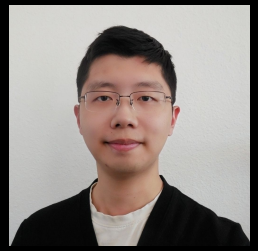
ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms

Magdalen Dobson Manohar
Zheqi Shen
Guy E. Blelloch
Laxman Dhulipala
Yan Gu
Harsha Vardhan Simhadri
Yihan Sun

Presented by Luca Musk

The Authors

Magdalen Manohar
Carnegie Mellon



Zheqi Shen
UC Riverside

Guy Blelloch
Carnegie Mellon

Laxman Dhulipala
University of Maryland

Yan Gu
UC Riverside

Harsha Simhadri
Microsoft

Yihan Sun
UC Riverside

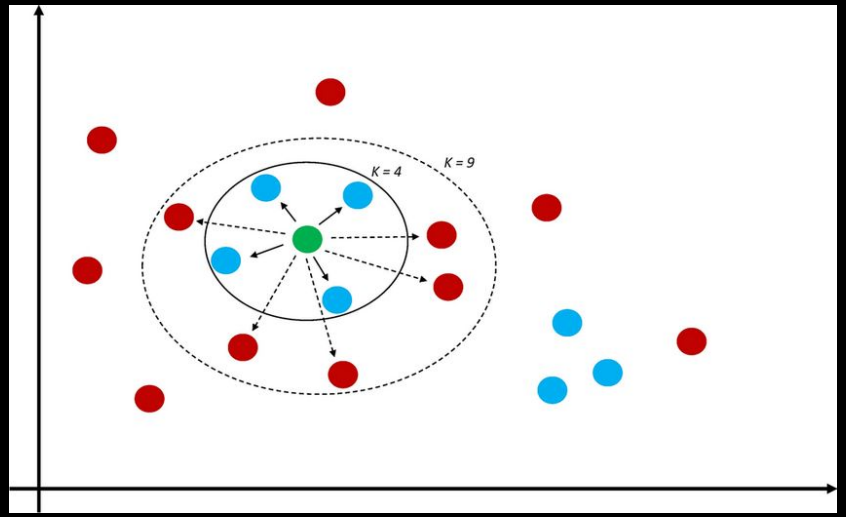
KNNs and ANNs

k-NNS

- K nearest neighbors to a point p
- Exact KNN is too slow on high dimensions

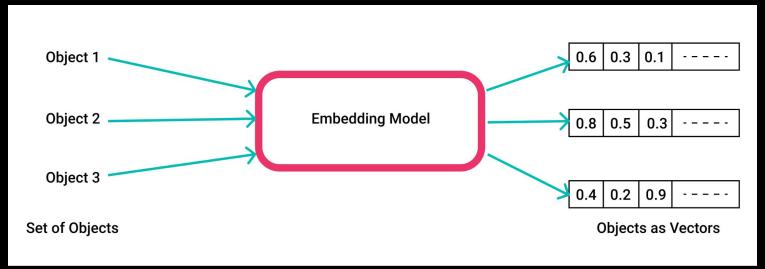
k-ANNS

- k@k′ recall: the proportion of correct answers of a k-NN algorithm if the approximate algorithm guesses k′ points
- Assume 10@10

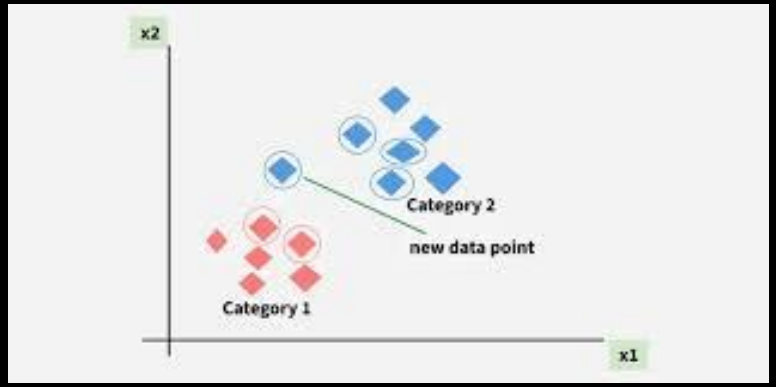


https://www.researchgate.net/figure/k-nearest-neighbors-A-diagram-showing-an-example-of-the-k-nearest-neighbor-machine_fig1_356781515

Why This Matters



https://www.pinecone.io/learn/vector-embeddings



https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/

Machine Learning and Vector Embeddings

- Convert data to high dimensional space
- Conversion should maintain semantic distance via L2 distance
- I.e. king close to queen

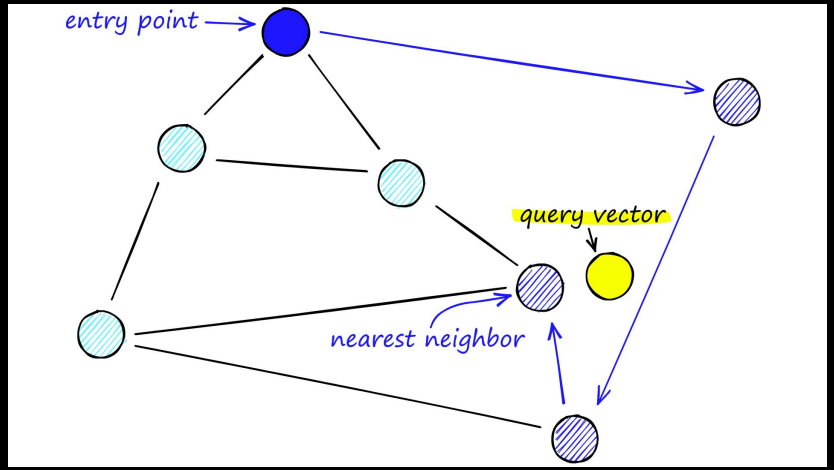
Graph-Based ANNs

Given a set of points

1. Construct proximity graph
2. Traverse from entry to target point
3. Compile KNN in process

ANNS graph

- Graph on a point set
- Most neighbors are nearby, some are far
  - Far neighbors are needed for traversal

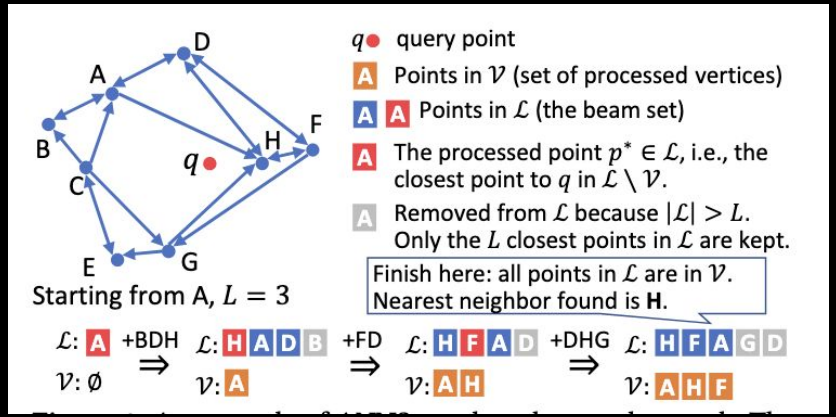


https://www.pinecone.io/learn/series/faiss/hnsw/

Techniques

Beam Search

- Beam of size L of NN candidates
- Pops closest vertex to q from L, adds out-neighbors to L
- Limit beam to L closest points to q
- Once every point in beam is visited, terminate and return k-nn in visited set (V)



Incremental Algorithms

Adds points to a graph with edges to prior points

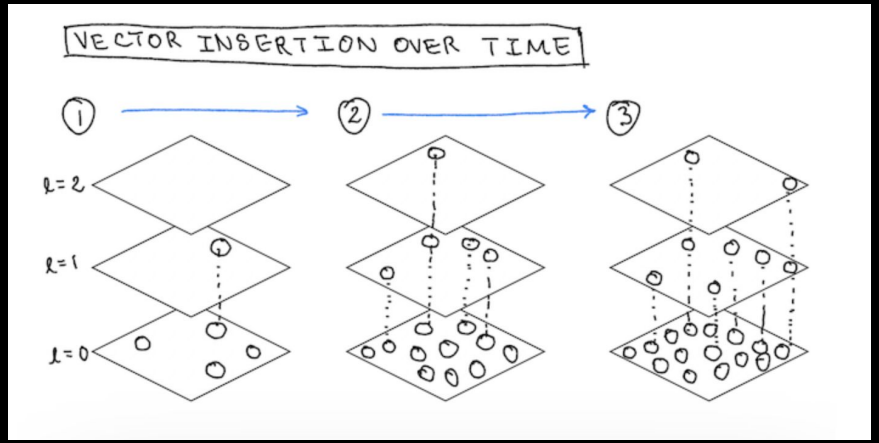
- DiskANN, HNSW

Insertion

- Neighbors are a subset of V from beam search
- Subset should cover variety of edge lengths

Problem

- All points are added in parallel, so need locks on each point
- Contention degrades performance, causes non-determinism



https://softwarediagrams.com/diagrams/hnsw-algorithm-hierarchical-navigable-small-world/

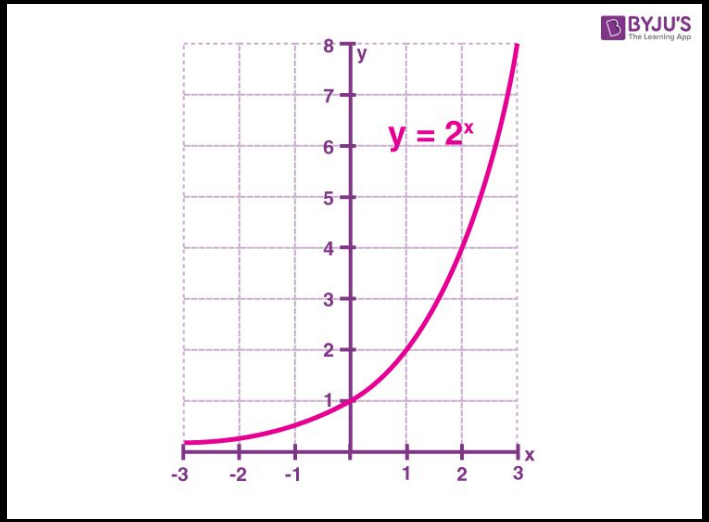
Batch Insertion

Batching

- Trade some sequentialism to remove locking
- Each batch adds to only the subgraph composed from result of inserting the prior batch

Insert in Parallel Batches with Prefix Doubling

- Exponentially increasing size of each batch
- Ensures minimal downside from batching



https://byjus.com/maths/exponential-functions/

Efficient Neighbor Reversal

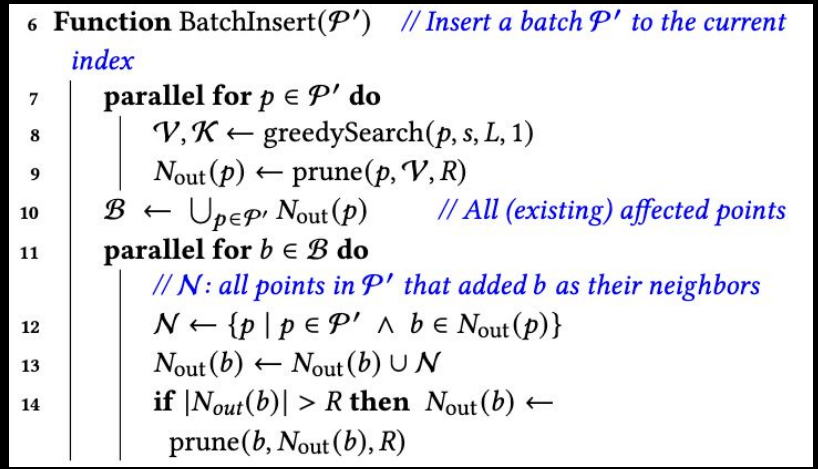
Semisort

- Group all the inserted edges from a batch together by their outgoing vertex
- Add those points to the neighbor of the outgoing vertex

Pruning

- Prune edges if the degree of a vertex gets too large

[1,2,1,1,3,4,5,7,3,3,3,7,6] -> [2,1,1,1,3,3,3,4,5,7,7,6]



Batch Size Truncation

Problem: Large Batch Sizes

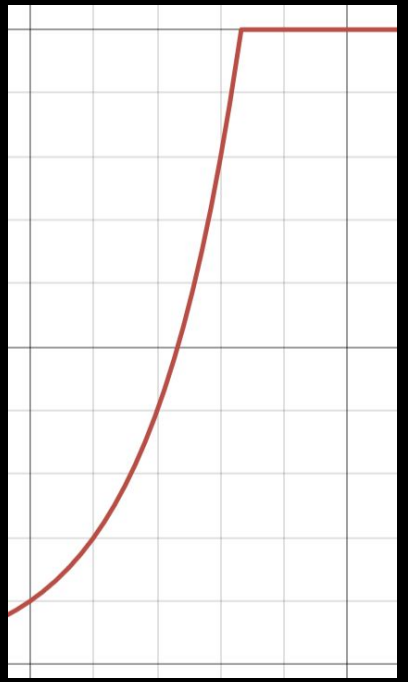
- Lose a lot of info in insertion

Idea: Bound batch size

- Parameter to set, chose 0.02n
- Doesn't hurt parallelism on large datasets

Result

- Nearly no difference compared to sequential
- No recall decrease



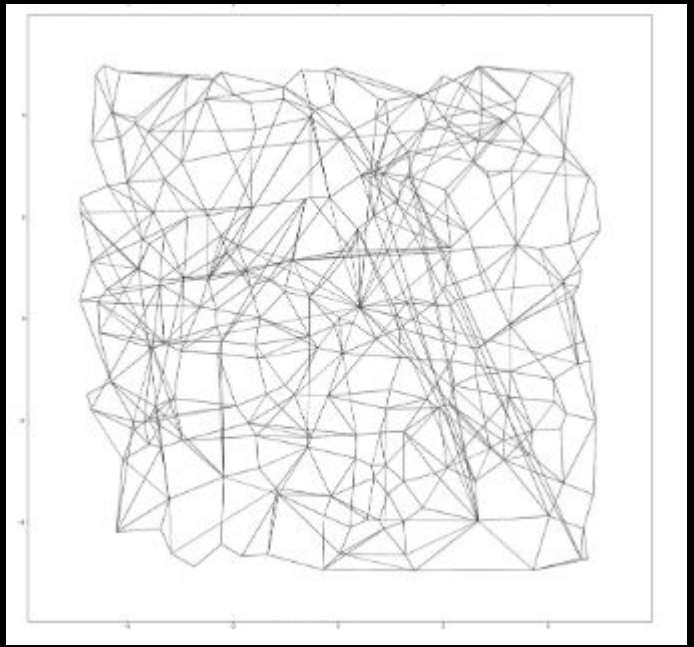
ParlayDiskANN

DiskANN

- Incremental and in-memory
- We just talked about it!

ParlayDiskANN

- Functionally the incremental technique as before
- Change: use DiskANN's robust pruning
  - Remove points that are too close the the recently added point



https://papers.nips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

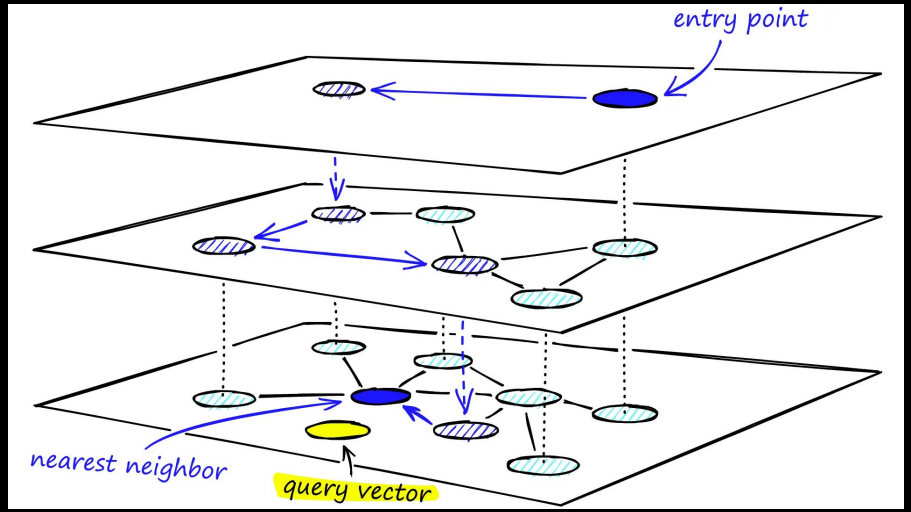
ParlayHNSW

Hierarchical Navigable Small World

- Incremental, constructs a hierarchy of NSWs
- Similar pruning to DiskANN

ParlayHNSW

- Prefix doubling, with batch insertion at each layer
- Also removes other locks in HNSW

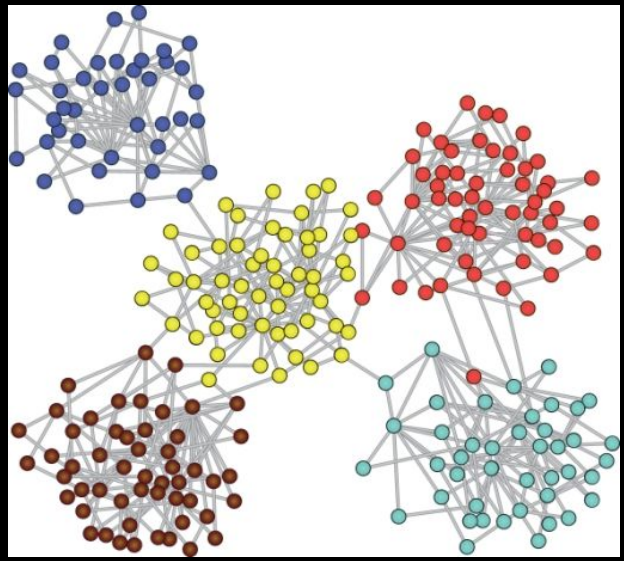


https://www.pinecone.io/learn/series/faiss/hnsw/

Clustering-Based Algorithms

Clustering Trees
- Randomly split the output recursively
- Bottom node is leaf clusters
- Local ANN graph in each leaf cluster
- Generate multiple cluster trees
  - take the union of the ANN graph for final result

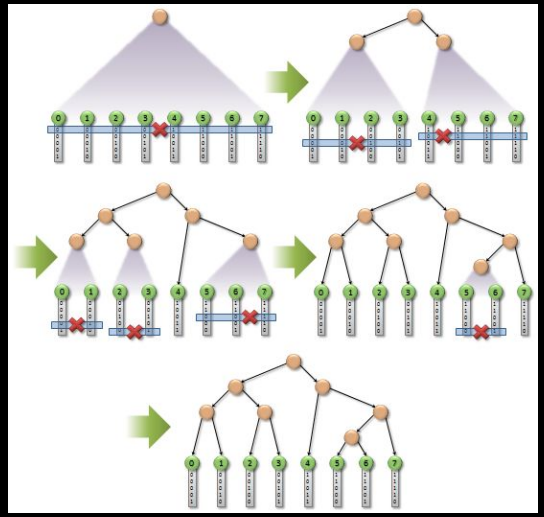
Problems
- Trees are constructed sequentially
- Contention in ANN merging
- Local ANN construction can be costly



https://github.com/benedekrozemberczki/awesome-community-detection

Parallelizing Cluster-Based Algorithms

1. Parallelize each tree
   a. Already done in literature
   b. Can only scale to optimal tree count (10-30)
2. Parallelize each branch
   a. Partition points in parallel
3. Avoiding Point Locks
   a. Also use a semisort!
   b. Groups a point's edges to build the graph easily



https://developer.nvidia.com/blog/thinking-parallel-part-iii-tree-construction-gpu/

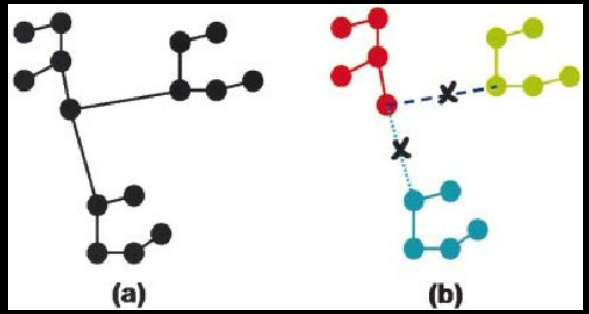
ParlayHCNNG

Hierarchical Clustering-Based Nearest Neighbor Graph

- Clustering-based
- Randomly select a pair of points
  - Partition on whether closer to one or the other
- Local ANN is a degree bound MST
- Can be implemented simply using prior techniques

Naive Problem: High Memory Usage

- MST is on entire wholly connected leaf cluster, overwhelms L3
- SOLN: Edge-restrict edges in leaf cluster to nearby points, then do MST

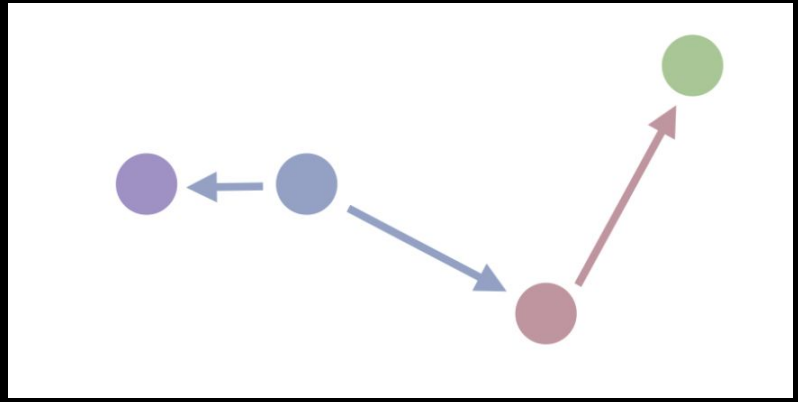


https://www.researchgate.net/figure/Basic-idea-of-MST-based-clustering-a-Data-representation-using-MST-Each-node_fig1_9087685

ParlayPyNNDescent

PyNNDescent
- Clustering + Refinement
- Local ANN is KNN
- Add the two hop undirected neighborhood of p, then prune

ParlayPyNNDescent
- Two hop undirected neighborhood is hard, so limit the degree
- Process two hop neighborhoods in batches
- Still too much memory to reach billions

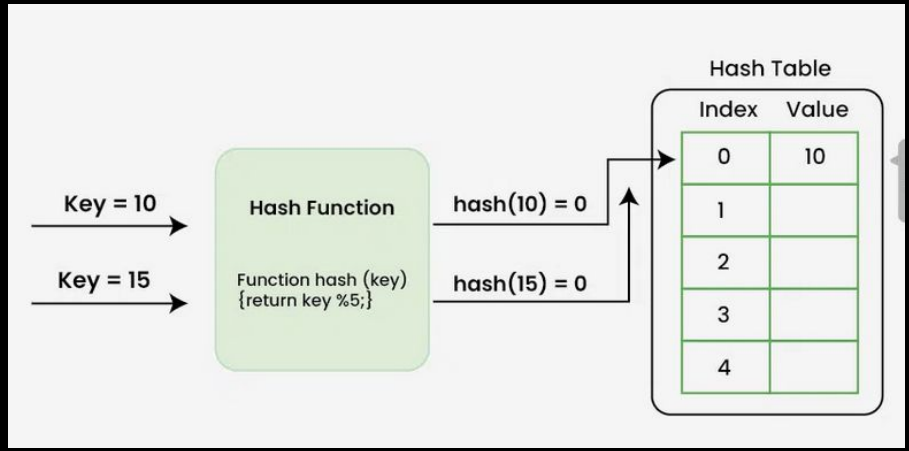


https://pynndescent.readthedocs.io/en/stable/how_pynndescent_works.html

Some Final Search Optimizations

1. Hash table for Visited Set
   a. Approximate, simply remove if a collision happens
   b. The table's size is L^2, so fits in L1 cache

2. 1+e Pruning
   a. Don't search vertices that are more than 1+e in distance from current kth NN

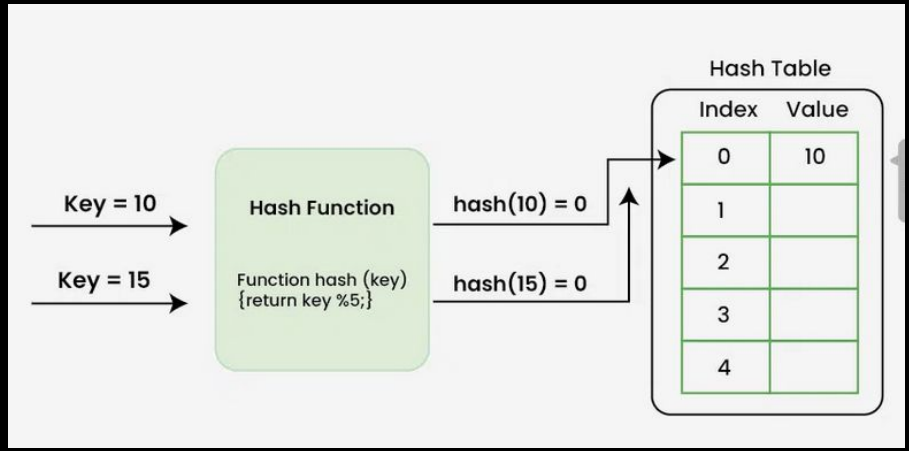


https://www.geeksforgeeks.org/dsa/hash-table-data-structure/

Some Final Search Optimizations

1. Hash table for Visited Set
   a. Approximate, simply remove if a collision happens
   b. The table's size is L^2, so fits in L1 cache

2. 1+e Pruning
   a. Don't search vertices that are more than 1+e in distance from current kth NN



https://www.geeksforgeeks.org/dsa/hash-table-data-structure/

Experiments

Datasets

- 3 Billion point datasets
- 2 of images, 1 of web documents

Baselines

- DiskANN, HSNW, HCNNG, ParlaryNNDescent
- Compare on 1M point set
- Added parallelization to HNSW
- Also compared to non-graph examples, FAISS and FALCONN

	BIGANN	MSSPACEV	TEXT2IMAGE
DiskANN	.42	.35	.70
HNSW	.35	.37	.94
HCNNG	.45	.77	1.75
pyNNDescent	.42	.73	1.23
FAISS	.19	.13	.22

Table 1. Build times (hours) on hundred million scale datasets.

Single Thread Results

Matches Baselines

- On 1M result, parlay versions are equivalent
- Substantially outpace non-graph versions
  - So much so that FALCONN will not be reported

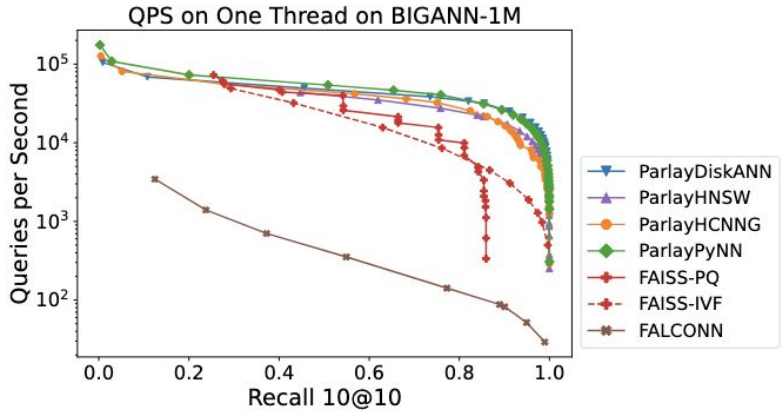


Figure 5. QPS on a single thread on BIGANN-1M. Shown to compare with ANN-benchmarks.

Parallel Results

Great Scalability

- DiskANN: 1.2x Improvement
  - Due to no lock contention at high thread count
- HSNW: 1.4x Improvement
  - Also due to lock contention
- HCNNG: 12x Improvement
  - Adding more parallelism
  - Faster single threaded too
- PyNNDescent: 28x Improvement
  - Added parallelism

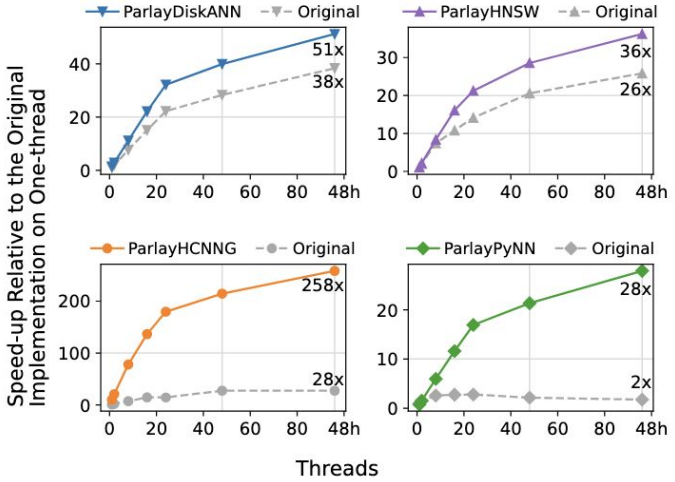
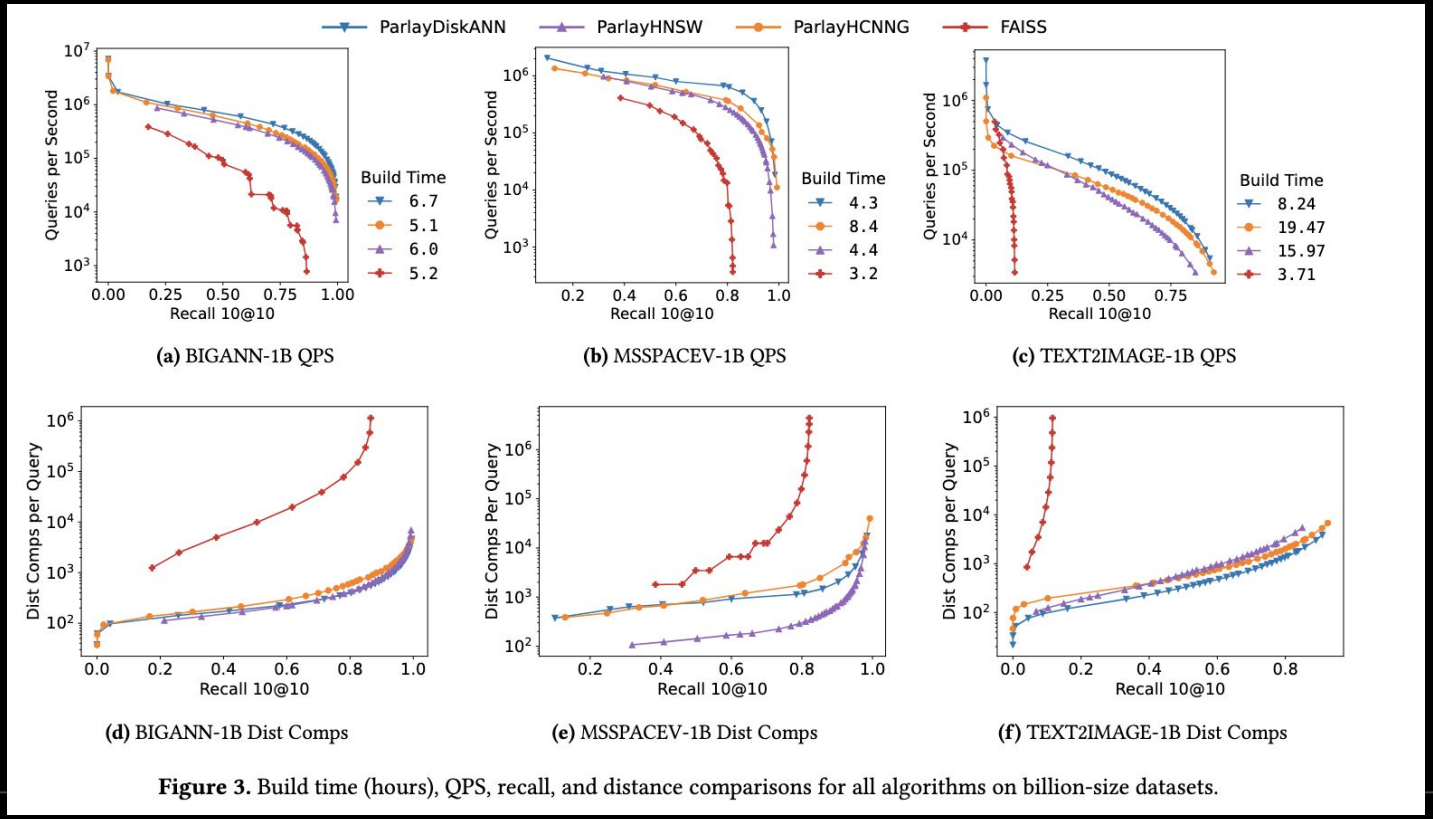


Figure 1. Scalability of original and our new implementations of four ANNS algorithms on various number of threads. Within each subfigure, all numbers are *speedup numbers relative to the original implementation on one thread*. Higher is better. Results were tested on a machine with 48 cores using dataset BIGANN-1M (10^6 points). "48h": 48 cores with hyperthreads. The two implementations in the same subfigure always use the same parameters and give similar query quality (recall-QPS curve).

1 Billion Points



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

(d) BIGANN-1B Dist Comps **(e)** MSSPACEV-1B Dist Comps **(f)** TEXT2IMAGE-1B Dist Comps

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

Strengths and Weaknesses

Strengths

- Strong results
- Multiple algorithms and algorithm types illustrate the library's flexibility
- Comprehensive experiments

Weaknesses

- No DiskANN billion point comparison
  - Would take a long time to build, but the DiskANN paper has some results
- Poor build times compared to non-graph algorithms

Discussion Questions

1. Where would one prefer build times to a faster QPS?
    a. Are those situations common enough to prefer non-graph approaches?
2. Can this parallelization strategy extend to GPUs and more complicated memory settings?
3. Does high-dimensional nearest neighbors algorithms have promise over neural networks? In what situations would we prefer KNNs?