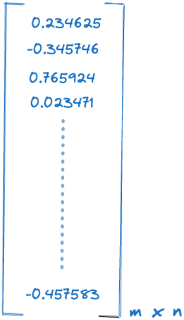


Vector Search

Jianguo Lu

November 15, 2023

Outline

Vector index in Lucene

- Lucene 9.0 (2021) added support for dense vector indexes and approximate k-NN search.
- Takes advantage of the HNSW algorithm.
- Lucene is all you need[1]

[1] Jimmy Lin et al. *Vector Search with OpenAI Embeddings: Lucene Is All You Need*. 2023. arXiv: 2308.14963 [cs.IR].

Vector search in Lucene

- KnnVectorField

```
document.addField(new KnnVectorField("field", float[] vector))
```

- KnnVectorQuery

```
indexSearcher.search(new KnnVectorQuery("field", float[] vector,  int
    topK)
```

Luence vector search

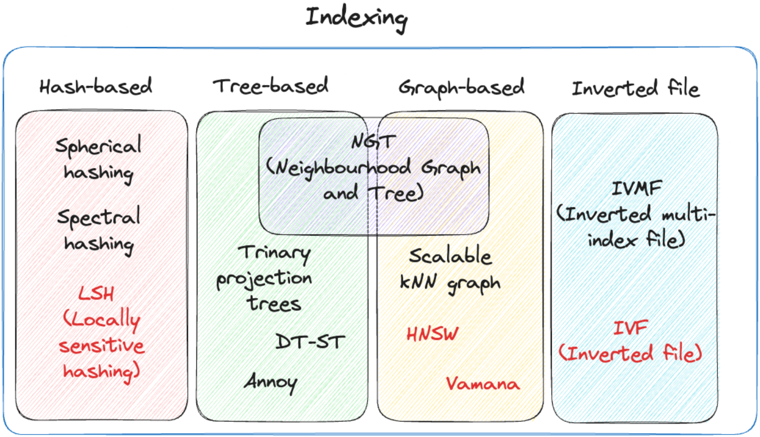
```
public static void main(String[] args) throws Exception {
    Directory directory = FSDirectory.open(Paths.get("my_index_knn"));
    StandardAnalyzer analyzer = new StandardAnalyzer();
    IndexWriterConfig indexWriterConfig = new
        IndexWriterConfig(analyzer);
    indexWriterConfig.setUseCompoundFile(false);
    IndexWriter indexWriter = new IndexWriter(directory,
        indexWriterConfig);

    for (int i = 1; i <= 500; i++) {
        Document document = new Document();
        document.add(new KnnVectorField("vector1",
            TestDataGenerator.generateData(128),
            VectorSimilarityFunction.EUCLIDEAN));
        document.add(new KnnVectorField("vector2",
            TestDataGenerator.generateData(128),
            VectorSimilarityFunction.EUCLIDEAN));
        indexWriter.addDocument(document);
        indexWriter.flush();
        indexWriter.commit();
    }
    indexWriter.flush();
    indexWriter.commit();
    IndexReader reader = DirectoryReader.open(indexWriter);
    IndexSearcher searcher = new IndexSearcher(reader);
    KnnVectorQuery knnVectorQuery = new KnnVectorQuery("vector1",
        TestDataGenerator.generateData(128), 10);
    TopDocs search = searcher.search(knnVectorQuery, 10);
```

FAISS (Facebook AI Similarity Search)

```
!pip install faiss-cpu

import faiss
import numpy as np

index = faiss.IndexIDMap(faiss.IndexFlatIP(768))
index.add_with_ids(embeddings, np.arange(len(data)))

import faiss
index = faiss.IndexFlatL2(d)
print(index.is_trained)
index.add(xb)
print(index.ntotal)
```

FAISS (Facebook AI Similarity Search)

IndexFlatL2 : brute-force L2 distance search on them
IndexIVF : Inverted File Index.
IndexHNSW : Hierarchical Navigable Small World.
IndexLSH : Locality Sensitive Hashing
... ...



Search Example in FAISS

```
k = 4                          # we want to see 4 nearest neighbors
D, I = index.search(xb[:5], k) # sanity check
print(I)
print(D)
D, I = index.search(xq, k)     # actual search
print(I[:5])                   # neighbors of the 5 first queries
print(I[-5:])                  # neighbors of the 5 last queries
```

Outline

Inverted File Index Example

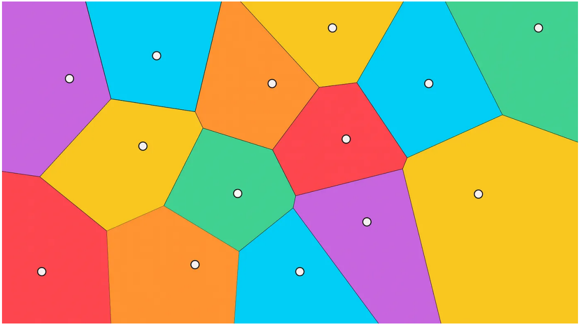
```
import faiss

d = 64  # Dimension of the vectors
nlist = 100  # Number of clusters
quantizer = faiss.IndexFlatL2(d)  # Quantizer (flat index)
index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC_L2)

xb = ...  # Your dataset
index.train(xb)
index.add(xb)

k = 5  # Number of nearest neighbors
xq = ...  # Your query vector
D, I = index.search(xq, k)
print("Nearest neighbors:", I)
```

Inverted File Index

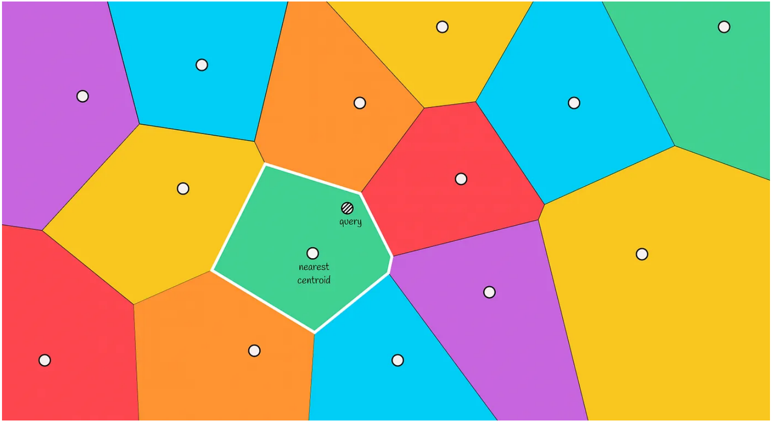
- `Flat` indicates that there is no decompression of original vectors and they are fully stored.
- To create this index, we first need to pass a quantizer — an object that will determine how database vectors will be stored and compared.
- Two parameters:
  - `nlist`: defines a number of regions (Voronoi cells) to create during training.
  - `nprobe`: determines how many regions to take for the search of candidates.

Voronoi graph

- Create several non-intersecting regions
- Each region has its own centroid (white dots)
- Distance from a centroid to any point of its region is less than the distance from that point to another centroid.



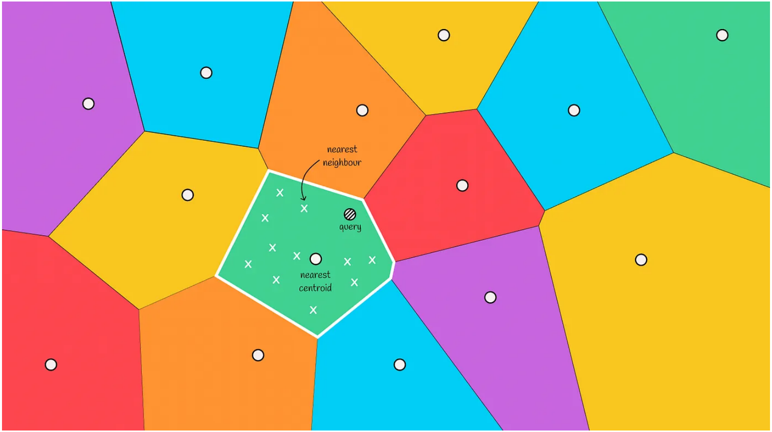
Search



- When given a query, distances to all the centroids of Voronoi partitions are calculated.
- The centroid with the lowest distance is chosen and vectors contained in this partition are then taken as candidates.
- Faster than brute force search

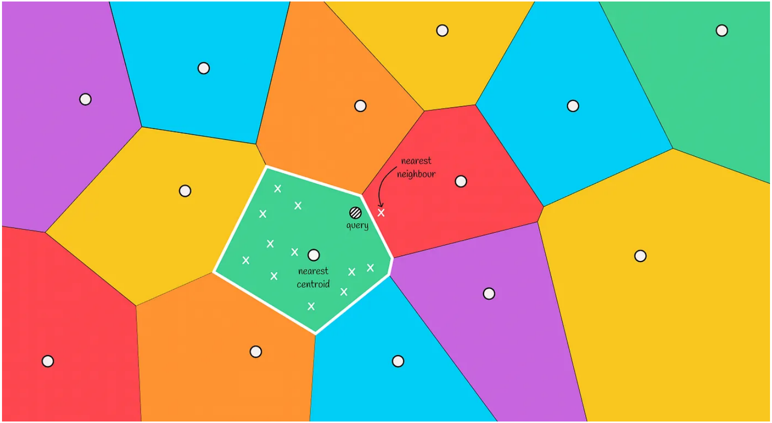
Edge problem

- IVF does not guarantee that the found object will always be the nearest.
- Example: the actual nearest neighbour is located in the red region but we are selecting candidates only from the green zone. Such a situation is called the edge problem.

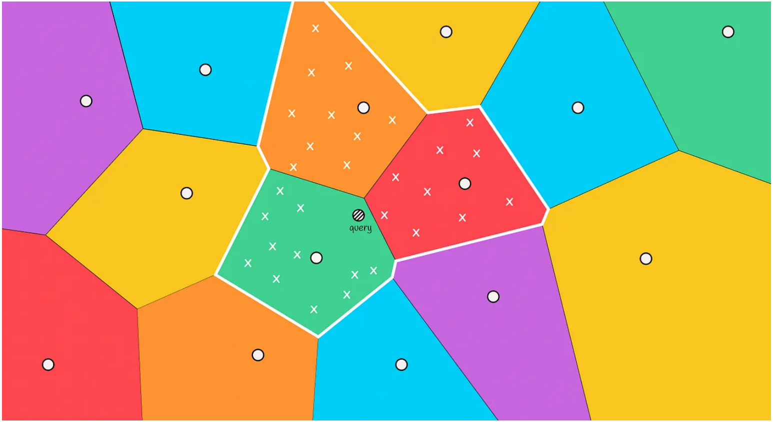


Solution

- This case typically occurs when the queried object is near the border.
- Solution: choose several regions to search for candidates based on the top m closest centroids to the object.



Now we search in three cells.



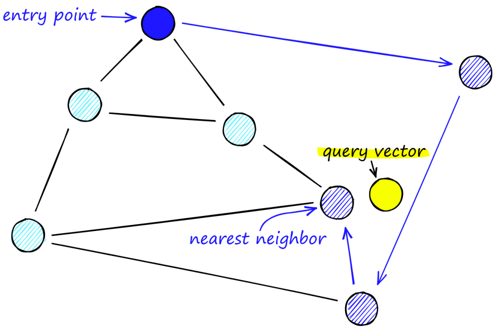
nprobe=3.

Outline

Code for HNSW

```
# Example: Create an IndexHNSW
d = 64  # Dimension of the vectors
index = faiss.IndexHNSWFlat(d, 32, faiss.METRIC_L2)

# Train on a dataset
xb = ...  # Your dataset
index.train(xb)

# Add vectors to the index
index.add(xb)

# Perform a search
k = 5  # Number of nearest neighbors
xq = ...  # Your query vector
D, I = index.search(xq, k)
print("Nearest neighbors:", I)
```

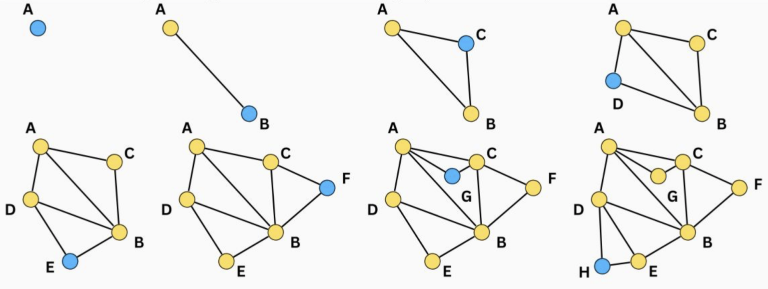
HNSW[2]



[2] Yu A Malkov and Dmitry 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 (2018), pp. 824–836.

Navigable Small World (NSW)

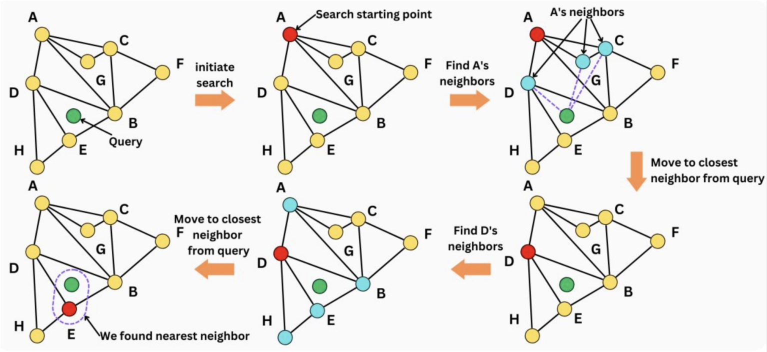
- Inserting data points one by one.
- When a new node is inserted, it is then linked by edges to the M nearest vertices to it.
- Long-range edges will likely be created at the beginning phase of the graph
- With more nodes added, newly connected edges will be smaller.



Search the graph

When we want to find the nearest neighbor to a query vector,

- initiate the search by starting at one node (i.e. node A in that case).
- Among its neighbors (D, G, C), look for the closest node to the query (D).
- We iterate over that process until there are no closer neighbors to the query.
- Once we cannot move anymore, we found a close neighbor to the query.



Problem 1: Accuracy

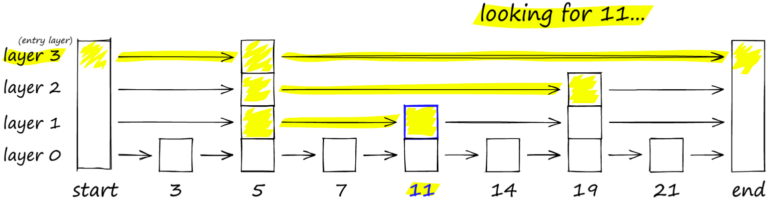
Problem : The search is approximate and the found node may not be the closest as the algorithm may be stuck in a local minima.

Solution : The search accuracy can be improved by using several entry points.

Problem 2: Speed

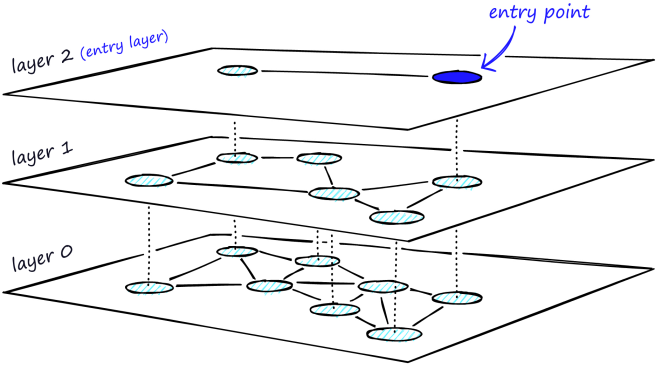
spend a lot of iterations traversing the graph to arrive at the right node.

- Recall the skiplist algorithm
- There are layers of links
- Upper layers have longer jumps



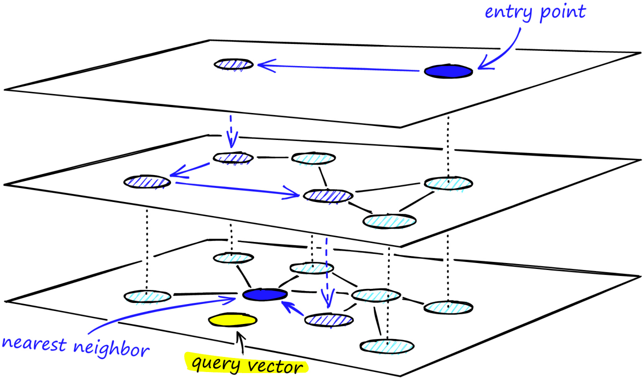
Hierarchical NSW

- Upper layers have less nodes/less degrees/less dense
- Layer 0 have all the nodes
- include a node in the graph at layer L with a probability
- The first layer allows us to traverse longer distances at each iteration
- In last layer, each iteration will tend to capture shorter distances.



Search process

- Search starts from the top layer where the distance is longer
- go to the next layer if the NSW algorithm finds the closest neighbor in that layer.

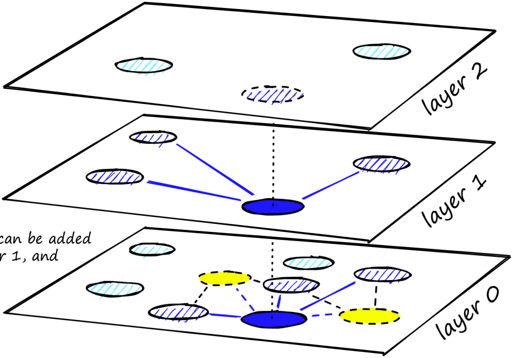


insert vector
at layer 1

with M = 3
layer 1 and 0
find 3 links

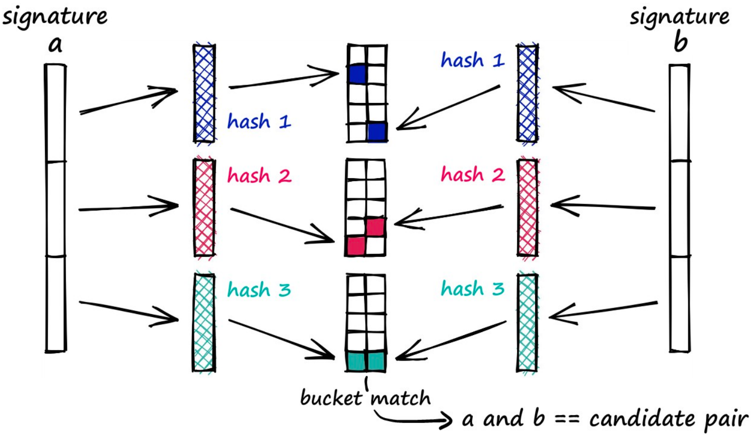
as more vertices are inserted, more links can be added - up to M_{max} for layer 1, and M_{max0} for layer 0

$M_{max} = 3$
$M_{max0} = 5$



Outline

LSH



- What is the signature?
- How to obtain the signature?