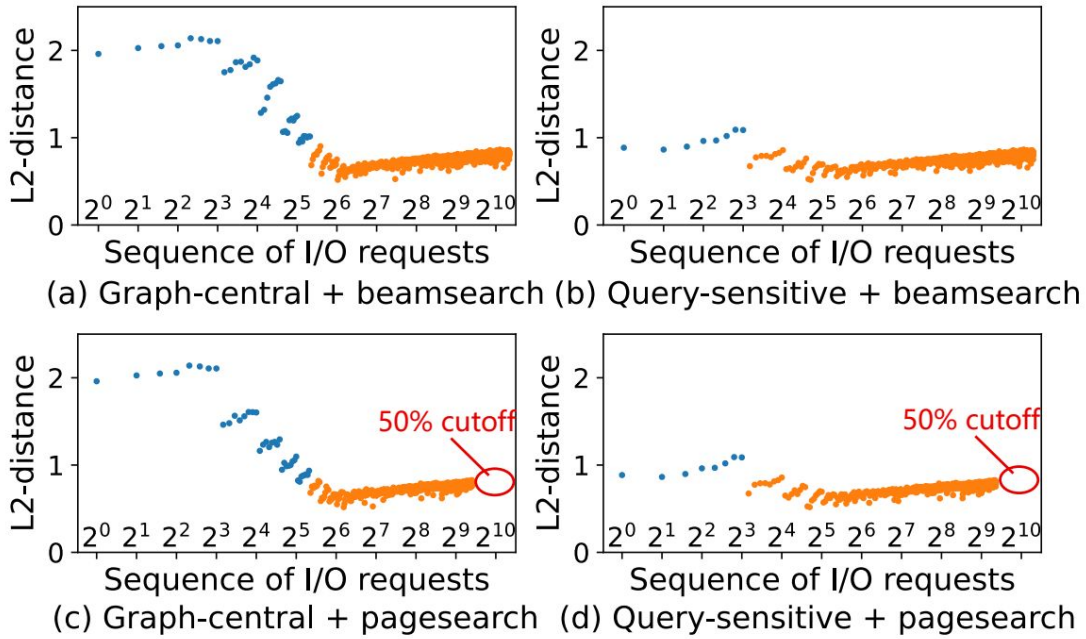
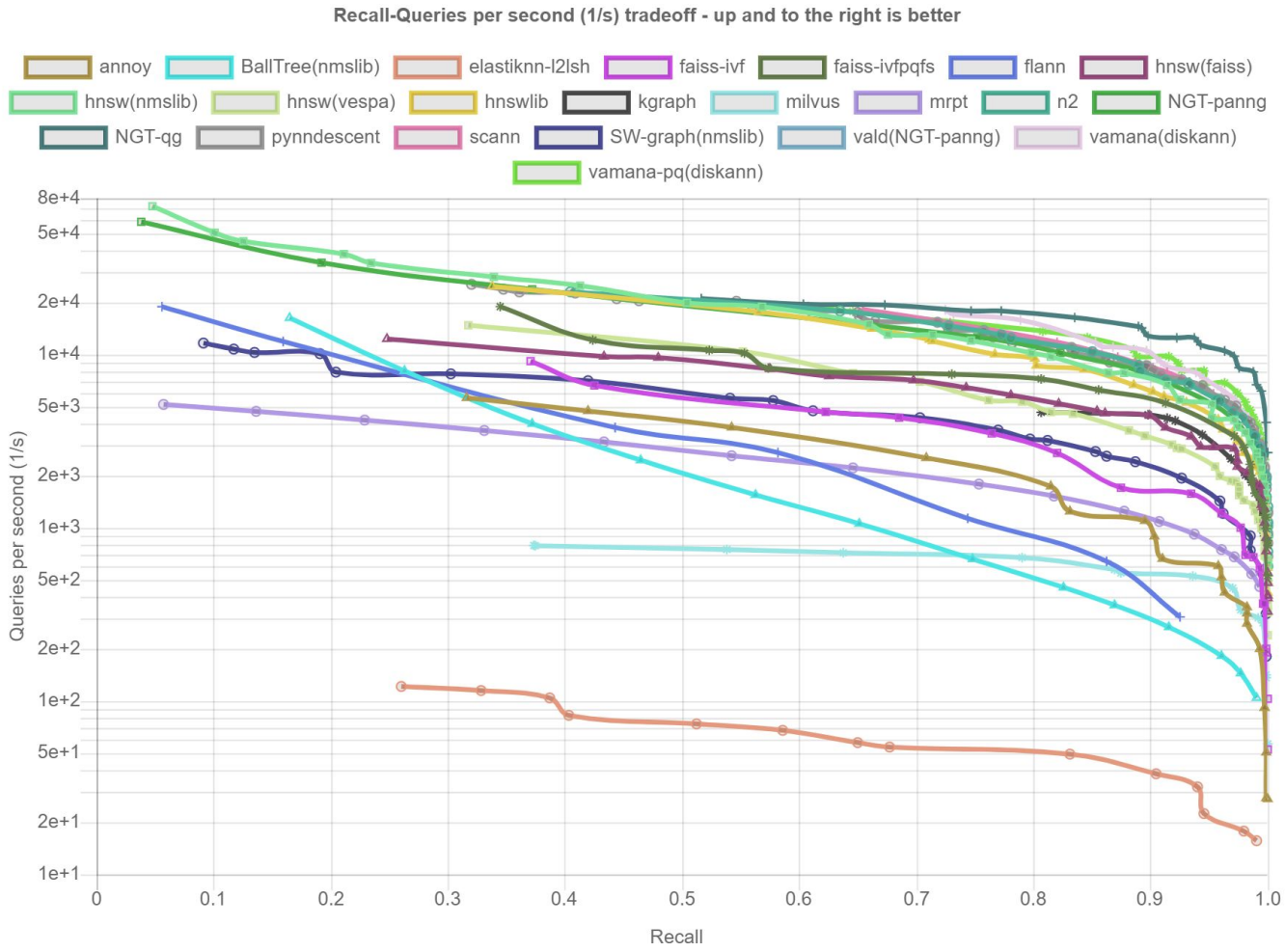


Figure 2: Comparison of our pagesearch and original beam-search with different entry vertex strategies (deep100M).

https://ann-benchmarks.com



SIFT Image descriptors
1M points,
128 dimensions,
L2 distance

NeurIPS'21: Billion-Scale Approximate Nearest Neighbor Search Challenge

https://big-ann-benchmarks.com/neurips21.html

Harsha Simhadri[*] (Organizer for Track T1/T2),
George Williams[§] (Organizer for Track T3), Martin Aumüller[¤], Matthijs Douze[†], Ravishankar Krishnaswamy [*+], Artem Babenko[‡], Dmitry Baranchuk[‡], Qi Chen[*], Lucas Hosseini[†], Gopal Srinivasa[*], Suhas Jayaram Subramanya[#], Jingdong Wang[^]

*Microsoft Research, [§]GSI Technology, ¤IT University of Copenhagen, †Facebook AI Research, ‡Yandex Labs, [#]Carnegie Mellon University, [+]IIT Madras, ^Baidu

Track 1:
Standard Azure hardware,
limited DRAM (64GB)
Baseline: FAISS IVF + PQ

Track 2:
Standard Azure hardware,
Limited DRAM(64GB) + 2TB SSD
Baseline: DiskANN

Track 3:
Any hardware,
Cost- and Watt-normalized query throughput
Winner: Intel's adaption of DiskANN to Optane pmem

[Thanks to Microsoft for generous support including Azure credits for participants and organizers]

Six Billion-scale Datasets from INRIA/IRISA, Facebook (now Meta), Microsoft, Yandex

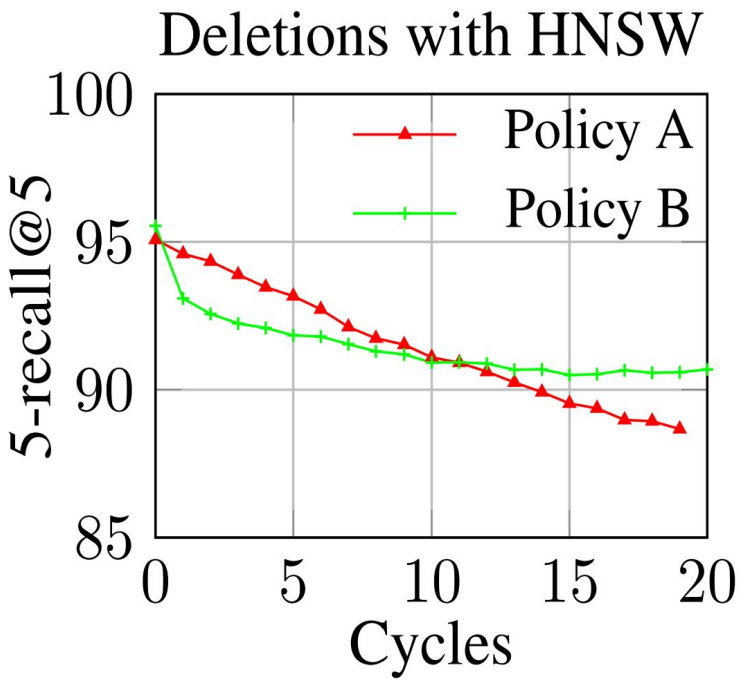
Dataset	Source	Size	Encoder/Task	Other notes
BIGANN-1B	CNRS/IRISA http://corpus-texmex.irisa.fr/	128 dims uint8 L2	SIFT descriptors for image similarity	
SSNPP-1B*	Facebook	256 dims uint8 L2	SimSearchNet++ image encoder	Range search
SpaceV-1B*	Microsoft https://github.com/microsoft/SPTAG/tree/master/datasets/SPACEV1B	100 dims int8 L2	Docs and queries encoded by Microsoft SpaceV Superior model to capture generic intent representation.	
Turing-ANNS-1B*	Microsoft Turing	100 dims float L2	Bing queries encoded by Turing AGI v5 encoder.	
Text2Image-1B*	Yandex https://research.yandex.com/datasets/biganns	200 dims float Inner-product	Images encoded by Se-ResNext-101 model, queries are text encoded by a variant of DSSM	Cross-modal; Query distribution different from index set
DEEP-1B	Yandex https://www.cv-foundation.org/openaccess/content_cvpr_2016/app/S09-38.pdf	96 dims float L2	GoogLeNet pretrained for Imagenet classification task + PCA + l2 normalized	

[* new datasets released for the competition]

Track 3 [any hardware]
Capex + Opex improvement over baseline for 100,000 QPS

Rank	Submission	Team	Hardware	Status	Score	Deep1B	BigANN	MSTuring	MSSpace	Text2Image	FBSSNet
1	optanne_graphann	Intel	Intel Optane	final	**$-3,978,198.83**	$16,086.82	$15,439.92	$16,347.45	$16,382.81	$103,599.49	-
2	cuanns_ivfpq	NVidia	NVidia GPU	final	**$-2,314,829.98****	$303,929.39	$304,166.48	$153,151.00	$153,155.12	$916,823.34	-
3	cuanns_multigpu	NVidia	NVidia GPU	final	**$-2,268,943.17****	$151,009.85	$150,824.13	$151,726.30	$150,816.00	-	-
4	gemini	GSI Technology(org)	LedaE APU	final	**$-907,570.13**	$569,058.09	$569,210.35	$286,911.87	$398,163.18	$1,213,773.56	$629,442.91
5	faiss_t3	Facebook Research(org)	NVidia GPU	final	**baseline**	$545,633.16	$737,886.17	$853,857.46	$735,942.66	$1,272,735.86	$428,074.79
6	-	-	-	-	**-**	-	-	-	-	-	-

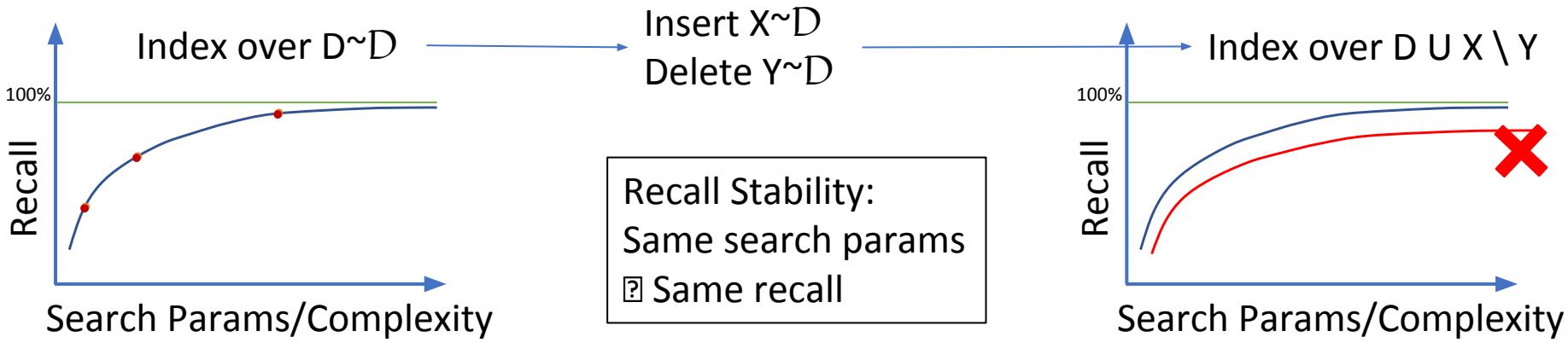
Deletes: Are they easy?



Policy A (drop): drop graph vertices corresponding to deleted points.

Policy B (shortcut): To delete vertex ***p***, for any pair of directed edges (p_{in}, p) and (p, p_{out}) in the graph, add the edge (p_{in}, p_{out}) in the updated graph.

FreshDiskANN: DiskANN + concurrent updates

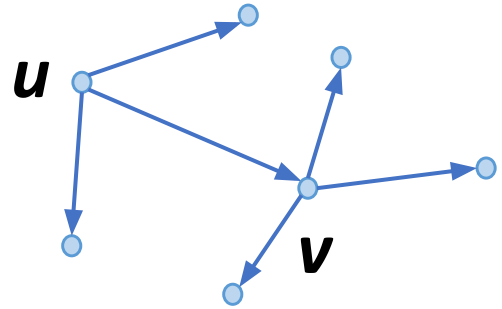


- First graph index that supports inserts/deletes with recall stability (empirically).
  - No theoretical guarantees or adversarial bounds, yet.

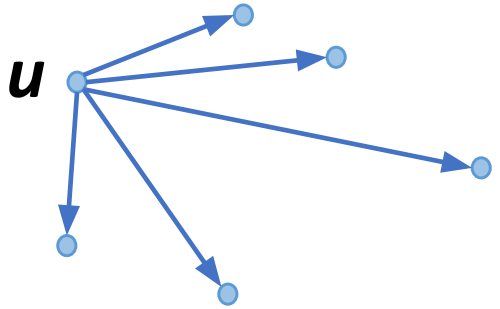
- A streaming system that supports concurrent updates and search
  - User facing latency: ~10ms search, ~1ms insert. Real time freshness.
  - 1000s of inserts, deletes and queries/sec/node
  - Primarily backed by SSD, Low memory footprint (128GB for ~1B points in 100 dimensions)

Delete (v)

For all ***u***, that are in-neighbors of ***v***,

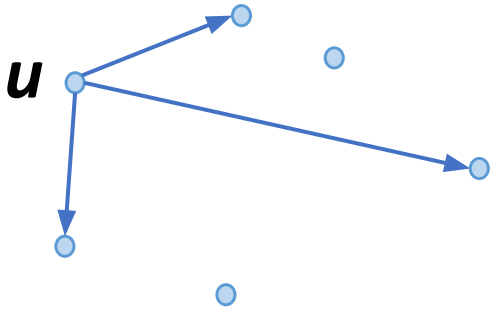


N(u) ← N(u) U N(v)



If ***|N(u)|*** exceeds budget

prune ***N(u)*** with ***α>1***



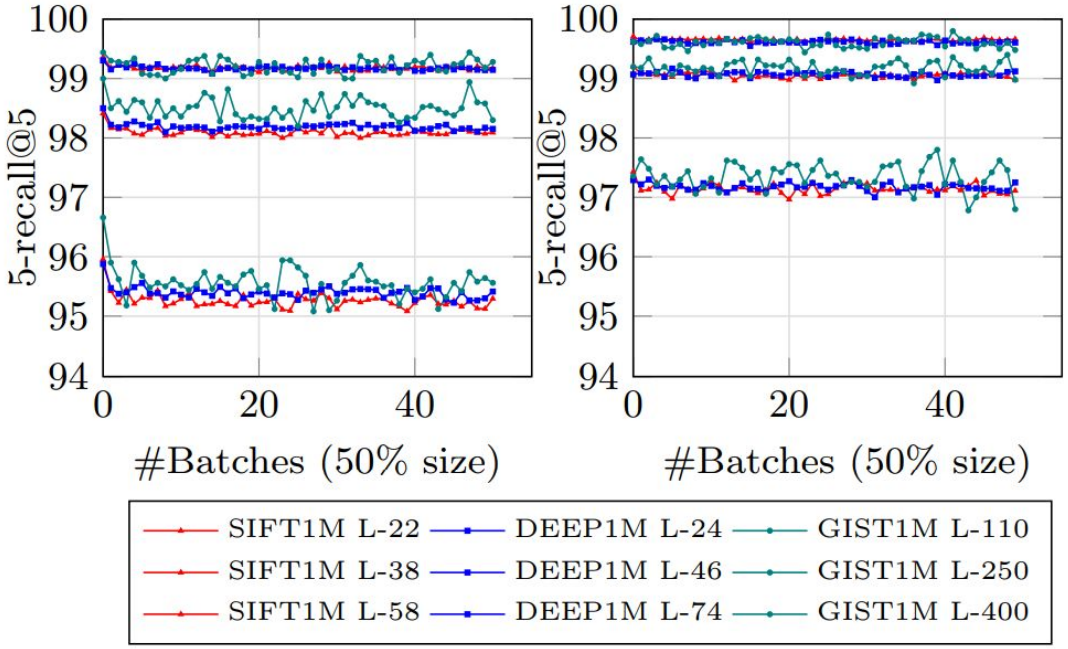
Eager execution requires in-graph, which doubles memory footprint
In practice, do lazily, to avoid in-graph and amortize cost better

Lazy Deletion via periodic consolidations

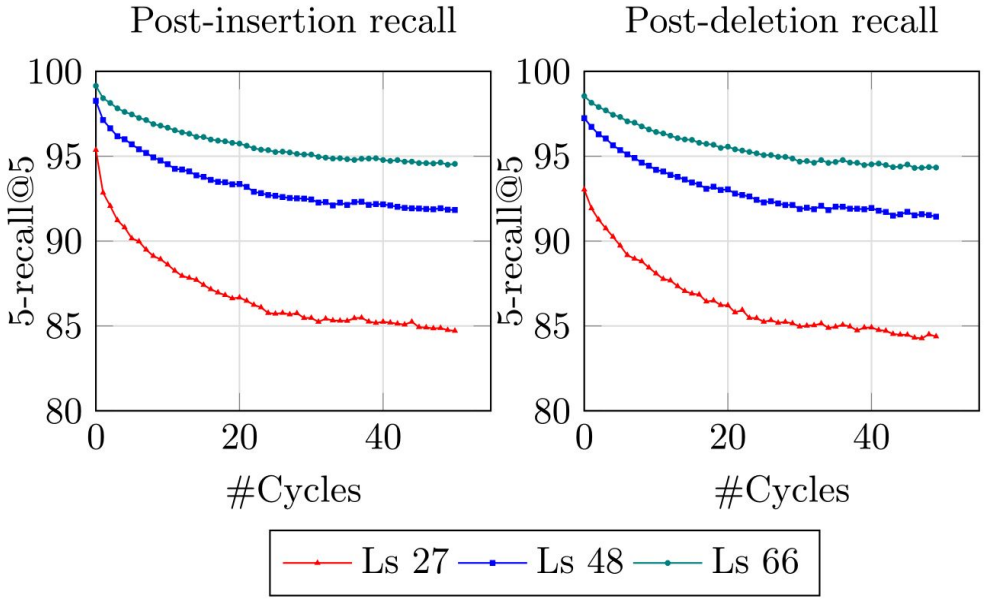
- Eager deletion requires in-neighbors
  - Double the memory
  - Much more complicated locking
- Lazy approach..
  - Maintain a deleted_set which keeps a record of points marked deleted.
  - Exclude from search results
  - Cleanup periodically
- For each vertex ***v*** in **V**
  - If ***v*** is deleted
    - Remove $N_{out}(v)$ from **G**
  - Else
    - For each ***w*** in $N_{out}(v)$ that is deleted
      - $N_{out}(V) \leftarrow N_{out}(v) \cup N_{out}(w) \setminus w$ *//exclude deleted nodes from* $N_{out}(w)$
      - If $|N_{out}(V)| >$ **max-degree**
        - $N_{out}(V) \leftarrow prune(N_{out}(V), v)$

Recall stability

Evolution of search recall of DiskANN ***(α=1.2)*** over 50 cycles of deletion (**right**) and reinsertion (**left**) of 50% of data for 3 datasets with varying search parameter.

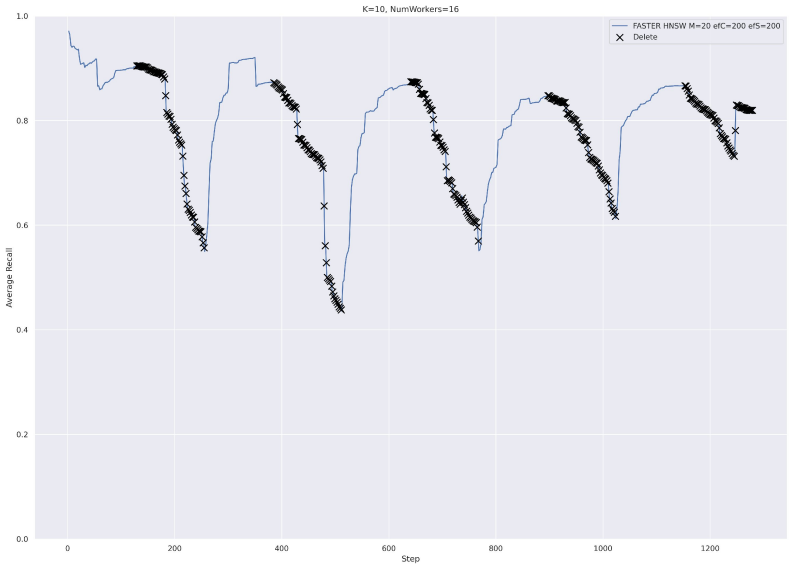


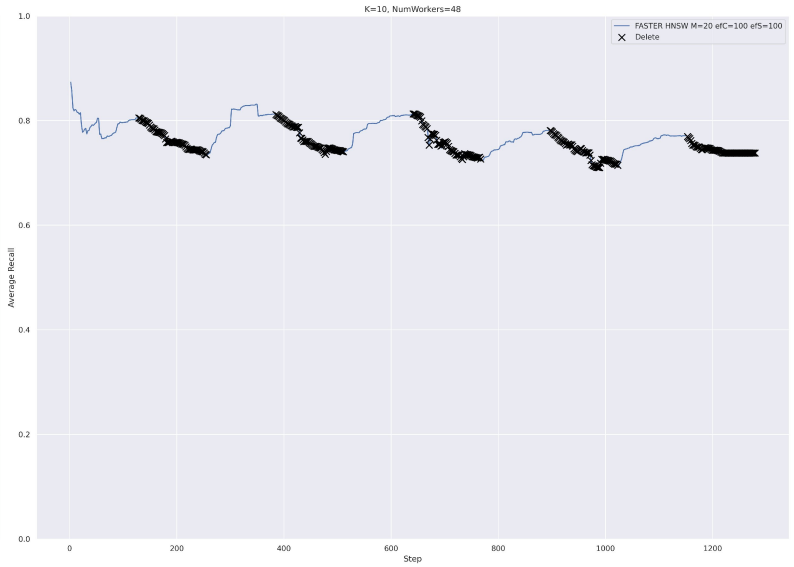
Same experiment with ***α=1***
Recall degrades over iterations

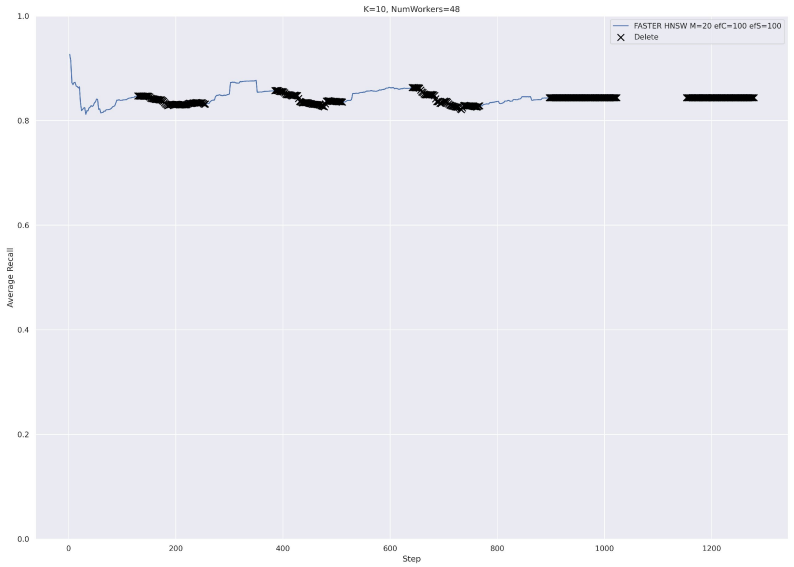


HNSW and DiskANN prune “final_runbook” on Big-ANN-Benchmarks from NeurIPS’23

[credit: Erkang Zhu]







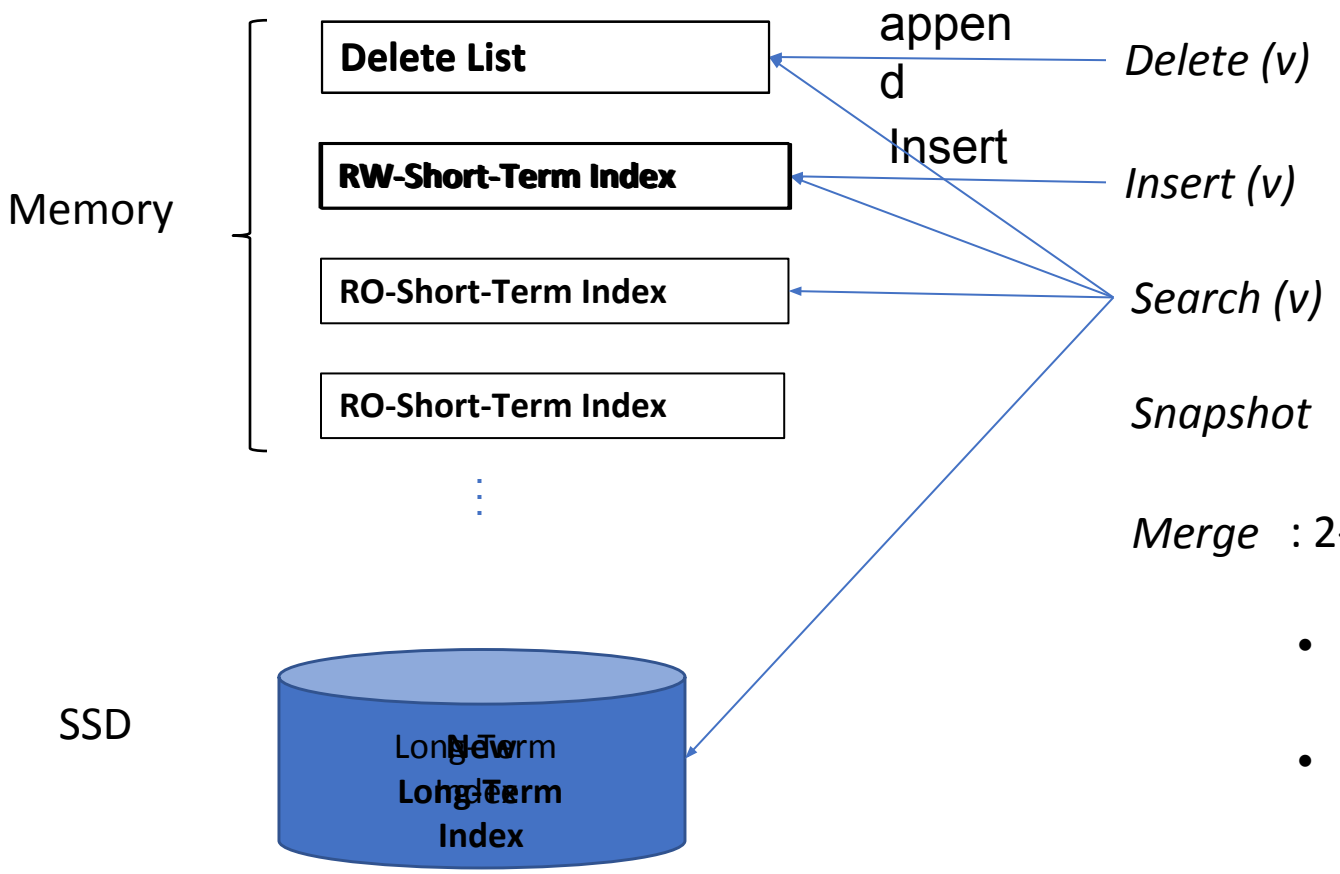
HNSW delete policy

Inspect candidates by decreasing distance, select a candidate as neighbor only when the distance between the source point and the candidate is greater than the distance between the candidate and all of

HNSW
+
DiskANN edge repair
Applied lazily (consolidate_delete)

HNSW
+
DiskANN eager edge repair

Fresh-DiskANN System



Merge : 2-pass streaming algorithm

- DRAM footprint proportional to #new points, not overall index
- Periodically invoked in the background

Comparison with PLSH on 1B point dataset

	Machine Settings	Search throughput Queries/ second	Freshness (Insert/ delete latency)	Insert /Delete throughput	Recall	Merge time	Notes
Fresh-DiskANN	1 machine 128GB RAM, 48 core Xeon 8160 Samsung PM1725a SSD	1800/s with 10 cores	<5ms insert Few µs for delete	1800/sec	90% recall@1 95%, 98% possible with slightly higher latency	~4.5 hours for 30M inserts + 30M deletes ~2.5 hours for 30M inserts	40, 2, 1, 10 threads for merge, insert, delete, search
DiskANN + rebuild daily	2 machines for build 1 machine for serve	“	24 hours	N/A	“	1 days to rebuild with 2 machines	
PLSH [VLDB’13]	100 machines 64GB DRAM, 8 core Xeon E5-2670, 64GB DRAM (total: 800 cores, 6.4TB DRAM)	~700/sec across 800 cores	<1sec	20000/sec		20sec for 1M entries	

Performance characteristics on 800M point dataset

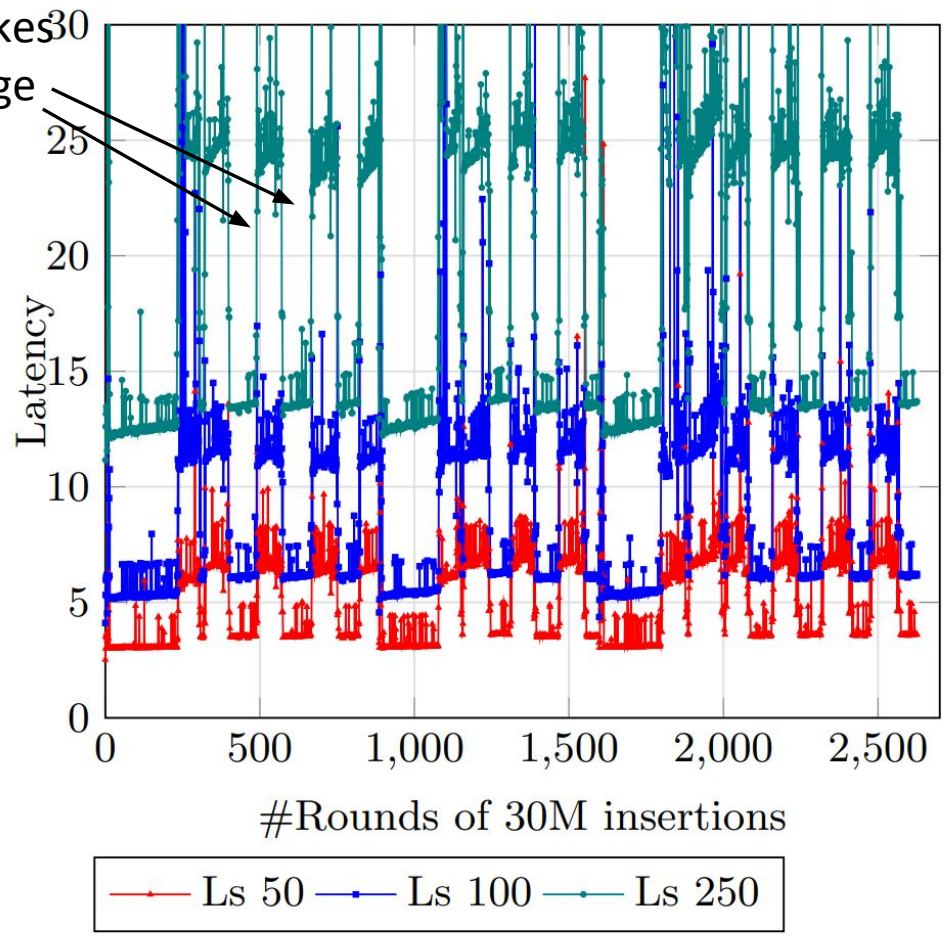
Machine:
48 cores, 2x Xeon 8160 CPUs
Samsung PM1725a SSD

Resource allocation
- Max 128GB memory used
- 40, 2, 1, 10 threads for merge, insert, delete, search

Background merge process into 800M point index takes
- ~4.5 hours to process 30M inserts and 30 deletes
- ~2.5 hours to process 30M inserts
- Compare with 2 days to rebuild index from scratch.

User facing performance
- Insert takes ~1ms, delete takes <1 microsec
- Supports **1800 inserts/sec + 1800 deletes/sec** in steady state with above thread allocation without merge backlog.

Search latency trends over 5 days



Mean search latencies in milliseconds for search recall of >90% (Ls 50), >95% (Ls100) and >98% (Ls250) search recall over the course of ramping up an index from 100M points to 800M, in increments of 30M.

Filtered Queries: Problem Statement

Given

- Point Set $P = \{p_1, p_2, p_3, \ldots, p_n\}$ in R^d,
- A set of labels $\{L_1, L_2, \ldots, L_k\}$,
- An association p_i ? $\{L_{i1}, L_{i2}, ..\}$ of labels to each point

Construct an index that efficiently supports queries that filters on the label..

- For query $<q, L_q=\{L_{q1}, L_{q2}, \ldots\}>$, find points in P nearest to q *associated with any of the labels in* L_q

Examples:

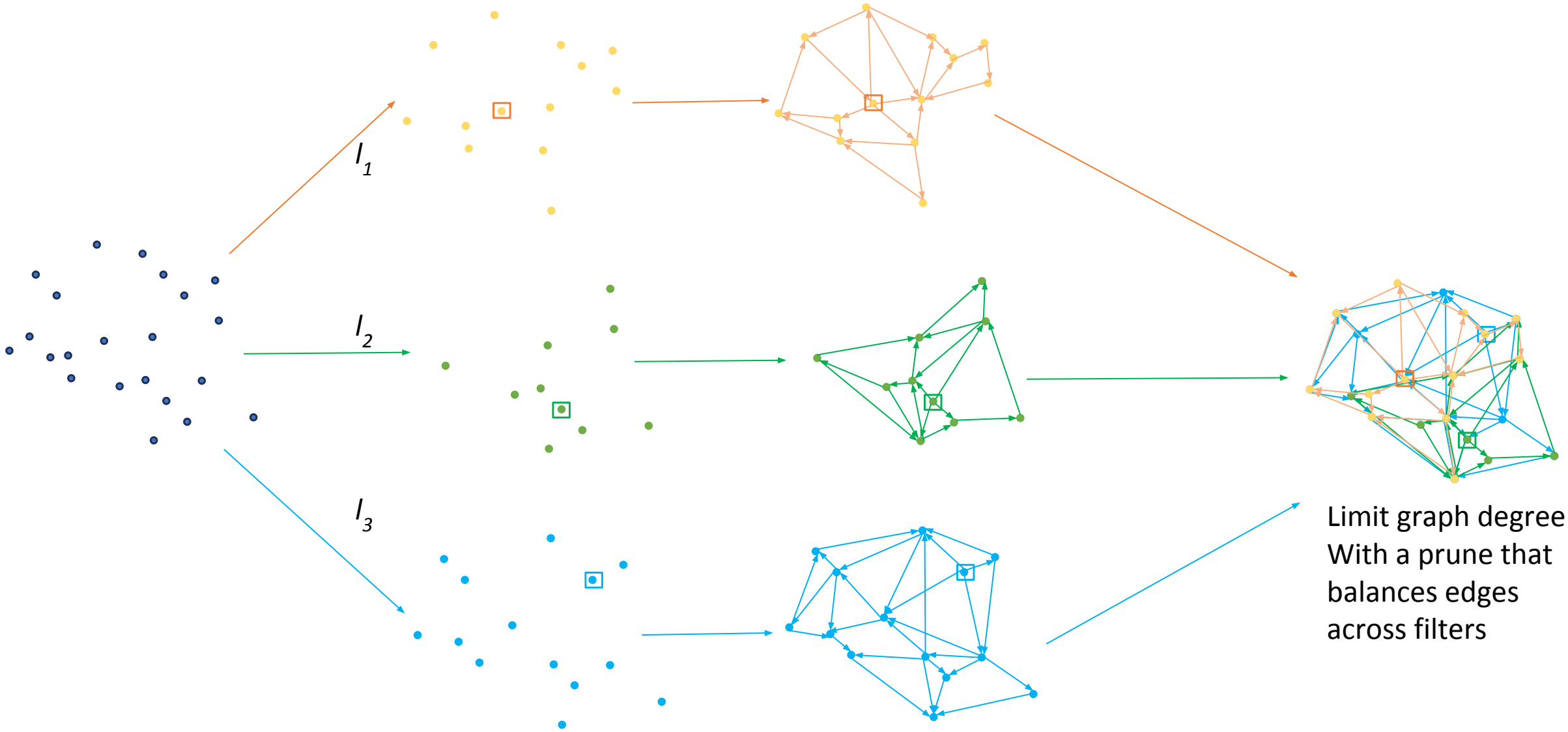
- Documents accessible to a subdomain (eecs.berkeley)
- Ads relevant to a market
- Shopping with price and rating filters

Possible approaches to filtered queries

- **Post-filtering:** standard ANN algorithm following by filter by label
  - Not so good for “rare” labels
- **Pre-filtering**: construct a separate ANN index for each label
  - Too many indices, since the relation between points and labels is many-to-many
- **In-line filtering**: Apply predicates as you traverse the index

- **Goal**: Match the search efficiency of “unfiltered” graph-based ANN search.
  - + Streaming
  - +External memory index

Index using filter information + vector geometry



Limit graph degree
With a prune that
balances edges
across filters

Filtered Prune Principle

For any triplet of vertices a, b, c, and constant $\alpha \geq 1$, the directed edge (a, c) can be pruned out of the graph if

- (1) the edge (a, b) is present,
- (2) the vector x_b is closer to x_c than x_a is to x_c by α, i.e., $\| x_b - x_c \| \leq (1/\alpha) \| x_a - x_c \|$, and
- (3) F_b contains all common filter labels of F_a and F_c, i.e., $F_a \cap F_c \subseteq F_b$.

Datasets

Dataset	Dim	# Pts.	# Queries	Source Data	Filters	Filters per Pt.	Unique Filters	100pc.	75pc.	50pc.	25pc.	1pc.
Turing	100	2,599,968	996	Text	Natural	1.09	3070	0.127	$1.56x10^{-4}$	$4.15x10^{-5}$	$1.54x10^{-5}$	$7.7x10^{-6}$
Prep	64	1,000,000	10000	Text	Natural	8.84	47	0.425	0.136	0.130	0.127	0.09
DANN	64	3,305,317	32926	Text	Natural	3.91	47	0.735	0.361	0.183	0.167	0.150
SIFT	128	1,000,000	10000	Image	Random	1	12	0.083	0.083	0.083	0.083	0.082
GIST	960	1,000,000	1000	Image	Random	1	12	0.083	0.083	0.083	0.083	0.082
msong	420	992,272	200	Audio	Random	1	12	0.083	0.082	0.082	0.082	0.082
audio	192	53,387	200	Audio	Random	1	12	0.085	0.084	0.083	0.082	0.081
paper	200	2,029,997	10000	Text	Random	1	12	0.083	0.083	0.083	0.083	0.082

Table 1: Datasets used in the evaluation and their statistics. Top 3 rows are real-world datasets; the rest are semi-synthetic.

Graph useful for “unfiltered” and disk search too

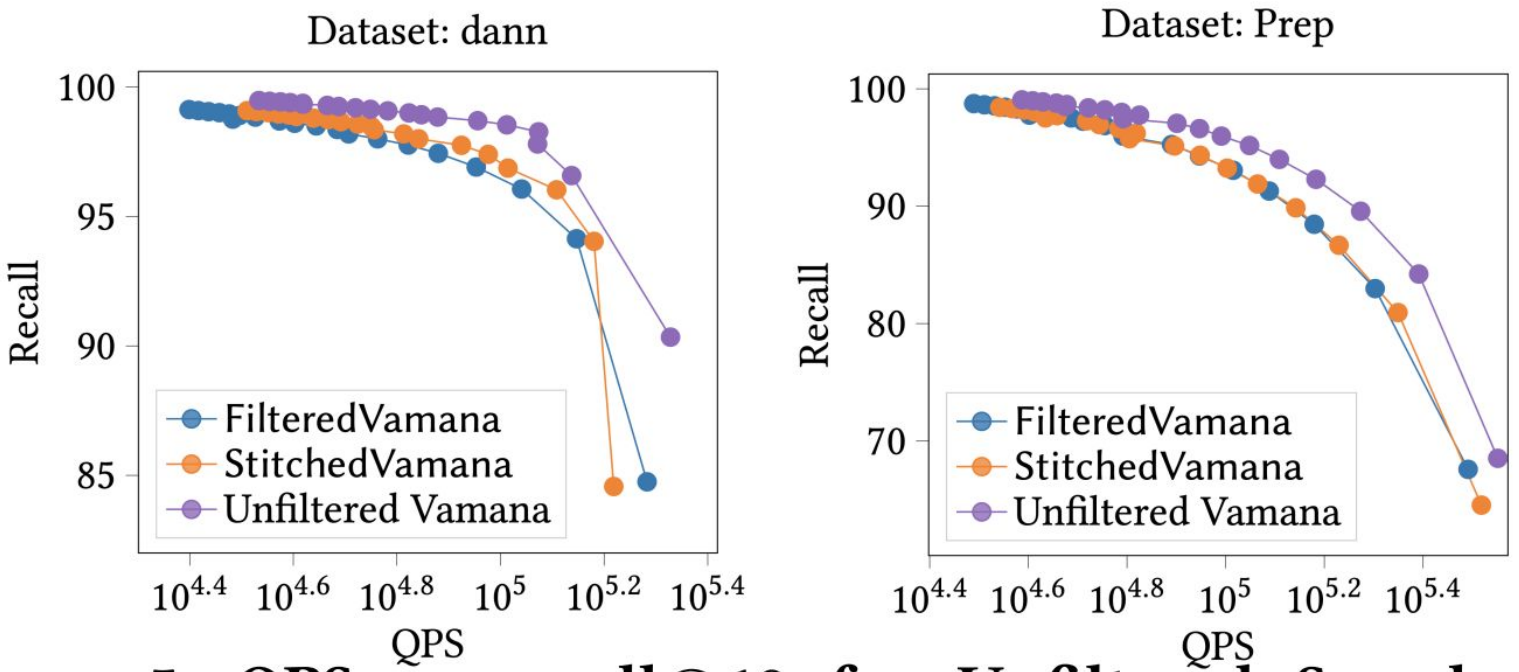
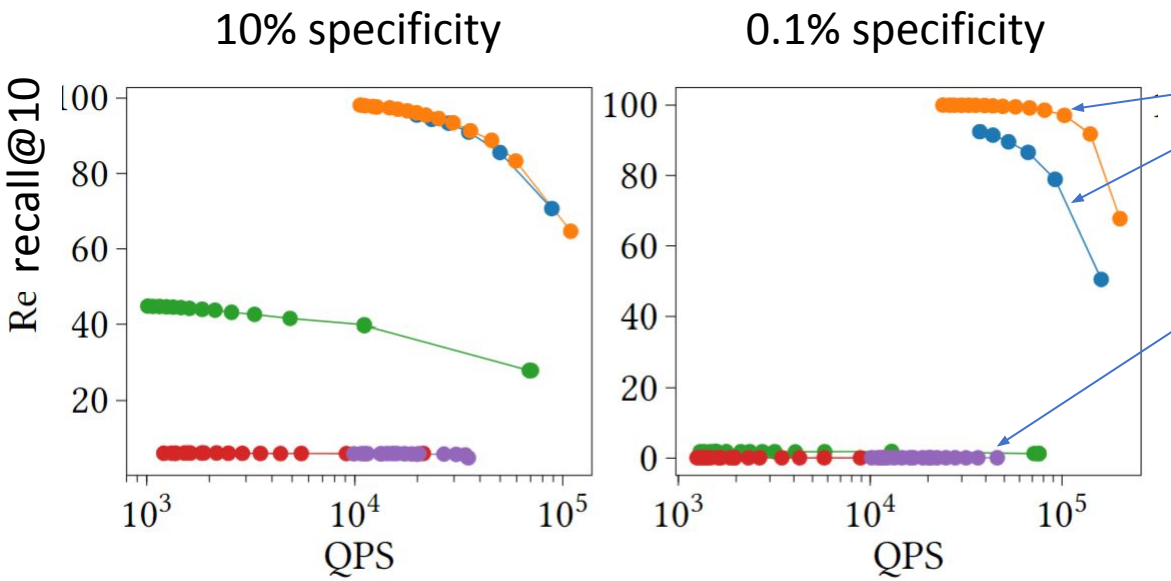


Figure 5: QPS vs recall@10 for Unfiltered Search on FilteredVamana **and** StitchedVamana **built on original labels.**

Comparison with other approaches

Index of enterprise text data.
Predicate = domain within enterprise.
Specificity = % data in the domain.



New algorithms that includes filter information
Orange = build + overlay
Blue = streaming version

Approaches used by VectorDBs
Green = IVF inline processing
Red = IVF post processing
Purple = HNSW post processing

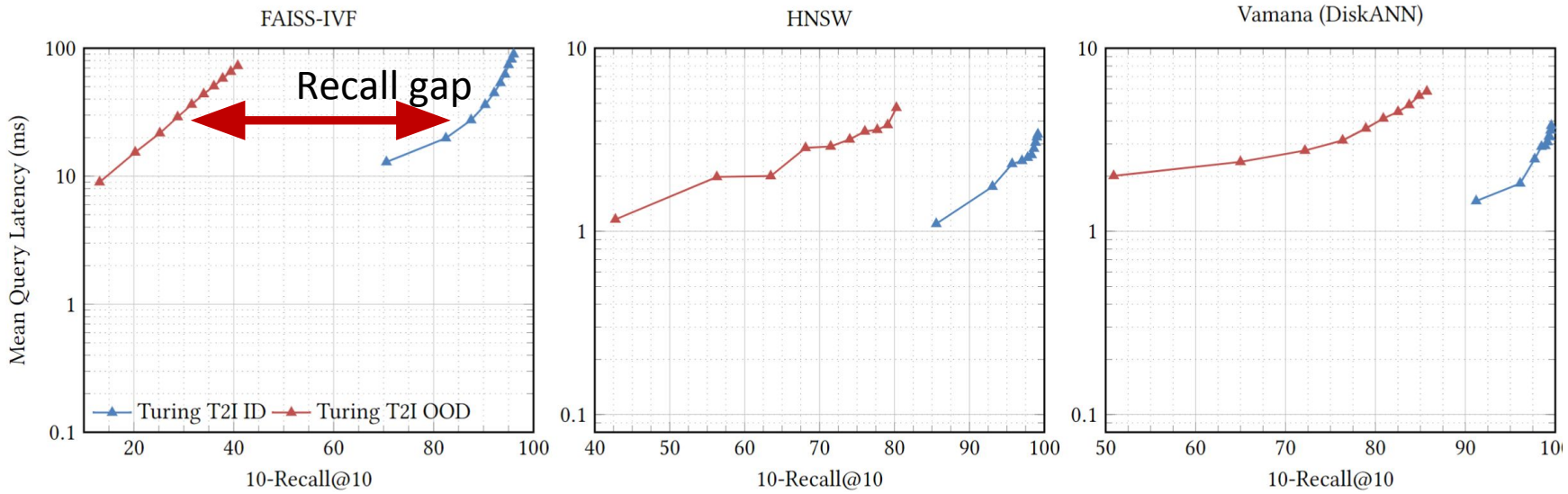
How about complex predicates?

- Access-controls
- Shopping filters (maker, price, rating, color, …)
- Image search (size, copyright, background, …)
- Arbitrary SQL expressions?

Possible with query plans [AnalyticDB-V, VBase, SingleStore,…]

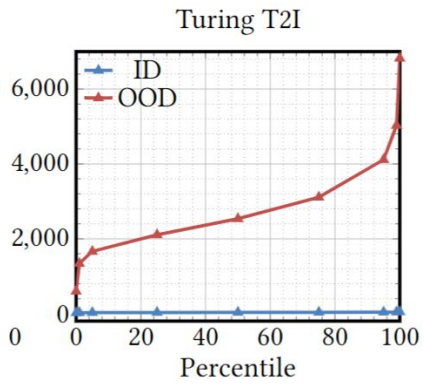
But can we do it algorithmically?

Current indices have low recall on out of distribution queries



Recall vs latency plots of 3 algorithms (FAISS-IVF, HNSW, DiskANN).
Notice the gap between **in-distribution (image queries, image index)** and **out-of-distribution queries (text queries, image index)**

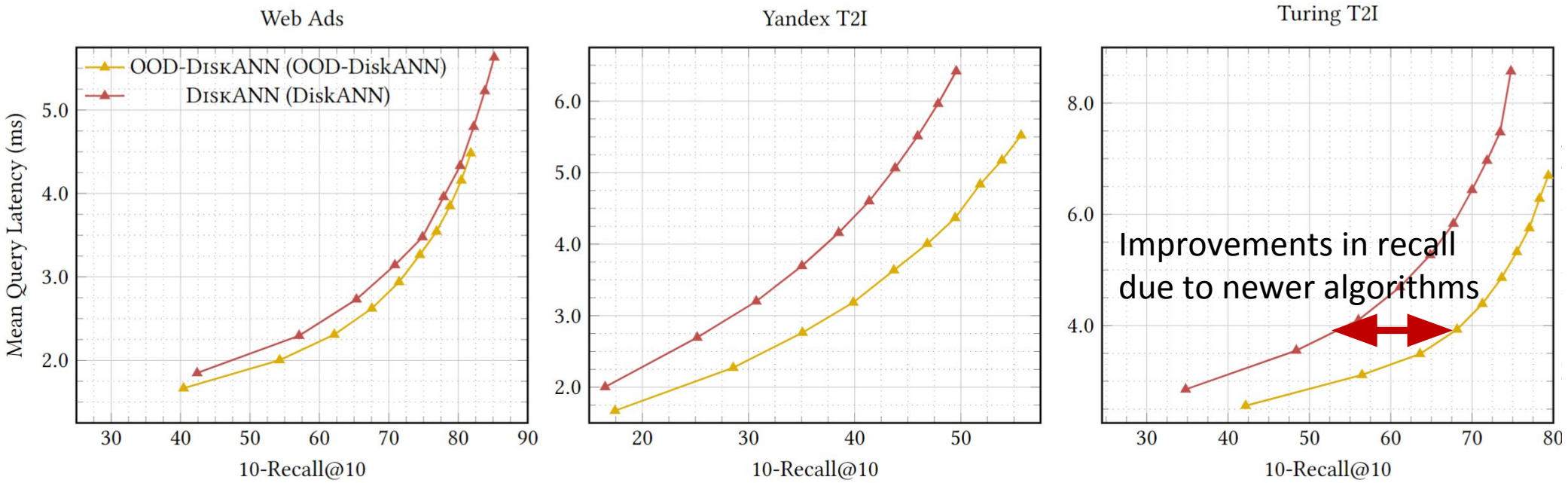
Overfitting to index distribution makes OOD queries hard



Histogram of Mahalanobis distances of In-distribution queries and Out-of-Distribution queries to the index data

- Data-dependent ANN indices overfit to index data distribution, leading to lower recall on OOD queries
- Lower recall/efficiency when queries are drawn from different distribution
- Typical of multi-modal scenarios: e.g., text queries – image index
- Can adapt indexing algorithms to query distribution with 10-20% recall gains at fixed latency.

OOD-DiskANN and improved results



NeurIPS'23: Billion-Scale Approximate Nearest Neighbor Search Challenge

https://big-ann-benchmarks.com/neurips23.html

4 tracks with 10M size datasets

- **Filtered queries**: YFCC + CLIP embeddings, filter = tags
- **Out-of-distribution**: Yandex T2I, Text queries, Image index
- **Sparse**: Bag of Words high-dimensional vectors,
- **Streaming**: MSTuring/SpaceV embeddings

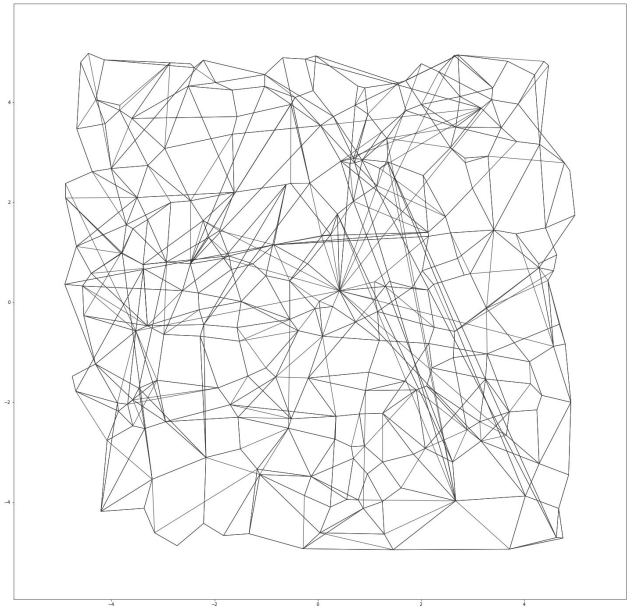
Martin Aumüller[¤], Dmitry Baranchuk[‡], Matthijs Douze[†],

Amir Ingber*, Edo Liberty*, Frank Liu [§], George Williams

¤IT University of Copenhagen, †Meta AI Research, *Pinecone, ‡Yandex Labs, §Zilliz

Vector indices vs Vector DB

DiskANN graph



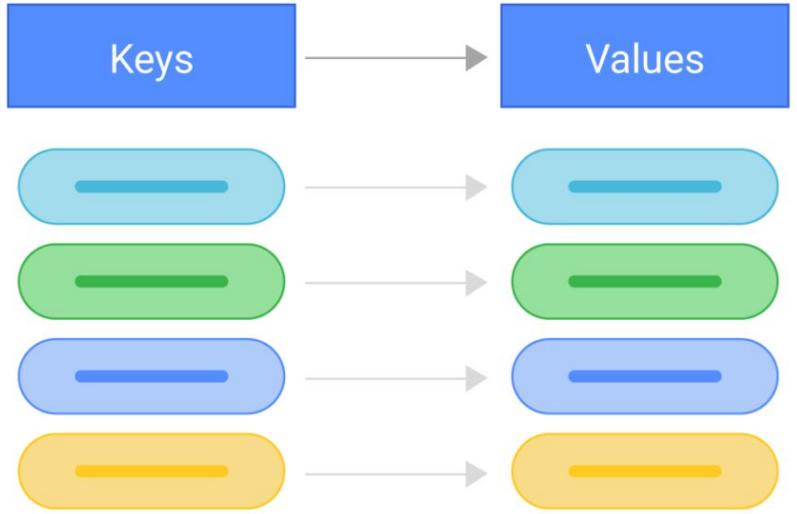
Search | Insert | Delete

No globally consistent state

Shoehorn
Into legacy DB

High costs &
perf overheads;
Lost features

K-V store or a
SQL database



CRUD

Checkpoint
/Restore

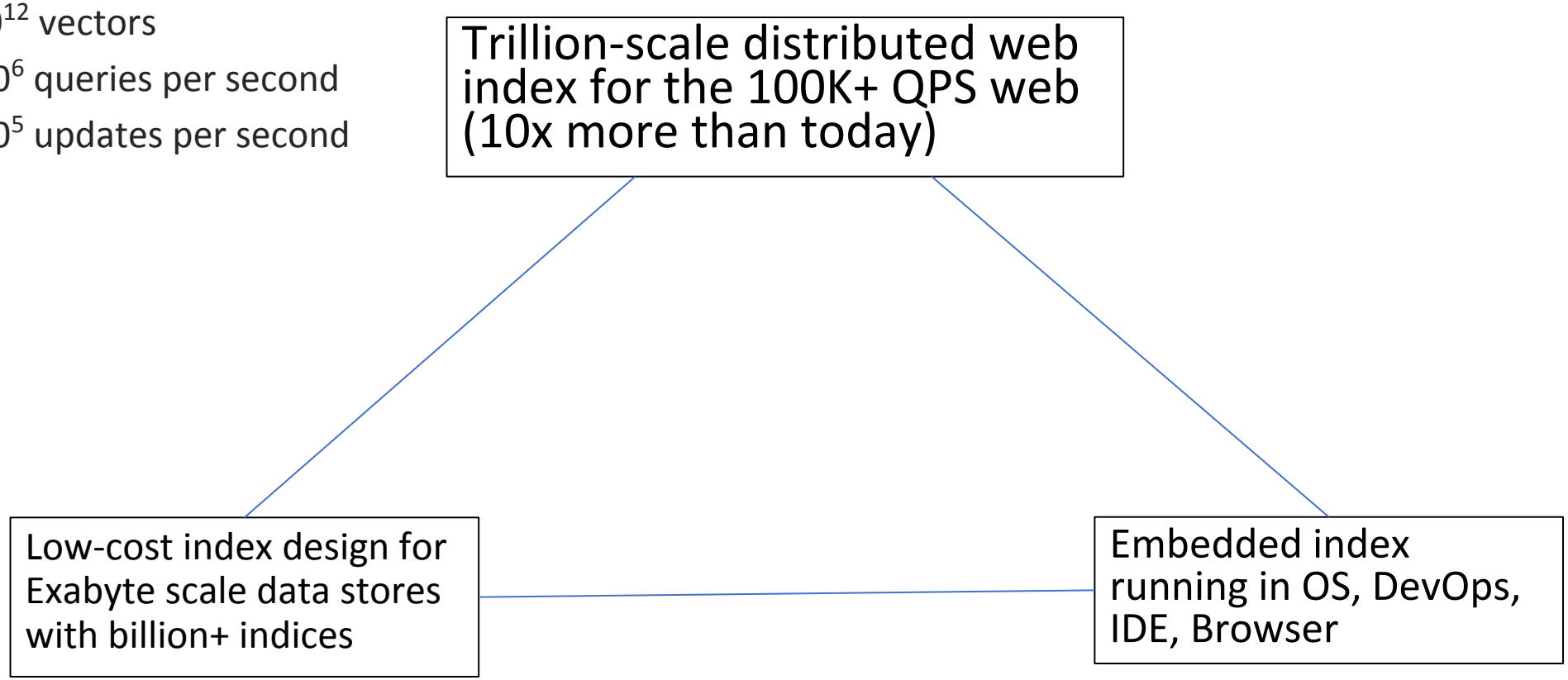
Transaction

Challenge: Vector DB designed ground up for

Use cases spanning from web, enterprise, email, OS, IDE, Browser, Github repos..

Index capabilities ranging from:

- 10^5 to 10^{12} vectors
- 10^{-5} to 10^6 queries per second
- 10^{-4} to 10^5 updates per second



https://github.com/Microsoft/DiskANN

https://big-ann-benchmarks.com

harshasi@microsoft.com