HNSW

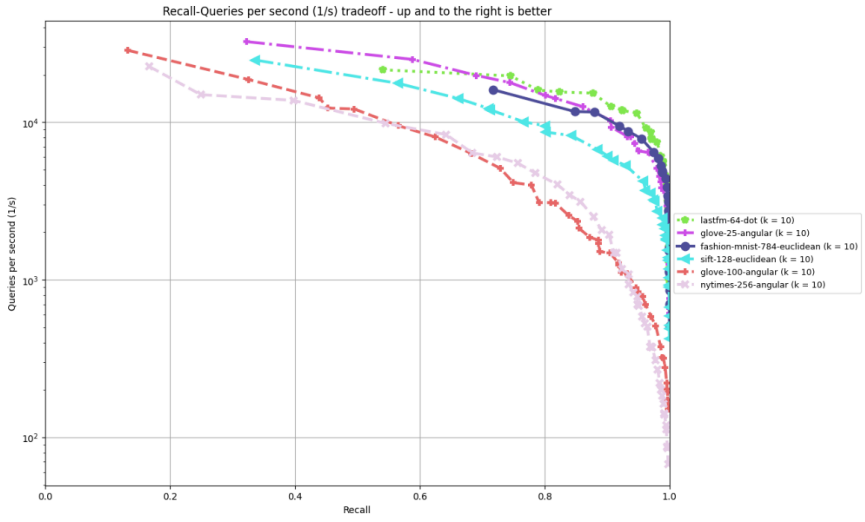
最鄰近搜索 ANN

What is similarity search:

https://www.pinecone.io/learn/what-is-similarity-search/

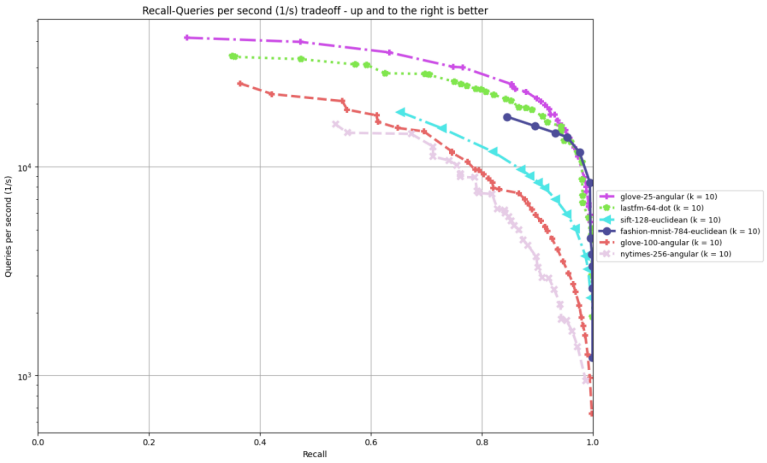
▼ **Top Algorithm**

- mslib: HNSWliB - **HNSW (Hierarchcal Navigable Small World graphs)**
  - 2016
  - Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs
  - https://arxiv.org/ftp/arxiv/papers/1603/1603.09320.pdf
  - https://github.com/nmslib/hnswlib

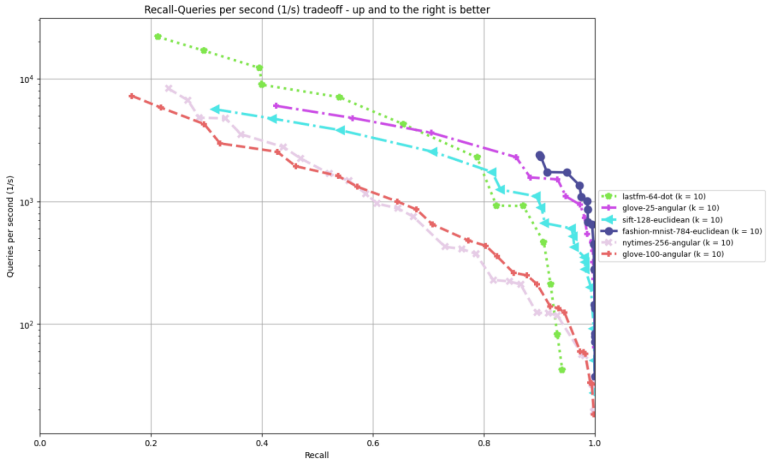


- Google's: ScaNN （Scalable Nearest Neighbors）
  - 2020
  - Accelerating Large-Scale Inference with Anisotropic Vector Quantization
  - https://arxiv.org/pdf/1908.10396.pdf

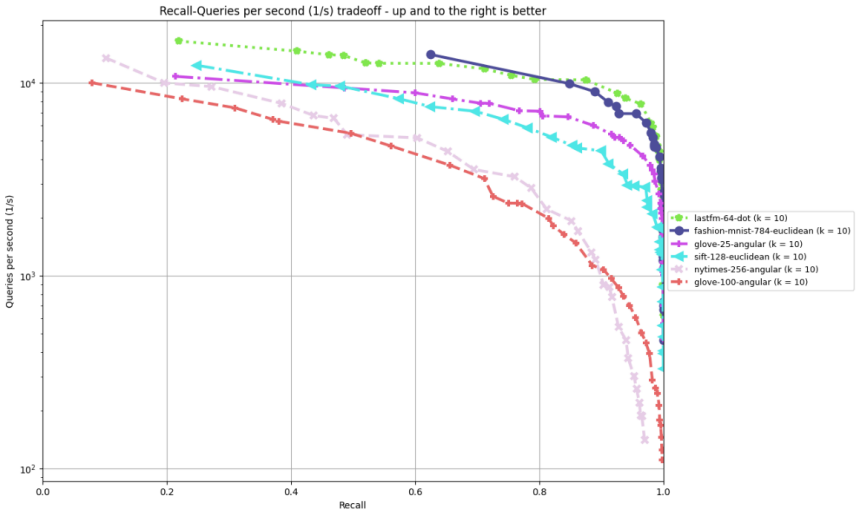
- https://github.com/google-research/google-research/tree/master/scann



- Spotify's: ANNOY （Approximate Nearest Neighbors Oh Yeah）
  - https://github.com/spotify/annoy



- Facebook's: Faiss （Billion-scale similarity search with GPUs）
  - 2017
  - Billion-scale similarity search with GPUs
  - https://arxiv.org/abs/1702.08734
  - https://github.com/facebookresearch/faiss



▼ **Bench marks compare**

running all benchmarks on a r5.4xlarge machine on AWS with `--parallelism 7`

http://ann-benchmarks.com/index.html

https://github.com/erikbern/ann-benchmarks

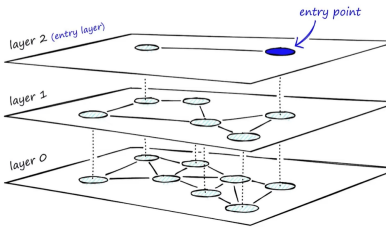
https://cloud.tencent.com/developer/article/2041936

▼ **HNSW**

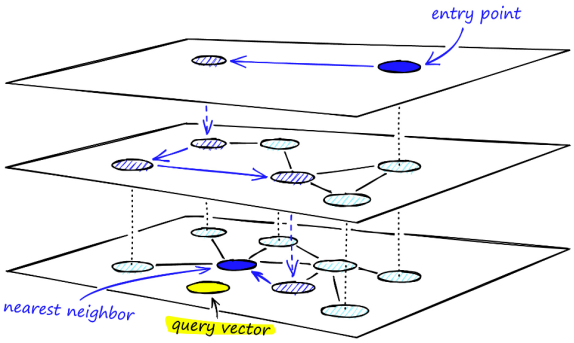
▼ **原理**

video link | article link

- ANN algorithms: 3 categories
  - trees
  - hashes
  - graphs —> **HNSW (hierarchical navigable small world graph)**
- HNSW
  - Top layer: longest links
  - Bottom layer: shortes links



- 在某層連到離 target 點最近的點後，就跳到下一層



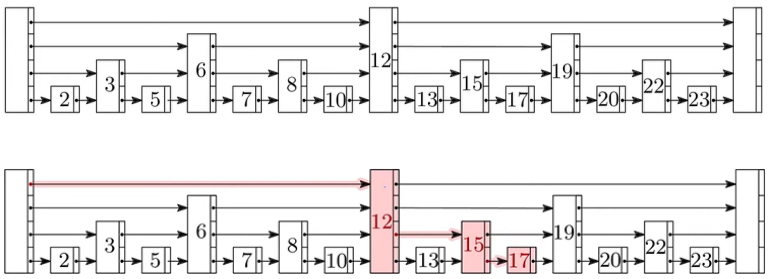
layer 0 會包含所有點

HNSW = NSW + Skip List

- Probability Skip List

  : Linked list with shortcuts (Searching method)

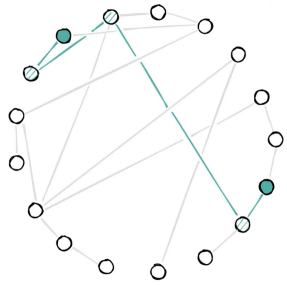
  - start in top level, go to successor if successor ≤ x
  - if successor in level > x, go one level down



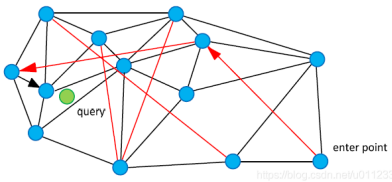
- Navigable Small World (NSW) graph

  : long range links (比較長的links) + short range links

  : Choose friend that is closest to our target vertex

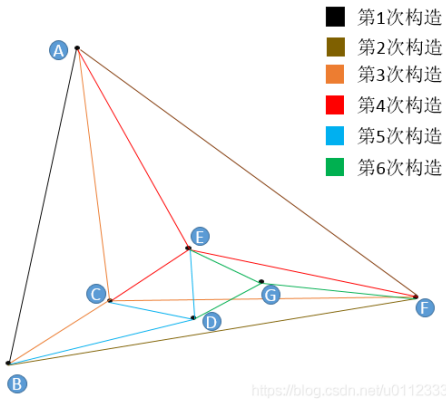


  - 解釋



NSW論文中配了這樣一張圖，黑色是近鄰點的連線，紅色線就是“高速公路機制”了。我們從enter point點進入查找，查找綠色點臨近節點的時候，就可以用過紅色連線“高速公路機制”快速查找到結果。

為什麼會形成“高速公路”呢？來看下面的例子

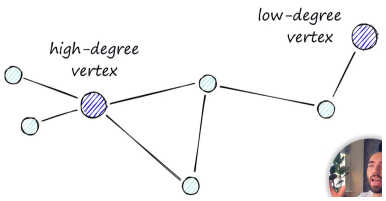


我們對7個二維點進行構圖，用戶設置m=3（每個點在插入時找3個緊鄰友點）。

1. 首先初始點是A點（隨機出來的），A點插入圖中只有它自己，所以無法挑選“友點”。
2. 然後是B點，B點只有A點可選，所以連接BA，此為第1次構造。
2. 然後插入F點，F只有A和B可以選，所以連接FA，FB，此為第2此構造。
3. 然後插入了C點，同樣地，C點只有A，B，F可選，連接CA，CB，CF，此為第3次構造。
4. 重點來了，然後插入了E點，E點在A，B，F，C中只能選擇3個點（m=3）作為“友點”，根據我們前面講規則，要選最近的三個，怎麼確定最近呢？樸素查找！從A，B，C，F任意一點出發，計算出發點與E的距離和出發點的所有“友點”和E的距離，選出最近的一點作為新的出發點，如果選出的點就是出發點本身，那麼看我們的m等於幾，如果不夠數，就繼續找第二近的點或者第三近的點，本著不找重複點的原則，直到找到3個近點為止。由此，我們找到了E的三個近點，連接EA，EC，EF，此為第四次構造。
5. 第5次構造和第6次與E點的插入一模一樣，都是在“現成”的圖中查找到3個最近的節點作為“友點”，並做連接。

圖畫完了，請關注E點和A點的連線，如果我再這個圖的基礎上再插入6個點，這6個點有3個和E很近，有3個和A很近，那麼距離E最近的3個點中沒有A，距離A最近的3個點中也沒有E，但因為A和E是構圖早期添加的點，A和E有了連線，我們管這種連線叫“高速公路”，在查找時可以提高查找效率（當進入點為E，待查找距離A很近時，我們可以通過AE連線從E直接到達A，而不是一小步一小步分多次跳轉到A）

- Terminology
  - high-degree vertex: many friends
  - low-degree vertex: fewer friends



▼ **Reference & Code**

github

https://github.com/nmslib/hnswlib

文章

適合打code看這篇的後面的流程。比較簡潔，有附論文code，解釋**建圖流程&搜索流程**

https://zhuanlan.zhihu.com/p/80552211 (2018-12-21)

適合懂了之後複習用。有解釋 HNSW sudo code

https://blog.csdn.net/u013630299/article/details/100893392

適合想要完全弄超懂。從頭到尾都有寫，背景脈絡清楚

https://blog.csdn.net/u011233351/article/details/85116719 (2018-12-21)

適合想要延伸資訊。很數學，沒有講很細，但是補充很多其他算法

https://www.modb.pro/db/103254 (2021-08-26)

▼ **程式碼參數註解**

- 計算一個 vector (query) 與 已經建立好的 map (.bin) 的關係時，不用再把 vector 加進 map 裡就可以搜
- `knn_query`

  返回兩個numpy數據。分別包括k個最近鄰結果的標籤和與這k個標籤的距離。
- `hnswlib.Index( space = 'l2' , dim = dim)`

Distance	parameter	Equation
Squared L2	'l2'	d = sum((Ai-Bi)^2)
Inner product	'ip'	d = 1.0 - sum(Ai*Bi)
Cosine similarity	'cosine'	d = 1.0 - sum(Ai*Bi) / sqrt(sum(Ai*Ai) * sum(Bi*Bi))

- `get_ids_list()`

  returns a list of all elements' ids.

- `get_items(ids)`

  returns a numpy array (shape:`N*dim`)

▼ **參數含意**

https://blog.csdn.net/redhatforyou/article/details/107021560

- `k`

  : 結果中返回的最近鄰的結果的個數k

- `M`

  : 表示在構建期間，每個元素創建的雙向鍊錶的數量。

  : (就是要有幾個友點的意思)

  - 合理範圍：2-100；通常：12-48
  - M值高在高召回率數據集上效果好
  - M值較低在低召回率數據集上效果好
  - M值決定了算法內存消耗，大概是 M×(810) Bytes

- `ef`

  : parameter controlling query time/accuracy trade-off

  - ef 越大，越準確，但檢索速度越慢
  - ef 可以是 大於 k； 小於集合大小之間的任意值

- `ef_construction`

  : parameter that controls speed/accuracy trade-off during the index construction.

- ef_constraction越大，構建時間越長，但是索引質量更好
- 在某種程度上提高ef_construction並不能提高index的質量。

▼ **實驗結果**

詳見 效能.csv 檔

▼ **ScaNN**

▼ **Info**

文章

https://medium.com/@kumon/similarity-search-scann-and-4-bit-pq-ab98766b32bd

https://ai.googleblog.com/2020/07/announcing-scann-efficient-vector.html (2020-07-8)

(中文版 https://zhuanlan.zhihu.com/p/164971599 (2020-07-29))

github

https://github.com/google-research/google-research/tree/master/scann