Approximate Nearest Neighbours

IN3120

Based on this and this
See also this survey paper

Overview

What we'd like:

- We'd like to do *k*-NN lookups in high-dimensional spaces
- We'd like to do this efficiently when we have a gazillion data points
- We might be willing to do it approximately, and sacrifice exactness for efficiency

What are some of the strategies we can apply?

- Brute force
- Tree-based algorithms
- Locality-sensitive hashing
- Quantization or clustering-based algorithms
- Graph-based algorithms

When should we choose what?

- Rules of thumb that depend on application requirements, dimensionality, and volume

Voronoi Diagrams

Voronoi diagram

Article Talk

From Wikipedia, the free encyclopedia

In mathematics, a **Voronoi diagram** is a partition of a plane into regions close to each of a given set of objects. In the simplest case, these objects are just finitely many points in the plane (called seeds, sites, or generators). For each seed there is a corresponding region, called a **Voronoi cell**, consisting of all points of the plane closer to that seed than to any other. The Voronoi diagram of a set of points is dual to that set's Delaunay triangulation.

The Voronoi diagram is named after mathematician Georgy Voronoy, and is also called a **Voronoi tessellation**, a **Voronoi decomposition**, a **Voronoi partition**, or a **Dirichlet tessellation** (after Peter Gustav Lejeune Dirichlet). Voronoi cells are also known as **Thiessen polygons**.[1][2][3] Voronoi diagrams have practical and theoretical applications in many fields, mainly in science and technology, but also in visual art.[4][5]

Illustration [edit]

As a simple illustration, consider a group of shops in a city. Suppose we want to estimate the number of customers of a given shop. With all else being equal (price, products, quality of service, etc.), it is reasonable to assume that customers choose their preferred shop simply by distance considerations: they will go to the shop located nearest to them. In this case the Voronoi cell R_k of a given shop P_k can be used for giving a rough estimate on the number of potential customers going to this shop (which is modeled by a point in our city).

For most cities, the distance between points can be measured using the familiar Euclidean distance:

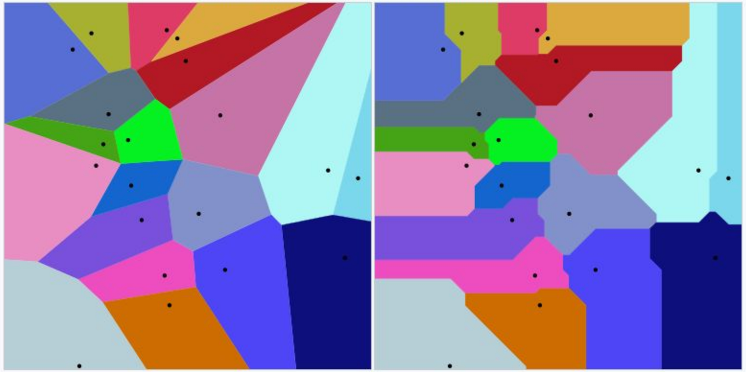
$$\ell_2 = d\left[(a_1, a_2), (b_1, b_2)\right] = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

or the Manhattan distance:

$$d\left[(a_1, a_2), (b_1, b_2)\right] = |a_1 - b_1| + |a_2 - b_2|.$$

The corresponding Voronoi diagrams look different for different distance metrics.

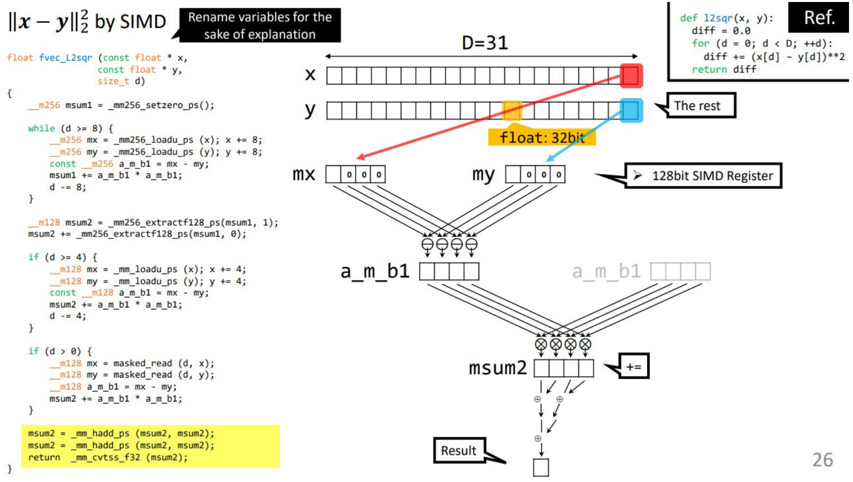
Voronoi diagrams of 20 points under two different metrics

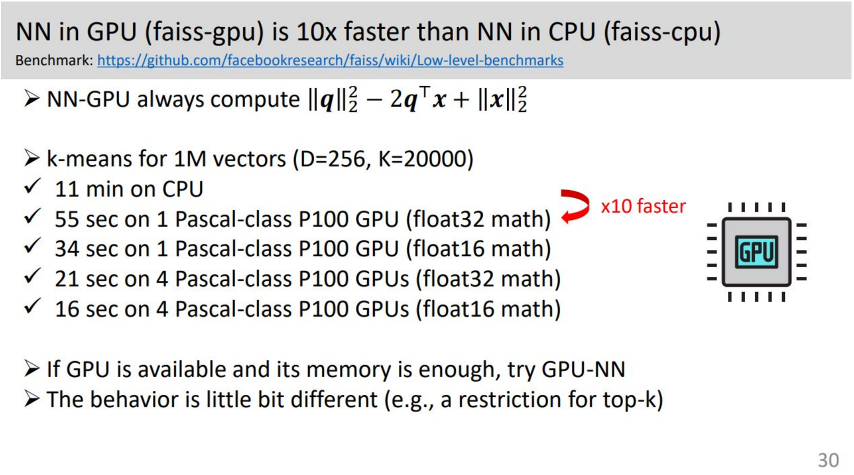


Euclidean distance Manhattan distance

Brute Force

- Do a full scan and get exact results
- Optimized implementations make heavy use of SIMD instructions and GPUs
- Application requirements, dimensionality and volume dictates if this is feasible





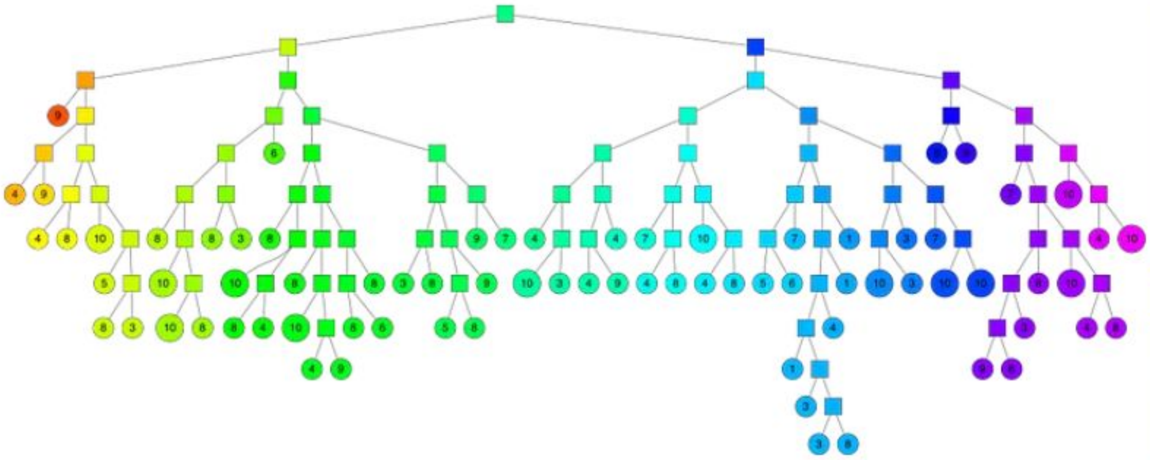
Tree-Based Algorithms

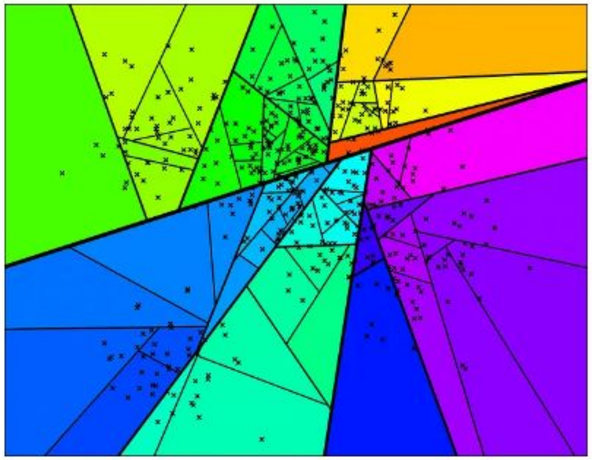
Indexing:

- Construct a tree by recursively partitioning the data, until the leaf nodes hold “few enough” data points
- Construct a forest of trees

Querying:

- Start at the query point
- Select one or more trees, follow the branches
- Consider the points located in the regions you end up





See also this tutorial.

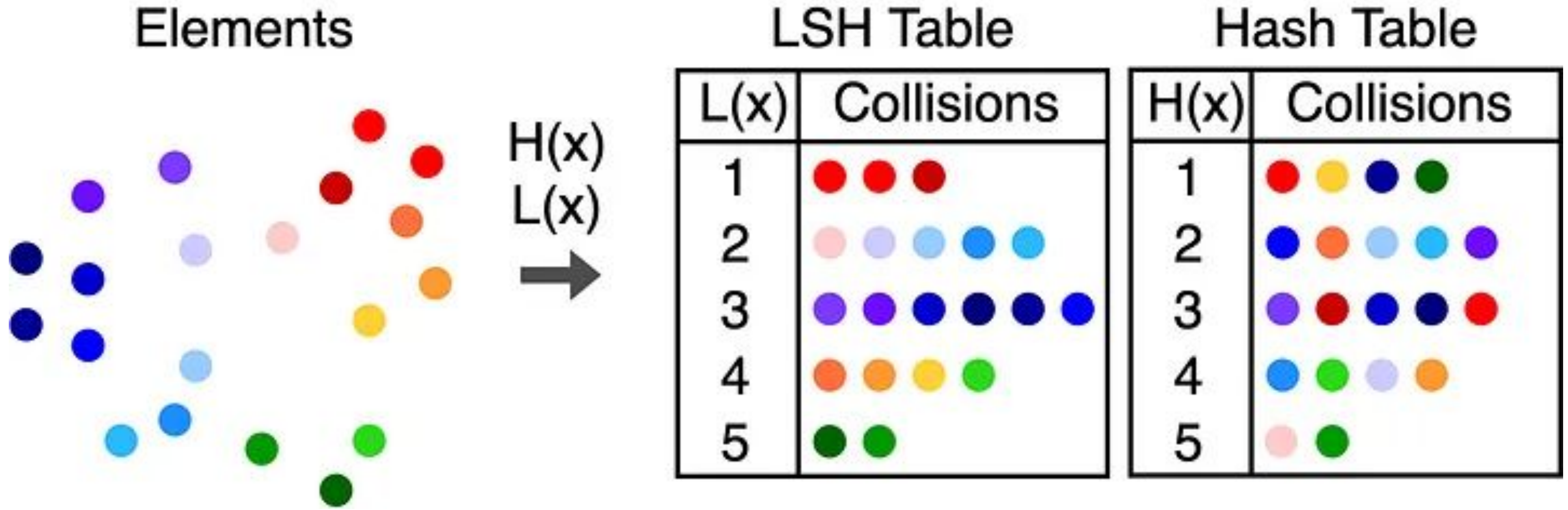
Locality-Sensitive Hashing

Indexing:

- Apply multiple hash functions *h* to each point to bucket them
- Distance $d(x, y)$ is small → $\Pr(h(x) = h(y))$ is high

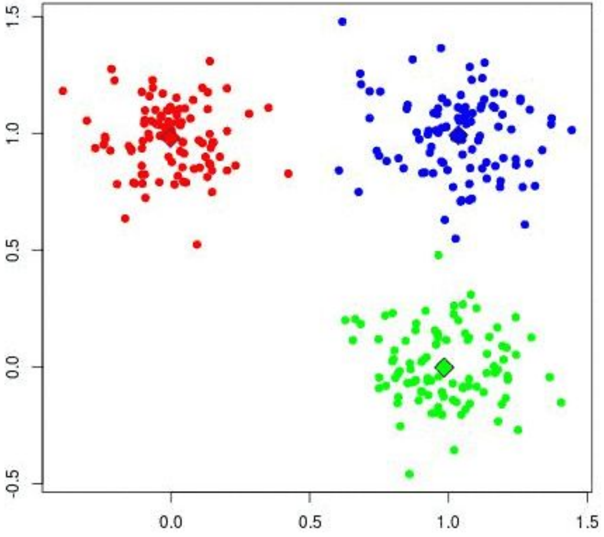
Querying:

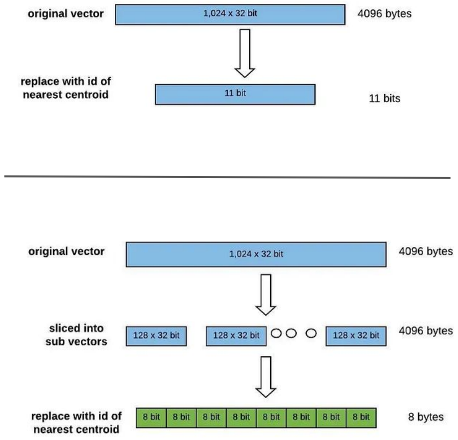
- Apply the hashes to the query point
- Consider the points in the buckets we hash to

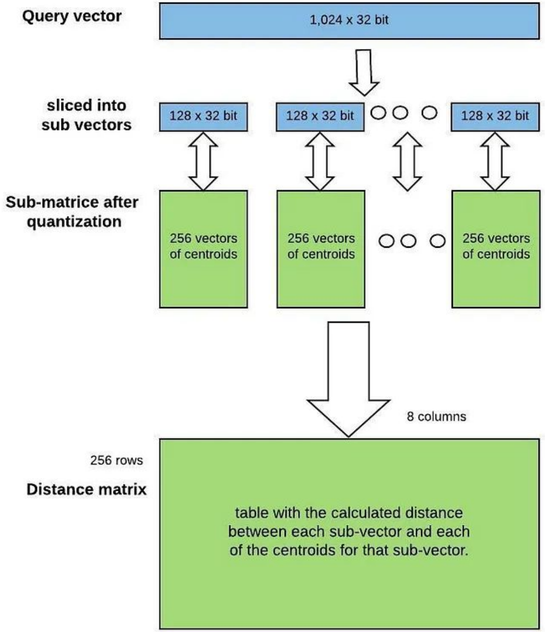


Quantization or Clustering-Based Algorithms

- Recode (cluster) the vectors to reduce the size of the dataset
- Replace each vector with a leaner, approximate and quantized representation (clusters)
- Can be combined with an inverted index







See also this tutorial.

Graph-Based Methods

Hierarchical Navigable Small World Graphs

The intuition of this method is as follows, in order to reduce the search time on a graph we would want our graph to have an average path.

This is strongly connected to the famous *"six handshake rule"* statement.

> "There is at most 6 degrees of separation between you and anyone else on Earth." — Frigyes Karinthy

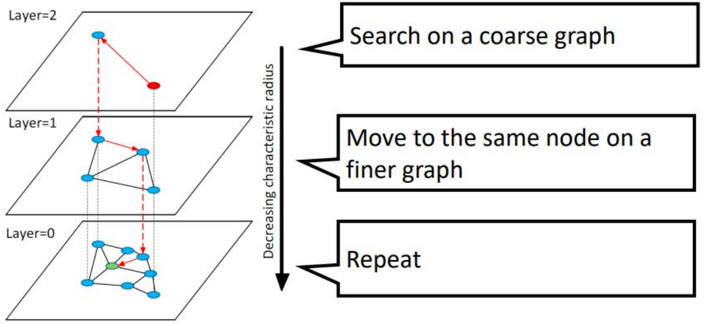
Many real-world graphs on average are highly clustered and tend to have nodes that are close to each other which are formally called small-world graph:

- highly transitive (community structure) it's often hierarchical.
- small average distance ~log(N).

In order to search, we start at some entry point and iteratively traverse the graph. At each step of the traversal, the algorithm examines the distances from a query to the neighbors of a current base node and then selects as the next base node the adjacent node that minimizes the distance, while constantly keeping track of the best-discovered neighbors. The search is terminated when some stopping condition is met.

Extension: Hierarchical NSW; HNSW

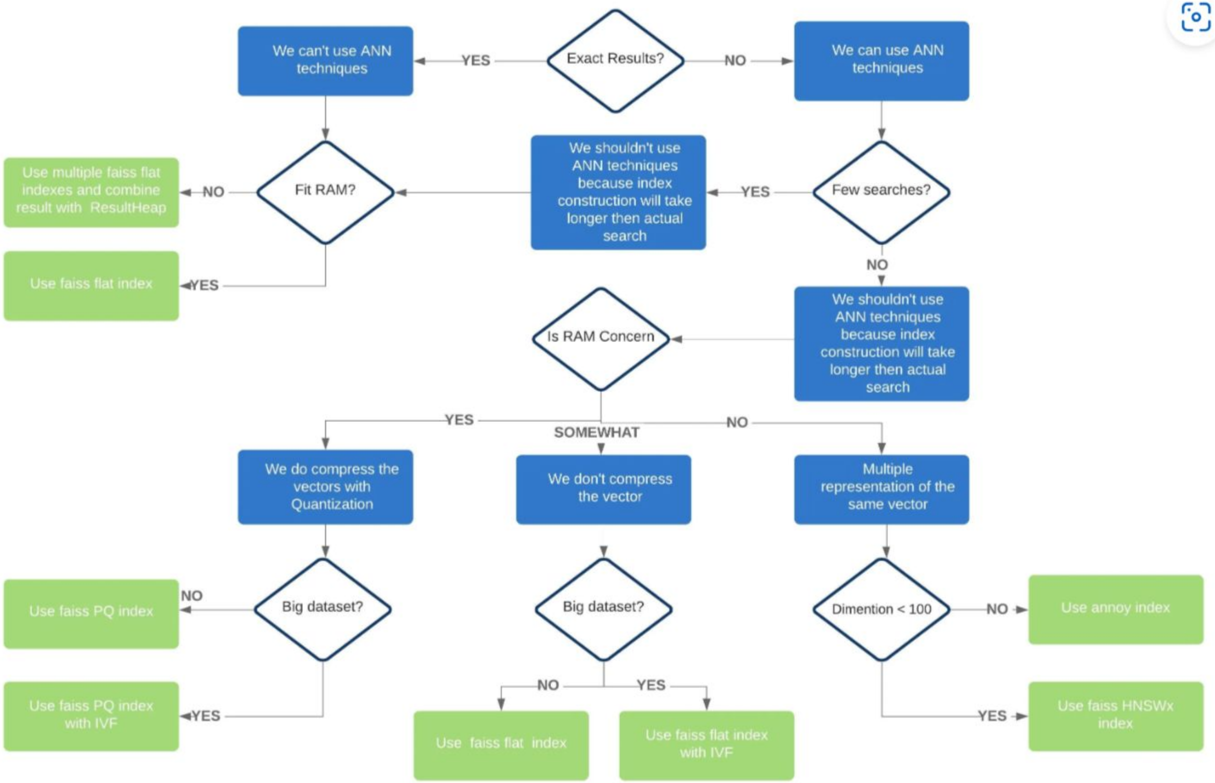
- Construct the graph hierarchically [Malkov and Yashunin, TPAMI, 2019]
- This structure works pretty well for real-world data



[Malkov and Yashunin, TPAMI, 2019]

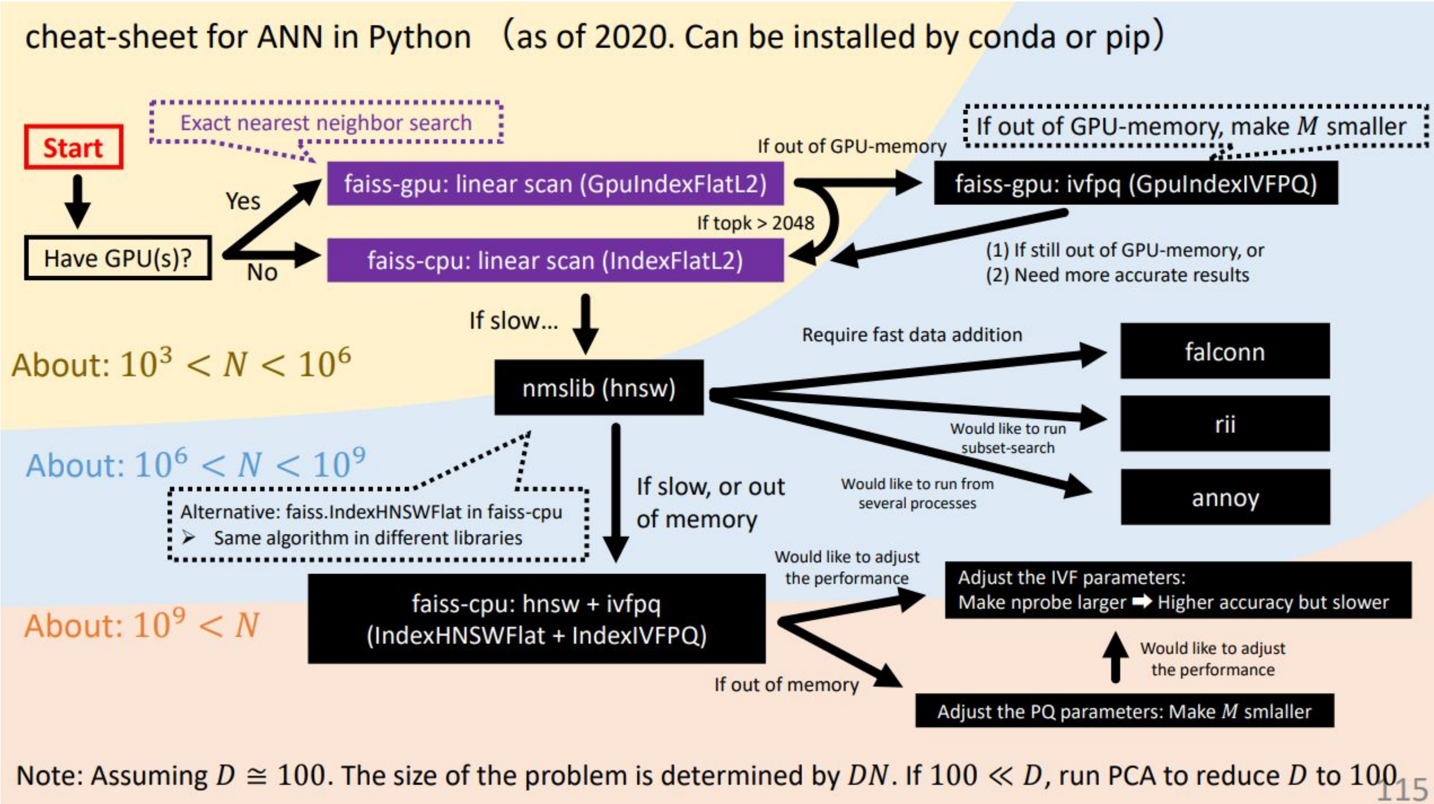
See also this tutorial.

Selection



Exact Results?
YES
NO
We can't use ANN techniques
We can use ANN techniques
Fit RAM?
Few searches?
We shouldn't use ANN techniques because index construction will take longer then actual search
Use multiple faiss flat indexes and combine result with ResultHeap
Use faiss flat index
Is RAM Concern
We shouldn't use ANN techniques because index construction will take longer then actual search
SOMEWHAT
We do compress the vectors with Quantization
We don't compress the vector
Multiple representation of the same vector
Big dataset?
Big dataset?
Dimention < 100
Use faiss PQ index
Use faiss PQ index with IVF
Use faiss flat index
Use faiss flat index with IVF
Use annoy index
Use faiss HNSWx index

Selection, cont.



Selection, cont.