Coursework

group project (70%):

submit by **Friday 20/11/2026 12:00**

individual review (30%)

submit by **Monday 30/11/2026 12:00**

Learn / coursework planner

`https://www.learn.ed.ac.uk/ultra/courses/_137627_1/document/_12996909_1?view=content&state=view`

Opencourse

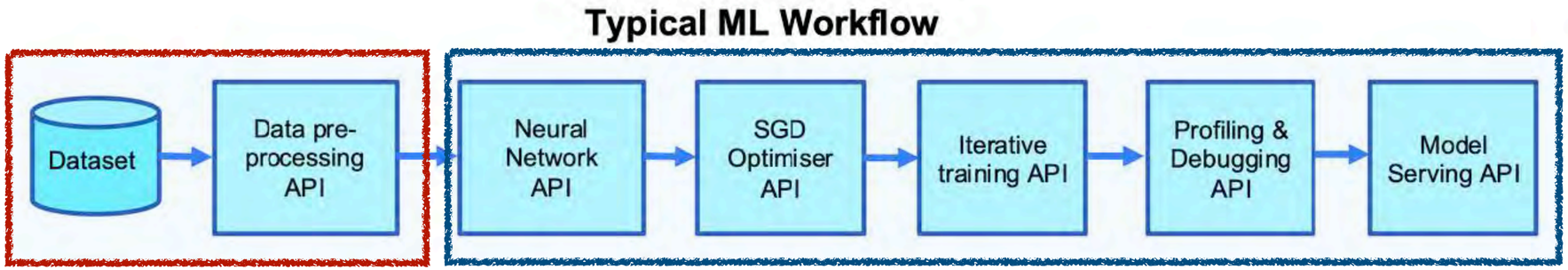
`https://opencourse.inf.ed.ac.uk/mls`

Vector Databases and RAG (VDBMS) Systems

Week 7 — Machine Learning Systems (Fall 2026)

Instructor: Yang Cao

Recall: overview



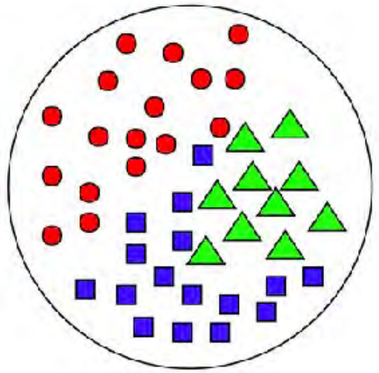
Week 2 Week 3, 5, 6 (GPU, training, inference)

(encoder, basic RAG concept)

Lecture plan:

- RAG (VDBMS) systems (**Week 7**)
- AI Infra & *Coursework Q&A* (Week 8)
- Agent systems and memory (Week 9)
- Interoperable AI data systems (Week 10)
- AI cluster (Week 11)

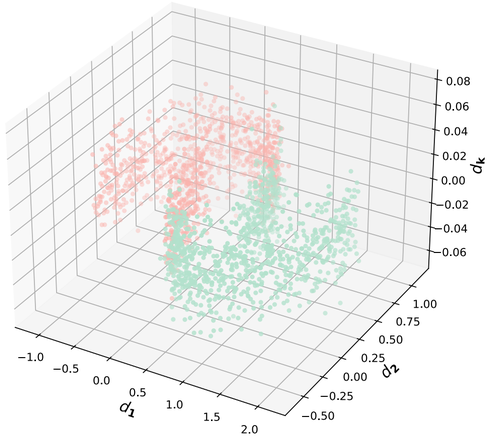
Recall: semantic (similarity) search via embedding vectors



similarity (metric) space

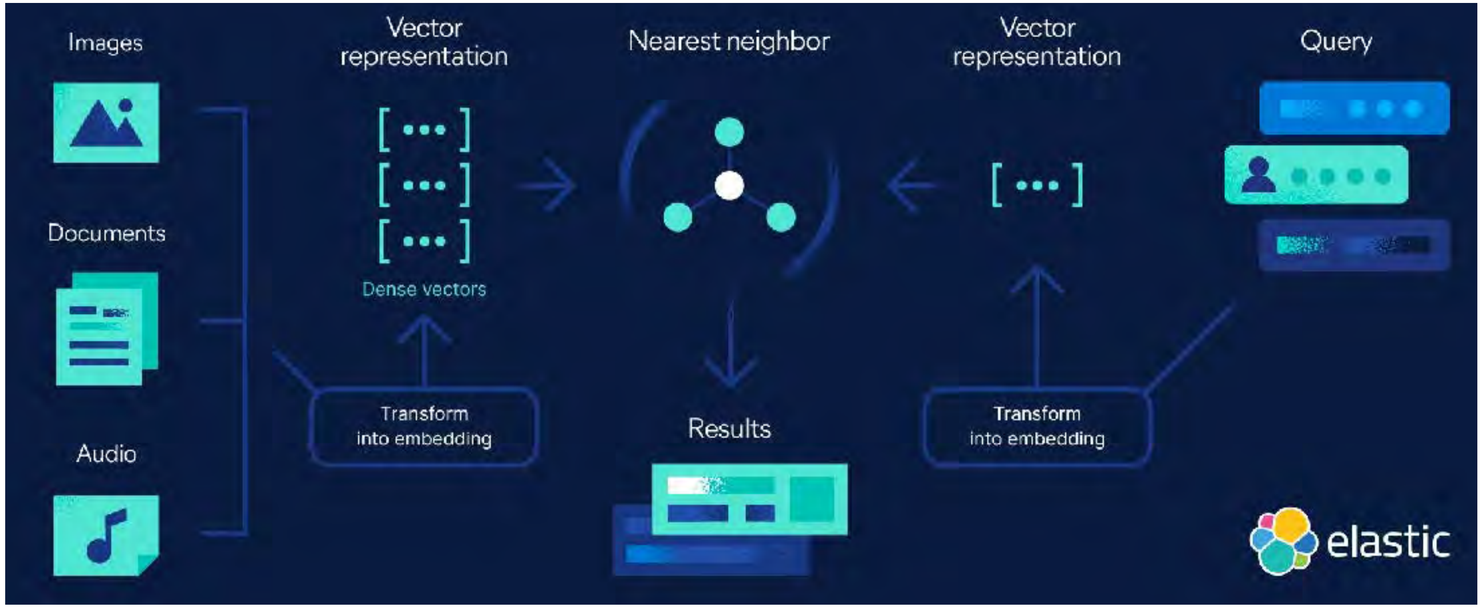
Task: top-k similarity search

implemented by



vector space

Task: top-k vector search



Credit: https://www.elastic.co/what-is/knn

Recall: similarity search → vector search

Input: dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, $\mathbb{R}^d$, query $\mathbf{q}$, distance function $\mathsf{dist}(\mathbf{x}, \mathbf{y})$ $\quad \mathbf{x} = (x_1, \ldots, x_d)$, $\mathbf{y} = (y_1, \ldots, y_d)$

Output: $S_k = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subseteq D$ such that

$$\mathsf{dist}(\mathbf{v}_i, \mathbf{q}) \leq \mathsf{dist}(\mathbf{x}, \mathbf{q}), \text{ for any } \mathbf{v}_i \in S_k, \mathbf{x} \in D \backslash S_k$$

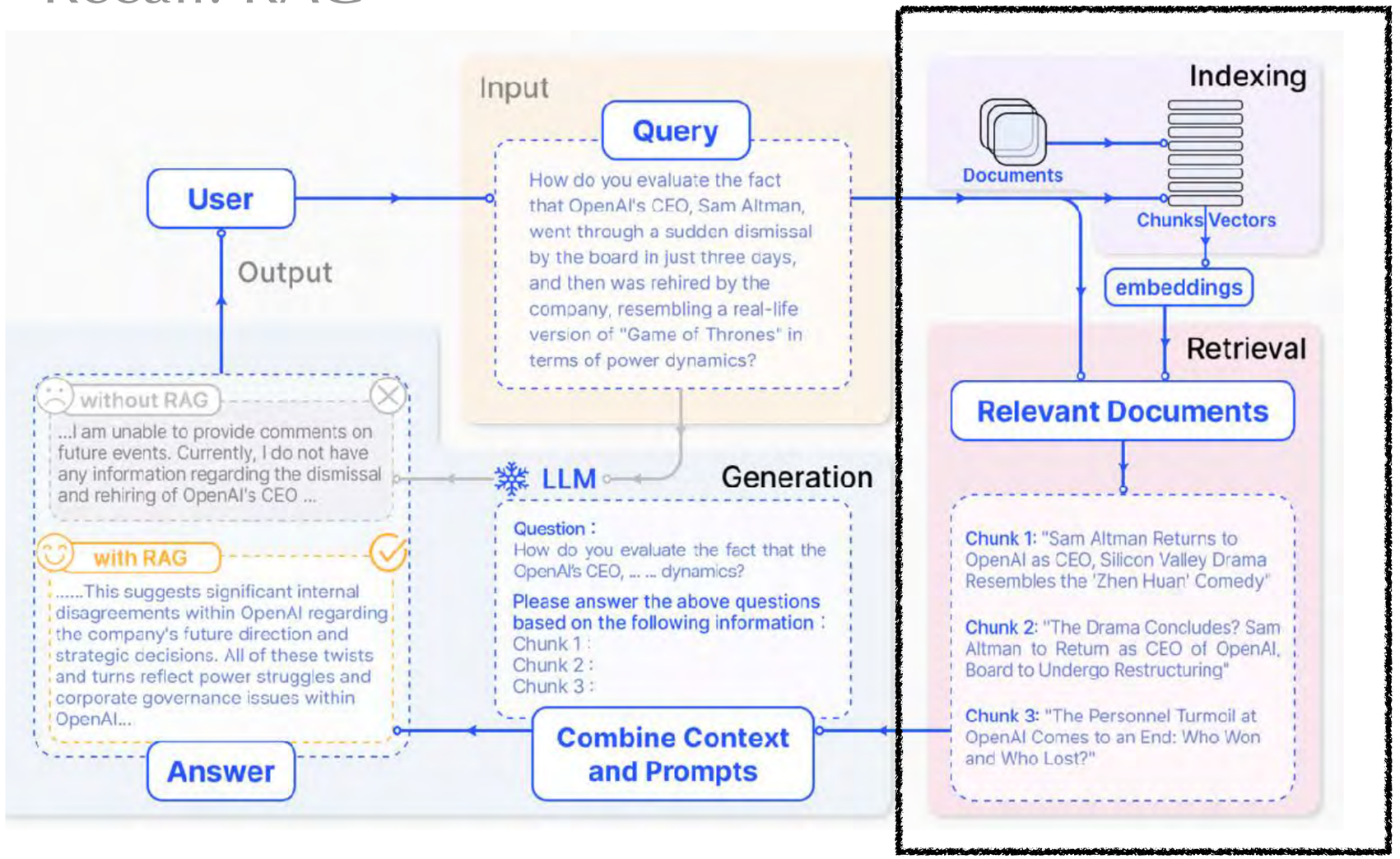
L_2-norm of $\mathbf{x}$: $\| \mathbf{x} \|_2 = \sqrt{\sum_{i=1}^{d} x_i^2}$ ($\mathbf{x}$ is L_2-normalised if $\| \mathbf{x} \|_2 = 1$)

Euclidean $\quad \mathsf{dist}(\mathbf{x}, \mathbf{y}) \doteq \| \mathbf{x} - \mathbf{y} \|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$

dot-product $\quad \mathsf{dist}(\mathbf{x}, \mathbf{y}) \doteq \mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{d} x_i y_i$

cosine $\quad \mathsf{dist}(\mathbf{x}, \mathbf{y}) \doteq \cos(\theta_{\mathbf{xy}}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\| \mathbf{x} \|_2 \| \mathbf{y} \|_2} = \frac{\Sigma_{i=1}^{d} x_i y_i}{\Sigma_{i=1}^{d} x_i^2 \Sigma_{i=1}^{d} y_i^2}$

Recall: RAG



embedding (encoder)

+

Top-k vector search

inference-time toll

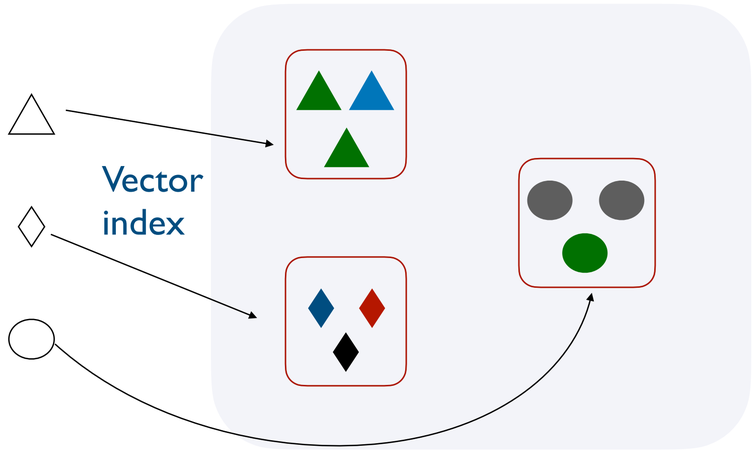
How can we make top-k search efficient?

vector indices for searching embeddings

Approximate Nearest Neighbor

Idea: approximate top-k answers

- trade **accuracy** for better **performance**
- use pre-computed **index**



Index: a key concept in database management systems (DBMS)

- a function f : predicate $p \rightarrow$ records satisfying p
- vector index: vector $v \rightarrow$ vectors u close to v (not necessarily the closest)
- criterion: construction *independent* of q (works for all queries)

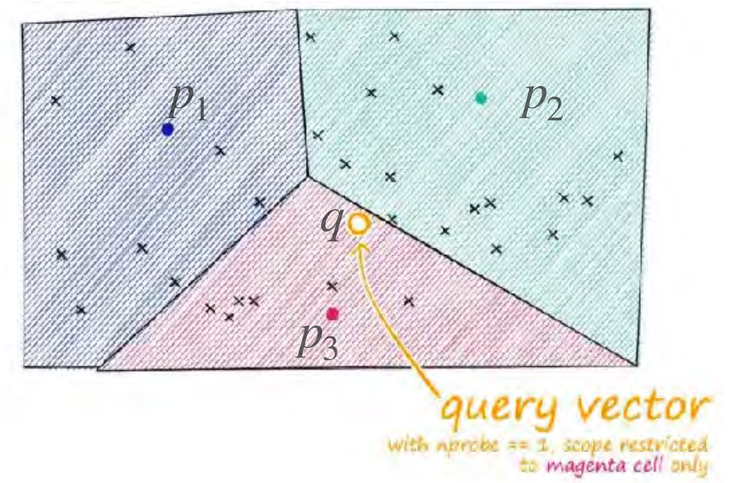
ANN index: IVF (inverted file index)

Idea: partition vectors in D into a *Voronoi* diagram (a set of subsets — "cells")

"birds of a feather flock together"

$$D = \bigcup_{i=1}^{K} \text{cell}(p_i)$$

$$\text{cell}(p_i) = \{x \in D \mid \text{dist}(x, p_i) \leq \text{dist}(x, p_j)(\forall j \neq i)\}$$



Properties: given a set D of vectors and a set $S \subset D$ of seeds

- The Voronoi diagram of D *w.r.t.* S is unique
- 1-NN query q over S returns c_i if $q \in \text{cell}(p_i)$
- heuristic for ANN: find $\text{cell}(p_i)$, and search top-k within $\text{cell}(p_i)$

edge case: (find *n_probe* closest seeds to q and search in the cells)

Figure credit: James Briggs, Faiss: The Missing Manual, https://www.pinecone.io/learn/series/faiss/

ANN index: IVF (Voronoi)

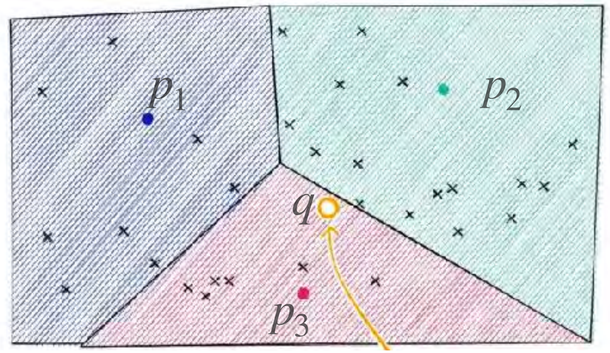
Inverted Index (= fancy name of database index for non-DB domains)

a function f : predicate $p(x) \rightarrow$ records y satisfying $p(x)$

(Index key: x, index value: y)

doc 1: "What goes around comes around"
doc 2: "It goes around the world"

Index key	Index value
"What"	doc 1
"goes"	doc 1, doc 2
"around"	doc 1, doc 2
"comes"	doc 1
"around"	doc 1, doc 2
"It"	doc 2
"the"	doc 2
"World"	doc 2



Index key	Index value
p_1	$x_1, x_2, \ldots$
p_2	$y_1, y_2, \ldots$
p_3	$z_1, z_2, \ldots$

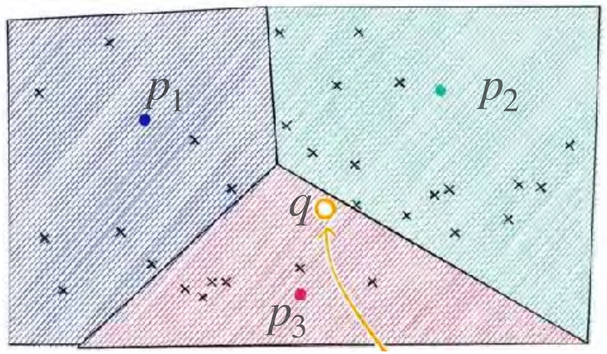
Index predicate: $x_i \in \mathsf{cell}(p_1)$

ANN index: IVF (Voronoi)

Inverted Index (= fancy name of database index for non-DB domains)

a function f : predicate $p(x) \rightarrow$ records y satisfying $p(x)$

(Index key: x, index value: y)



Index key	Index value
p_1	$x_1, x_2, \ldots$
p_2	$y_1, y_2, \ldots$
p_3	$z_1, z_2, \ldots$

Index predicate: $x_i \in \mathsf{cell}(p_1)$

Observations:

- IVF can be constructed offline once we know the *Voronoi* diagram — **IVF(Voronoi)**
- Assuming uniform cell size (for K cells, K $\leq \sqrt{n}$), **IVF(Voronoi)** is K times faster

How do we compute the Voronoi diagram for D?

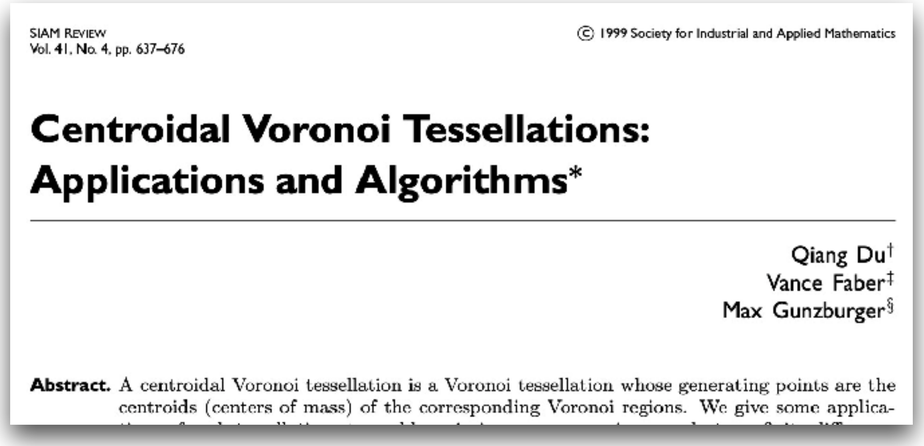
ANN index: IVF construction

Construction of IVF: Voronoi diagram ~ k-means clustering

K-means clustering:

- Input: a set D of vectors, K
- Output: $\mathbf{C} = \{C_1, \ldots, C_K\}$, s.t. $C_1 \cup \cdots \cup C_K = D$
- Objective: within-cluster variances (sums of squares)

$$\arg\min_{\mathbf{C}} \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} \| \mathbf{x} - \mu_i \|_2^2, \text{ where } \bar{\mu}_i = \frac{\Sigma_{\mathbf{x} \in C_i} \mathbf{x}}{| C_i |}$$


SIAM REVIEW
Vol. 41, No. 4, pp. 637–676

© 1999 Society for Industrial and Applied Mathematics

Centroidal Voronoi Tessellations: Applications and Algorithms*

Qiang Du†
Vance Faber‡
Max Gunzburger§

Abstract. A centroidal Voronoi tessellation is a Voronoi tessellation whose generating points are the centroids (centers of mass) of the corresponding Voronoi regions. We give some applica-

Connection: in Euclidean space,

- optimal k-means clustering = a *centroidal* Voronoi diagram of D cell($\bar{\mu}_i$) = C_i

ANN index: IVF with k-means clustering

k-means clustering

Construction (**offline**):

- group vectors into clusters $C_1, \ldots, C_K$
- compute centroid μ_i of C_i

optimal k-means = Voronoi diagram

IVF (k-means) = IVF (Voronoi)

Query procedure (**online**):

- find $C_j : \arg\min_j \text{dist}(\mu_j, q)$
- find top-k within C_j

multiple times as needed (*n_probe*)

Optimal k-means clustering is NP-complete ☹

Parameters of IVF search:

- #clusters to create (K): larger K = faster search (when K $\in o(\sqrt{n})$)
- #cluster to probe (n_probe): larger n_probe = more accurate answers

K-means clustering: Lloyd's K-means algorithm (1957)

ALGORITHM 1: Lloyd's K-means algorithm

Input: $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, K;
Output: clustering $\mathbf{C} = \{C_1, \ldots, C_K\}$ of D

initialize C_1, ..., C_K;
while not converged **do**
  compute centroid $\mu_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$ for each $i = 1, \ldots, K$;
  update C_1, ..., C_k by assigning each $\mathbf{x} \in D$ to cluster C_ℓ whose centroid μ_ℓ it is closest to;
  if C_1, ..., C_K don't change, we have converged;
return clustering C_1, ..., C_K

Properties: in Euclidean space,

- converges to a **local optimal**
- complexity O(nKdi) (worst-case i is $2^{\Omega(\sqrt{n})}$)
- parallelizable (GPU-friendly)

Weakness:

- sensitive to initialization (remedy: use (approx.) K-means++ for initialization)
- may not converge with non-Euclidean distances (remedy: set max iteration bound)

K-means clustering: Lloyd's K-means algorithm (1957)

ALGORITHM 1: Lloyd's K-means algorithm

Input: $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, K;
Output: clustering $\mathbf{C} = \{C_1, \ldots, C_K\}$ of D

initialize C_1, ..., C_K;
while not converged **do**
  compute centroid $\mu_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$ for each $i = 1, \ldots, K$;
  update C_1, ..., C_k by assigning each $\mathbf{x} \in D$ to cluster C_ℓ whose centroid μ_ℓ it is closest to;
  if C_1, ..., C_K don't change, we have converged;
return clustering C_1, ..., C_K

Properties: in Euclidean space,

- converges to
- complexity C
- parallelizable

Weakness:

tion (remedy: use
+ for initialization)

ith non-Euclidean
et max iteration bound)

Further reading:

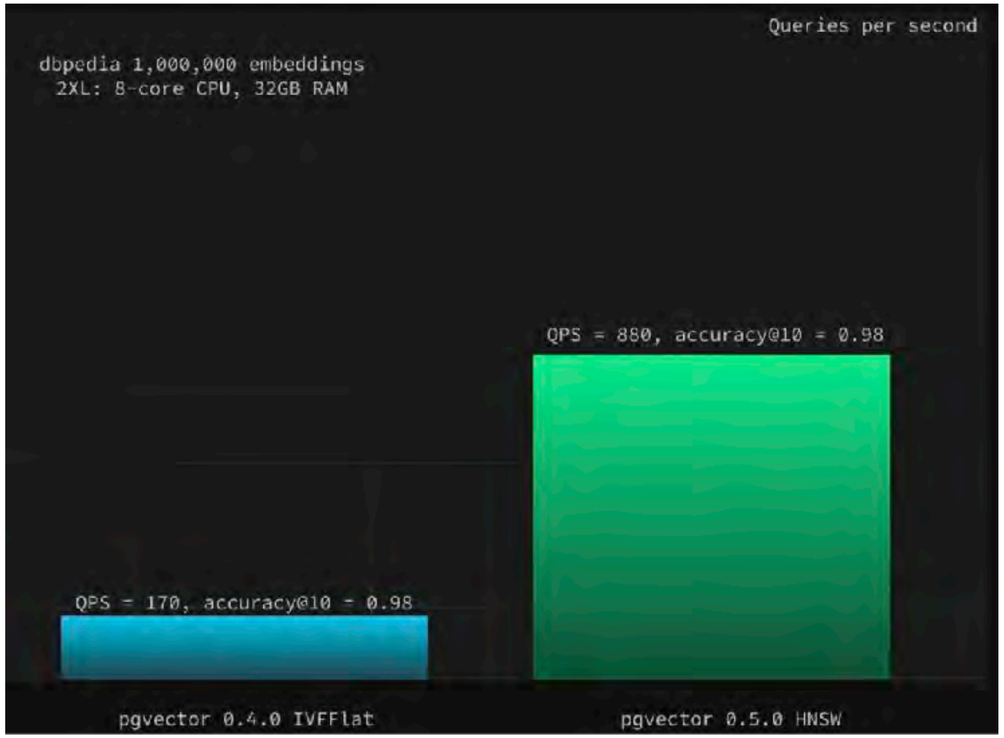
1. Farivar, Reza, et al. "A Parallel Implementation of K-Means Clustering on GPUs." *Pdpta*. Vol. 13. No. 2. 2008.
2. Arthur, David, and Sergei Vassilvitskii. "How slow is the k-means method?." *Proceedings of the twenty-second annual symposium on Computational geometry*. 2006.
3. Du, Qiang, Vance Faber, and Max Gunzburger. "Centroidal Voronoi tessellations: Applications and algorithms." *SIAM review* 41.4 (1999): 637-676.

ANN index: Graph-based vector index

Intuition: reduce vector search to graph traversal

- map a set D of vectors into a graph G_D
  - vertices are vectors, edges encode vector distances
  - nearer vectors are more likely connected

- reduce ANN search to graph traversal
  - **greedy routing strategy** holds when G_D is **navigable**



Credit: https://supabase.com/blog/increase-performance-pgvector-hnsw

Key concept: **Navigable** Graphs

A graph $G(V, E)$ is **navigable** if for any two vertices $s, t \in V$:

- if t is reachable from s, then **Greedy** always finds a path from s to t

The **Greedy** routing strategy:

- starting from s, each step we move from a vertex v to one of its neighbours
- **greedy routing**: at v, pick its neighbour $u \in \mathcal{N}(v)$ that minimises $\mathsf{dist}(u, t)$

An efficient local search strategy, need no global knowledge of G

Why would we care?

The **exact** nearest neighbour can be found by **Greedy**

on the **Delaunay Graph** of D

that is derived from the Voronoi diagram of $\mathbb{R}^d$ using D as the seeds

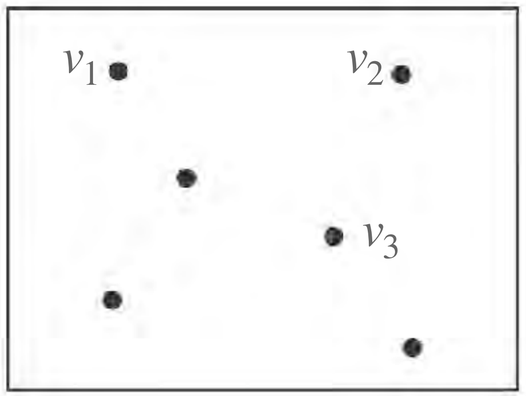
Delaunay Graph, a special navigable graph

Voronoi diagram of $\mathbb{R}^d$ using $D = \{v_1, \ldots, v_n\}$ as seeds:

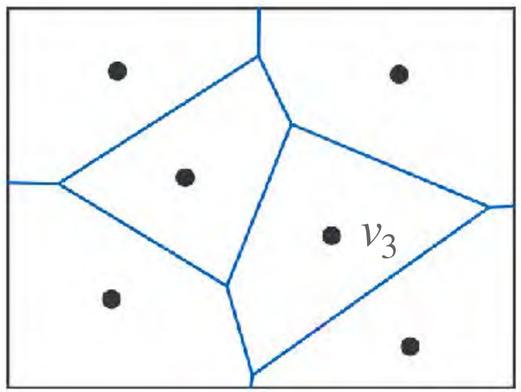
- a partition of $\mathbb{R}^d$ into n Voronoi cells $C_1, \ldots, C_n$
- $\forall q \in \mathbb{R}^d$, q is only in C_i if $\mathsf{dist}(q, v_i) < \mathsf{dist}(q, v_j)$, for any $j \neq i$

Delaunay graph $G(D, E)$ of $D = \{v_1, \ldots, v_n\}$:

- $(v_i, v_j) \in E$ if C_i and C_j are adjacent cells (v_i and v_j: Voronoi neighbours)

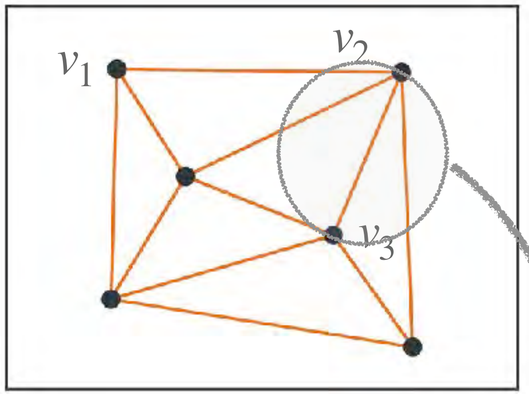


$D = \{v_1, \ldots, v_n\}$



perpendicular bisector between points

Voronoi diagram with D



Delaunay graph of D

every edge defines an "exclusive" circle

Figure credit: https://cartography-playground.gitlab.io/playgrounds/triangulation-delaunay-voronoi-diagram/

Delaunay Graph, where **Greedy** is no longer a heuristic

Property of Delaunay graph $G(D, E)$ for $D = \{v_1, \ldots, v_n\}$:

- for any query q only in Voronoi cell C_i, **Greedy** on G terminates at v_i
- complexity: total #-edges examined (i.e., #-steps * average vertex degree)

Property: **Greedy** is optimal for exact kNN search

Can we use Delaunay graph $G(D, E)$ for kNN queries over D?

No for two reasons:

(a) high-complexity: exponential in d ($O(n \log n)$ for special case $d = 2$ or 3)

(b) almost fully connected with high dimensionality d

↓ [Klee, 1980] average degree $O(n)$ in dimension $d = 3$ or higher

reduces to brute-force search ☹

Idea: approximating DG via **sparser** navigable graphs

Complexity: #-steps * average vertex degree

Sparsity is the key:

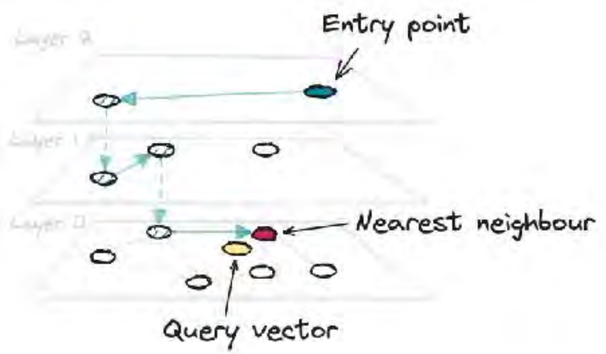
- worst: a complete graph of D is navigable
- slightly better: delaunay graph is navigable
- Ideal: find (**approximately**) navigable G of D as sparse as possible?

  so that we can use **Greedy** for ANN search!

Two examples:



Navigable Small-World Graphs



Hierarchical Navigable Small-World Graphs

Figure credit: Prashanth Rao, http://thedataquarry.com/posts/vector-db-3/

Navigable Small-World Graphs

Randomly rewired graphs:

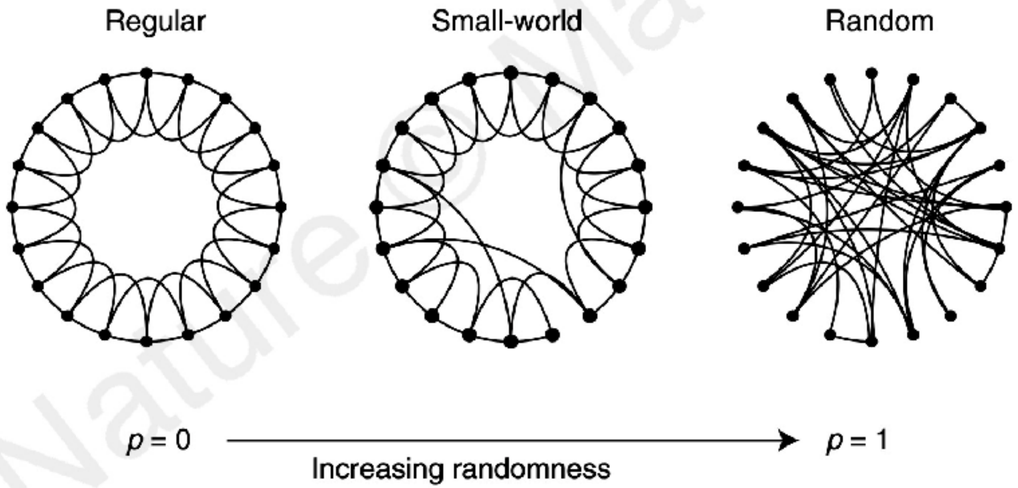
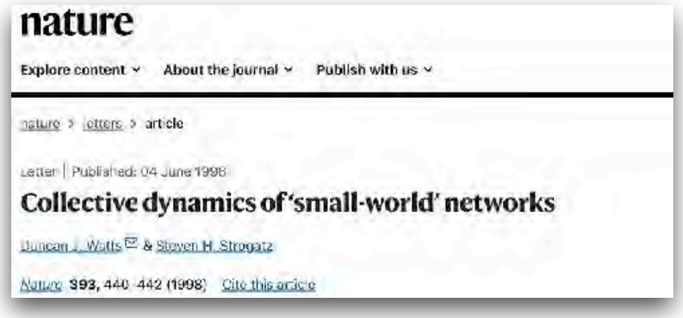


Figure 1 Random rewiring procedure for interpolating between a regular ring lattice and a random network, without altering the number of vertices or edges in the graph. We start with a ring of n vertices, each connected to its k nearest neighbours by undirected edges. (For clarity, $n = 20$ and $k = 4$ in the schematic


nature

Explore content ~ About the journal ~ Publish with us ~

nature > letters > article

Letter | Published: 04 June 1998

Collective dynamics of 'small-world' networks

Duncan J. Watts & Steven H. Strogatz

Nature 393, 440–442 (1998) Cite this article

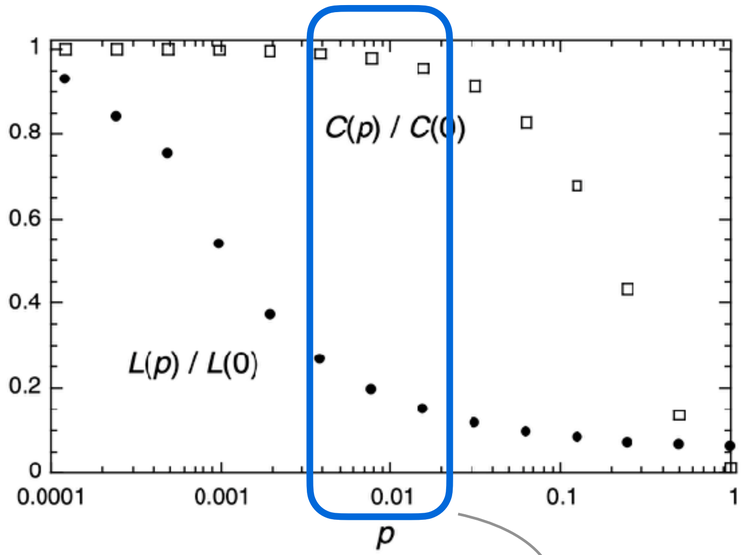
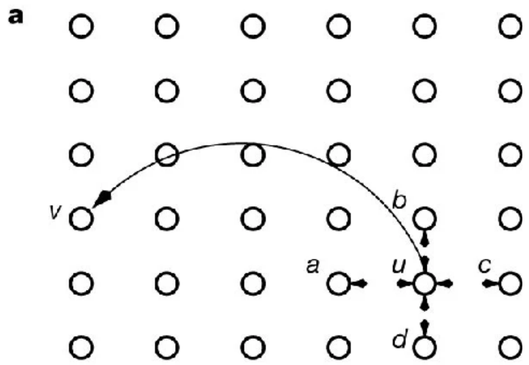


Figure 2 Characteristic path length $L(p)$ and clustering coefficient $C(p)$ for the family of randomly rewired graphs described in Fig. 1. Here L is defined as the

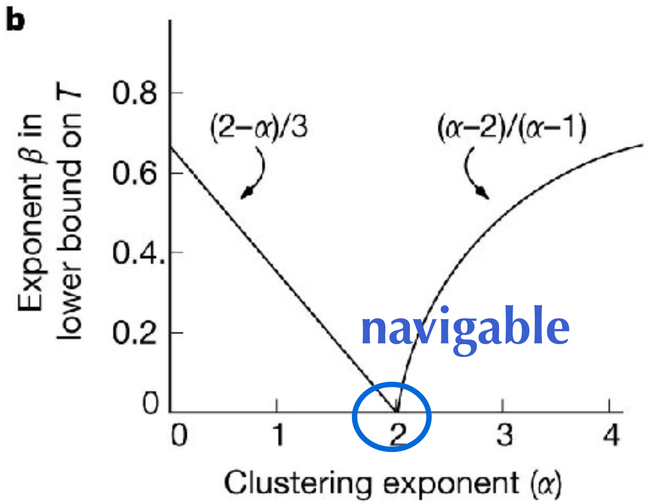
Small-world graphs:

- Small shortest path lengths
- Friends of friends are also friends

Navigable Small-World Graphs



connect (u,v) with probability $p = \text{dist}_{L_1}(u, v)^{-\alpha}$



Greedy complexity: $O(n^{\beta})$



$O(\log n)$ **greedy** search time

Not all small-world graphs are **navigable**.

- $\alpha = d$ for d-dimension Kleinberg's graph

How can we construct such NSWG for D?

ANN index: NSW (Navigable Small-World)

Given $D = \{v_1, \ldots, v_n\}$, construct its NSW graph-index G_D incrementally:

- Initially, G_D is empty
- for each $v_i \in D$, insert v_i to G_D as follows:
  - Connect v_i to its M nearest neighbours that are already in G_D

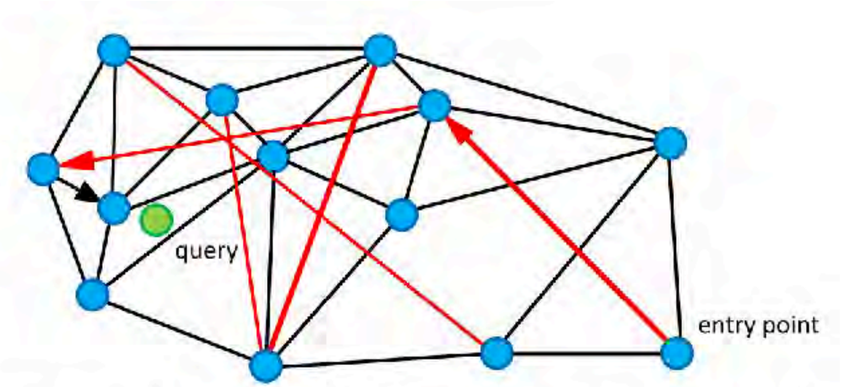


Fig. 1. Graph representation of the structure. Circles (vertices) are the data in metric space, black edges are the approximation of the Delaunay graph, and red edges are long range links for logarithmic scaling. Arrows show a sample path of the greedy algorithm from the entry point to the query (shown green). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



ANN index: NSW (Navigable Small-World)

Given $D = \{v_1, \ldots, v_n\}$, construct its NSW graph-index G_D incrementally:

- Initially, G_D is empty
- for each $v_i \in D$, insert v_i to G_D as follows:
  - Connect v_i to its M nearest neighbours that are already in G_D

The resulting graph G_D is (approximately) navigable and small-world.

Connection: Resulted G_D vs. Kleinberg's NSWG:

- edges added earlier in $G_D \rightarrow$ rewired edges in NSWG
  - ensure navigability (traversable in $O(\log n)$)
- edges added later in $G_D \rightarrow$ short distances in NSWG
  - ensure search accuracy (keep local neighborhood)

ANN index: Limitation of NSW

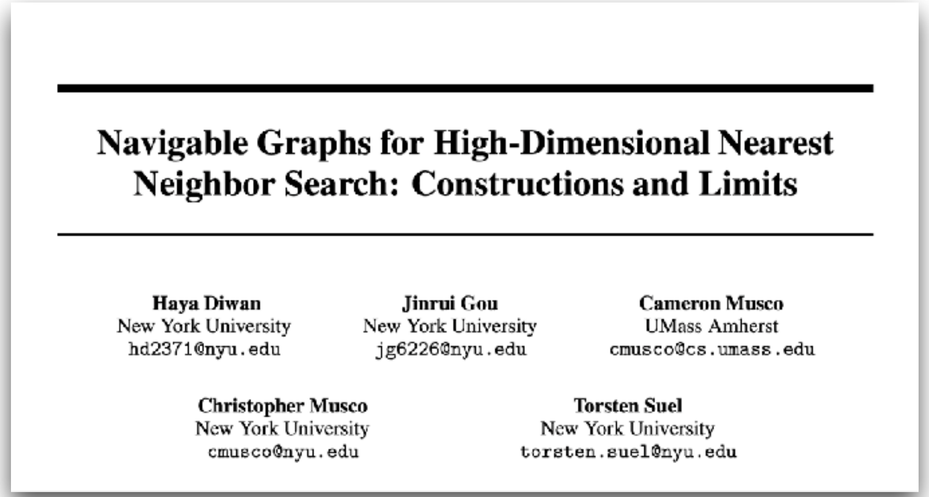
Given $D = \{v_1, \ldots, v_n\}$, construct its NSW graph-index G_D incrementally:

- Initially, G_D is empty
- for each $v_i \in D$, insert v_i to G_D as follows:
  - Connect v_i to its M nearest neighbours that are already in G_D

The resulting graph G_D is (approximately) navigable and small-world.

Limitation: any navigable graph for D has average degree of $\Omega(\sqrt{n}/\log n)$

if vectors are high-dimensional w.r.t. n $(d = \Omega(\log^3 n))$



Navigable Graphs for High-Dimensional Nearest Neighbor Search: Constructions and Limits

Haya Diwan
New York University
hd2371@nyu.edu

Jinrui Gou
New York University
jg6226@nyu.edu

Cameron Musco
UMass Amherst
cmusco@cs.umass.edu

Christopher Musco
New York University
cmusco@nyu.edu

Torsten Suel
New York University
torsten.suel@nyu.edu

ANN index: Limitation of NSW

Limitation: any navigable graph for D has average degree of $\Omega(\sqrt{n}/\log n)$

if vectors are high-dimensional w.r.t. n $(d = \Omega(\log^3 n))$

Overcome the limitation on sparsity?

very large M needed

long query time; costly storage

From **Local** to **Beam** search:

Breaking the "Strict Greedy" rule

use Priority Queue ("ef" parameter)

exploring ef closest nodes instead of one

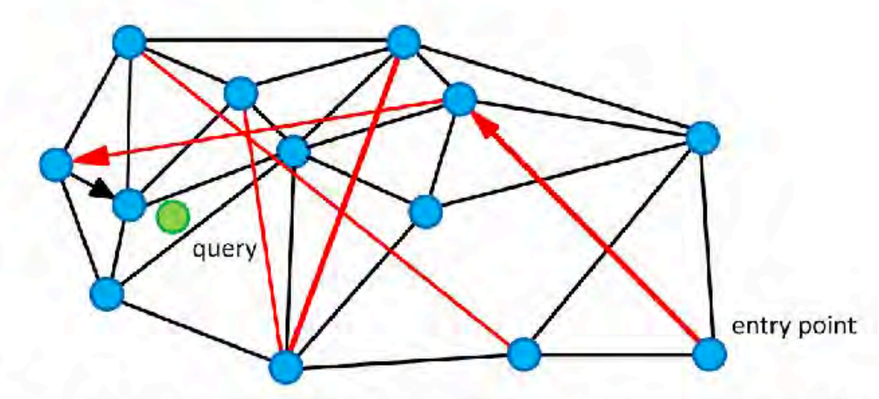
practice: reduce $\Omega(\sqrt{n}/\log n)$ to O(M) with M = 16 or 32

Complexity:

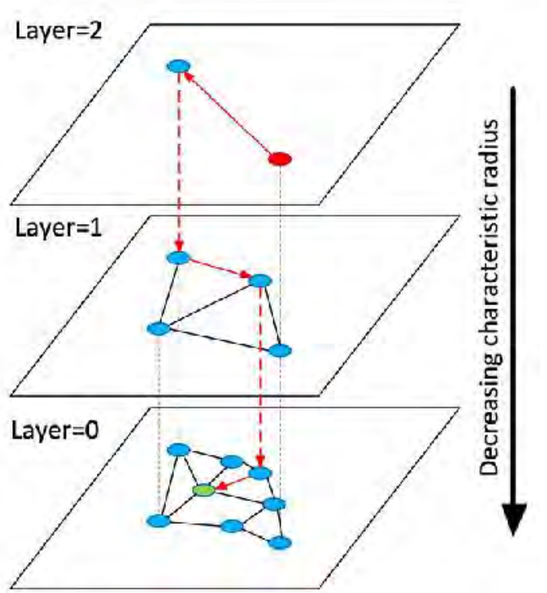
- query (search) complexity: $\approx O(\log^2 n)$
- Index building complexity: $O(n \log^2 n)$

an empirical estimation

ANN index: HNSW (Hierarchical Navigable Small World)

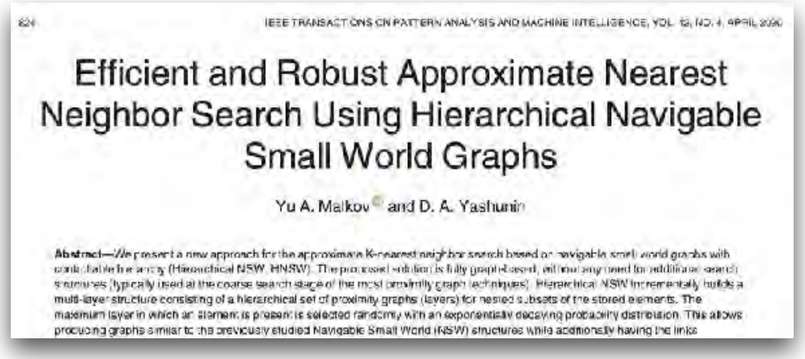


stacking up NSWs



Problems with NSW:

- poly-logarithmic search complexity
- “traffic hubs”/high out-degrees vertices


824 IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 42, NO. 4, APRIL 2020

Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs

Yu A. Malkov and D. A. Yashunin

Abstract—We present a new approach for the approximate K-nearest neighbor search based on navigable small world graphs with controllable hierarchy (Hierarchical NSW, HNSW). The proposed solution is fully graph-based, without any need for additional search structures (typically used at the coarse search stage of the most proximity graph techniques). Hierarchical NSW incrementally builds a multi-layer structure consisting of a hierarchical set of proximity graphs (layers) for nested subsets of the stored elements. The maximum layer in which an element is present is selected randomly with an exponentially decaying probability distribution. This allows producing graphs similar to the previously studied Navigable Small World (NSW) structures while additionally having the links

Hierarchical NSW (HNSW):

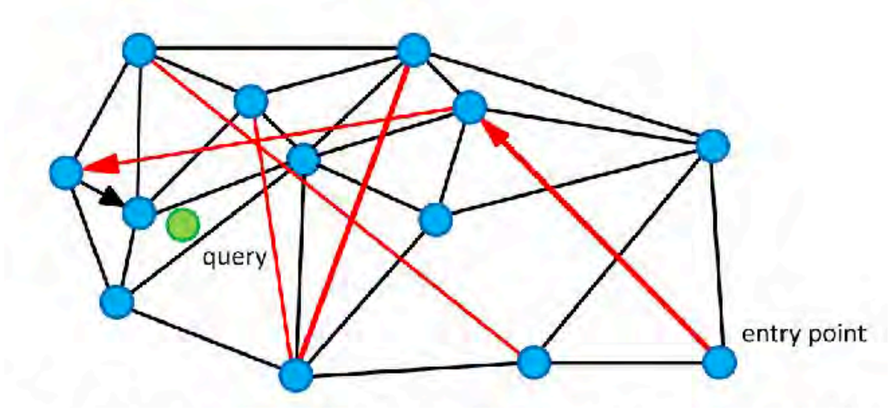
- stack multiple NSWs into a hierarchical structure
  - layer 0: NSW of D, layer 1: NSW of c%-D, …
  - upper layer: larger distance scale

Sampled through a probability distribution

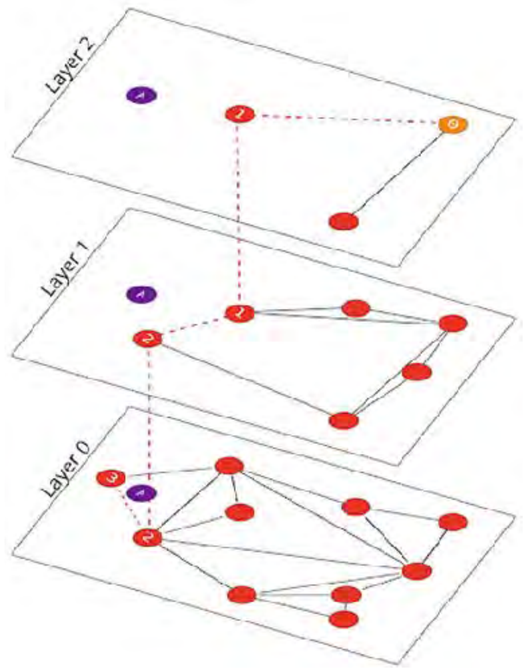
- bound average degree
  - avoid adding an edge that forms the longest edge of a triangle

RNG (relative neighborhood graph)

ANN index: HNSW (Hierarchical Navigable Small World)



stacking up NSWs

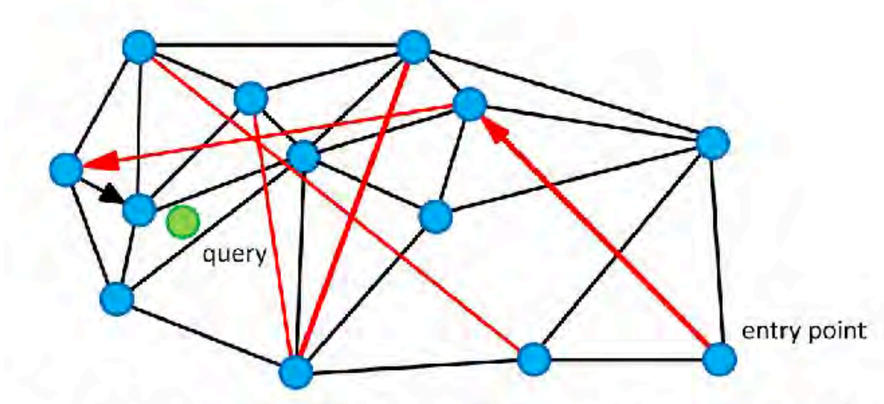


credit: https://www.datastax.com/guides/hierarchical-navigable-small-worlds

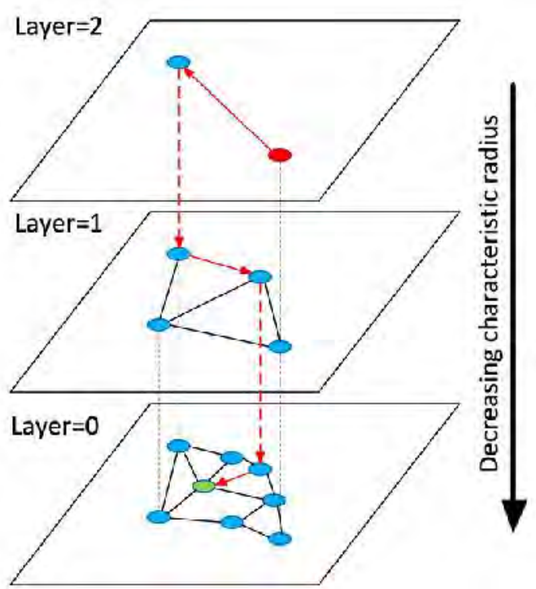
Query with HNSW (for query vector q):

- start with a pre-defined "entry" vertex as the *current* vertex in the top layer
- use **Greedy** to find closest neighbor u of the current vertex at the layer
- use u as the *current* vertex in the layer one level below, and go to the last step
- repeat until layer 0 is reached, and search top-k results via **Greedy**

ANN index: HNSW (Hierarchical Navigable Small World)

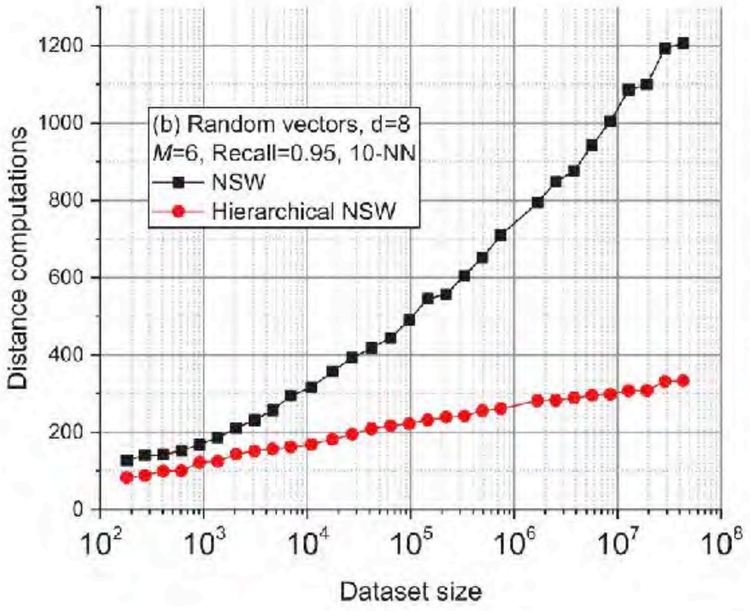


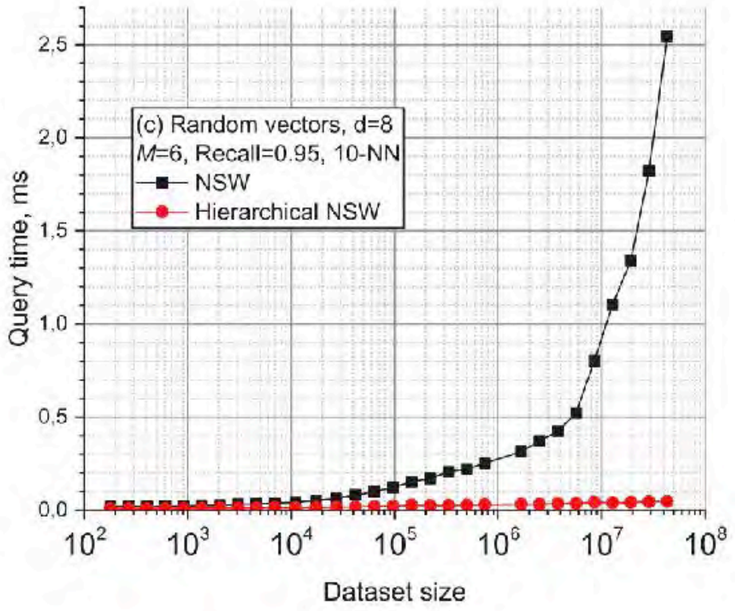
stacking up NSWs



Complexity with HNSW:

- building complexity: $O(n \log n)$
- query complexity: $O(\log n)$





Malkov, Y. A et al. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* (2020)

Knowing the limit …

Assumption of embedding-based retrieval

Key **principle**:

mapping a query and data object to dense embedding vectors

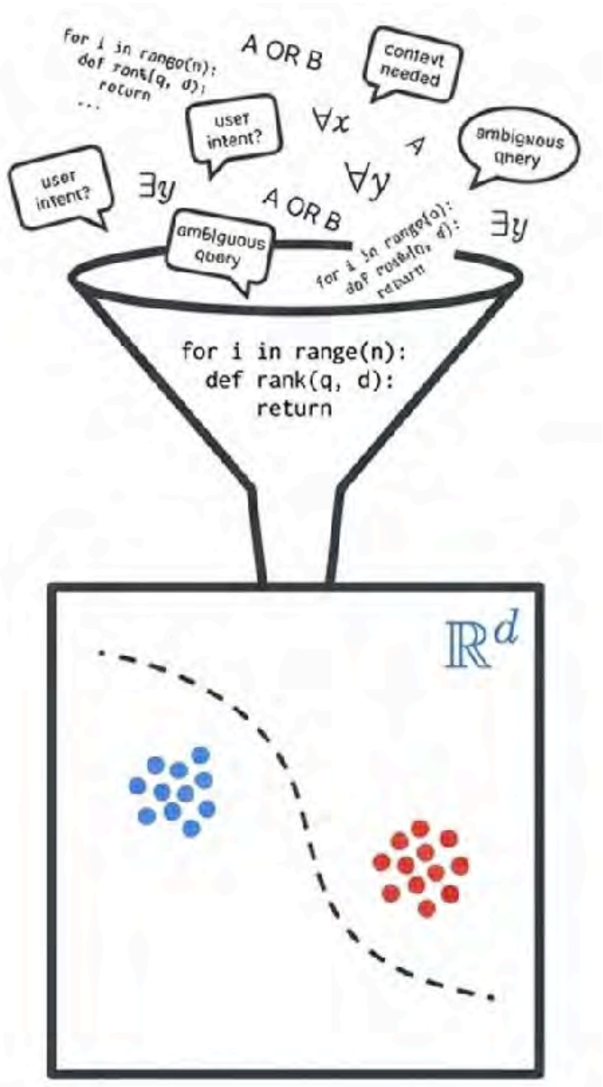
semantic relevance is “accurately” encoded by vector distances

Implicit **assumption**

any object, any query, and any definition of semantic relevance can be mapped into a fixed d-dimensional geometric space

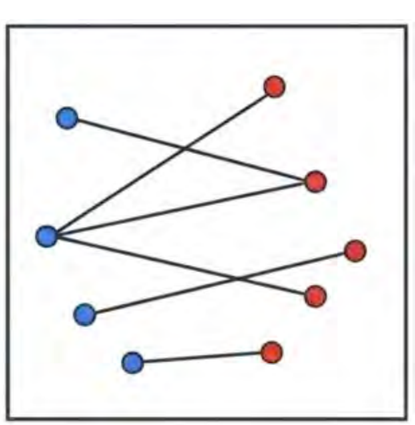
Implicit **belief**

if an embedding model fails, it simply needs more parameters, more data, or a larger d

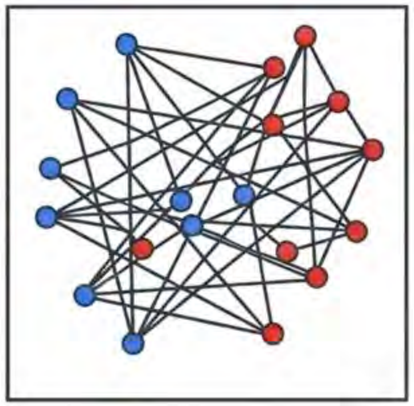


*credit: image generated by Gemini 3

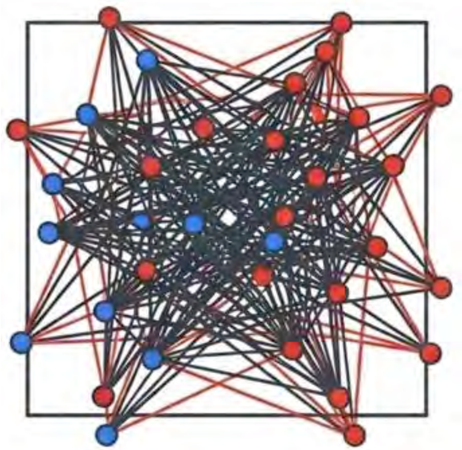
Semantic Similarity



+ connections



+ connections



increasing complexity of semantic similarity

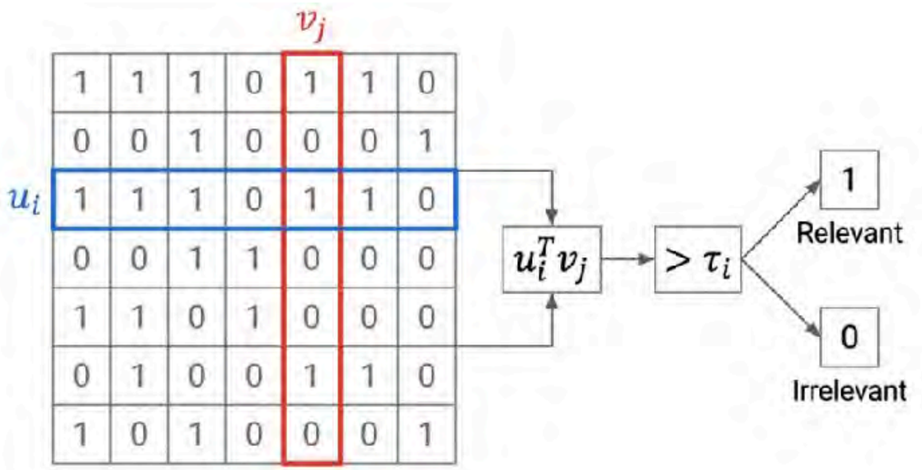
Can a fixed d-dimensional space truly encode any arbitrary connection between queries and objects?

Embedding retrieval → matrix factorization

m queries, n documents

ground-truth relevance matrix $A \in \{0,1\}^{m \times n}$

$A_{ij} = 1$ iff document j is relevant to query i



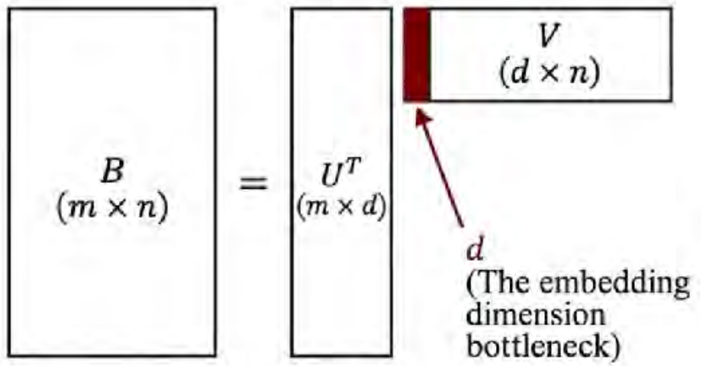
Embedding-based retrieval

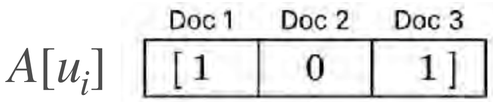
$V \in \mathbb{R}^{d \times n}$: n documents

$U \in \mathbb{R}^{d \times m}$: m queries

search computes score matrix $B = U^T V$

not all d (dimension) can allow B to "**express**" A





$B_{i1} > B_{i2}$ relevance scores must strictly

$B_{i3} > B_{i2}$ exceed irrelevant scores

The Geometric Ceiling

Google DeepMind

On the Theoretical Limitations of Embedding-Based Retrieval

Orion Weller*,[1,2], Michael Boratko[1], Iftekhar Naim[1] and Jinhyuk Lee[1]
[1]Google DeepMind, [2]Johns Hopkins University

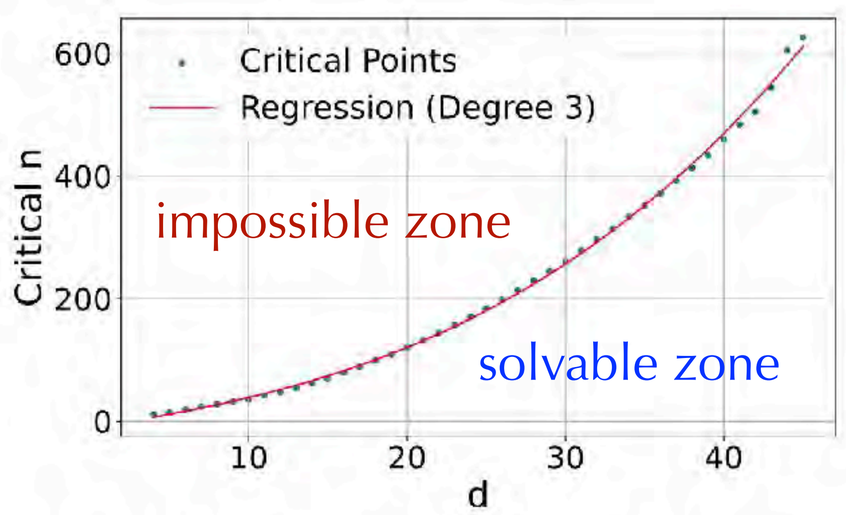


Figure 2 | The critical-n value where the dimensionality is too small to successfully represent all the top-2 combinations. We plot the trend line as a polynomial function.

Key takeaway

we need at least $\text{rank}_{\pm}(2A - \mathbf{1}_{m \times n})$ dimensions to capture A exactly

(kinda making "folklore" rigorous)

a good example of excellent reviewing (for 2nd part of coursework):

`https://openreview.net/forum?id=k9CzIvzfaA&noteId=914Qwkwemm`

check out review & discussion from Reviewer 8eB8

Excellent review:

- think differently, not just by picking on the details, but conceptually different
- clarify the work from a broad perspective, even beyond what authors have thought about
- aim to improve the work, not for the sake of attack

vector search (algorithms, indices) ⇒ RAG (VDBMS) systems

From vector search to vector database systems (VDBMS)

VDBMS: Vector DataBase Management Systems

a system for storing, managing and querying vectors

Why do we need VDBMS for vector search?

Why do we have RDBMS?

Performance: scalability, availability

Handling updates

Data governance

What defines a Database System (DBMS)?

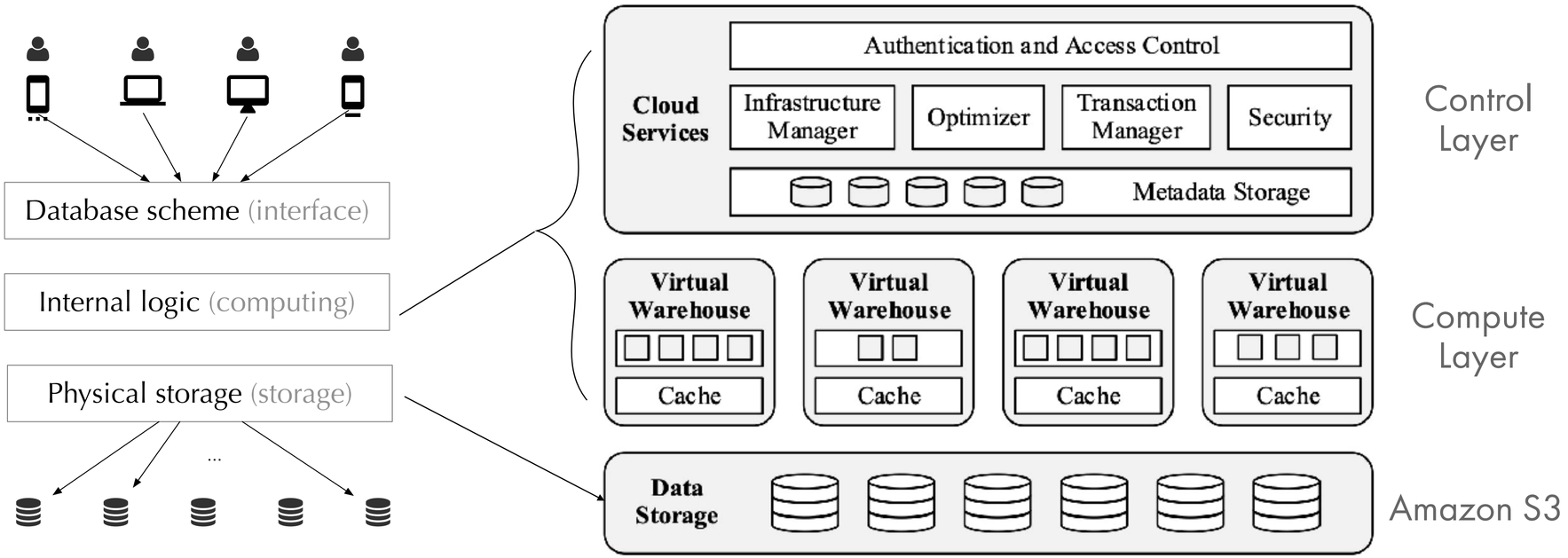
Characteristics

Logical-physical separation — just describe what you want; DBMS takes care of the rest

querying datasets larger than memory, efficiently — indexing, tiered storage/buffer pool

data persistence and governance — updates, integrity/consistency, access control, ...

(Week 9) **isolation**: an illusion of a single user — dealing with concurrent updates & queries



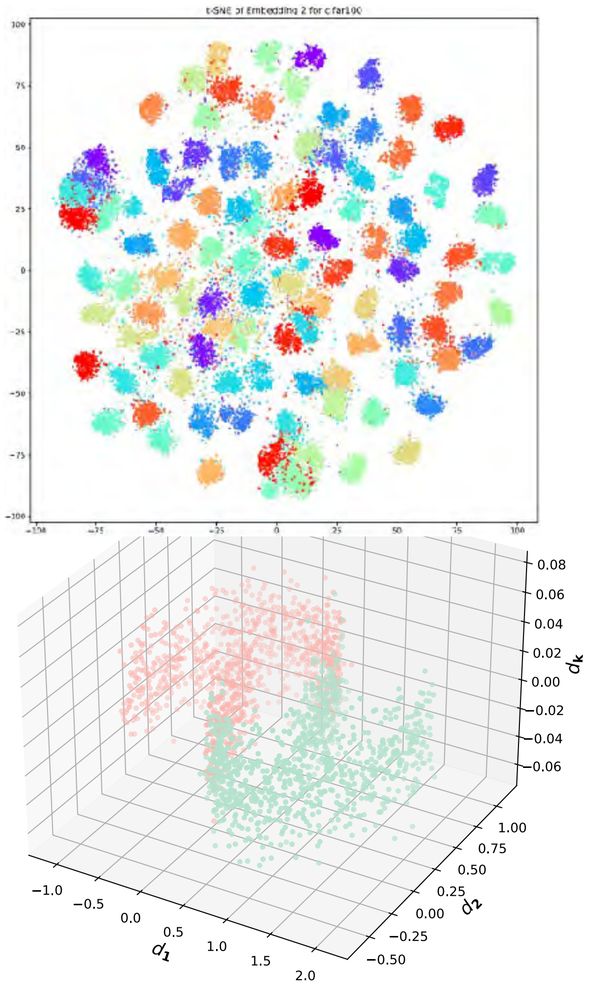
DBMS

VDBMS

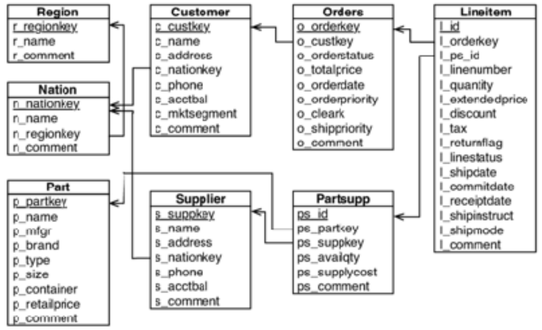
(RAG)

Dageville et al. *The Snowflake Elastic Data Warehouse*. SIGMOD (2016)

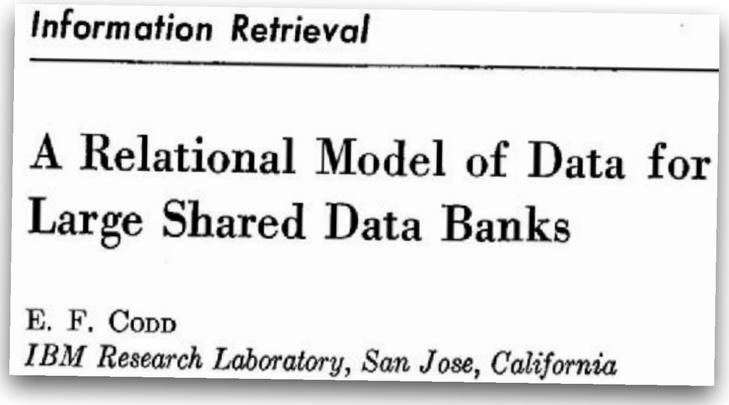
Why is building a VDBMS hard?



VS



	CustomerID	SalesOrderID	OrderDate	TotalDue	Total
1	11000	43793	2005-07-22 00:00:00.000	3756.989	9115.1341
2	11000	51522	2007-07-22 00:00:00.000	2587.8769	9115.1341
3	11000	57418	2007-11-04 00:00:00.000	2770.2682	9115.1341
4	11001	43767	2005-07-18 00:00:00.000	3729.364	7054.1875
5	11001	51493	2007-07-20 00:00:00.000	2674.0227	7054.1875



Information Retrieval

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Queries: Inherently **ambiguous** vs. **formal** semantics (SQL rooted in first-order logic)

Index: **distribution**-based, approximate (IVF, HNSW) vs. **attribute**-based exact (B-tree)

Reference VDBMS design

DB-Engines Ranking of Vector DBMS

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

This is a partial list of the complete ranking showing only vector DBMS.

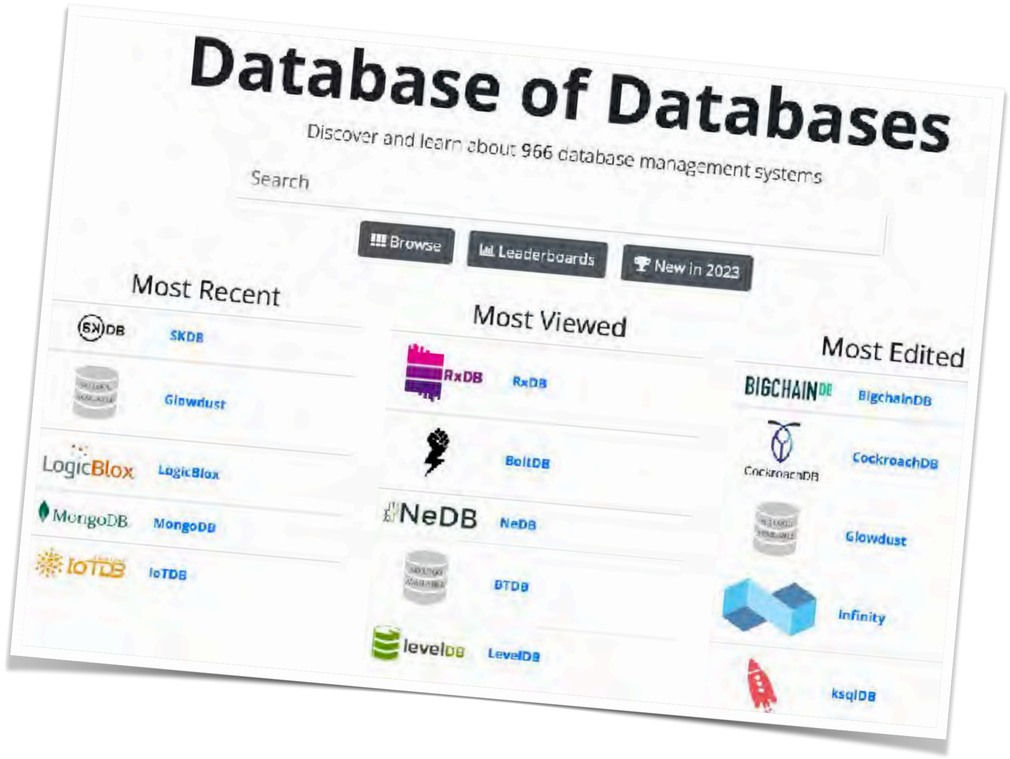
Read more about the method of calculating the scores.



trend chart

☑ include secondary database models — 30 systems in ranking, February 2025

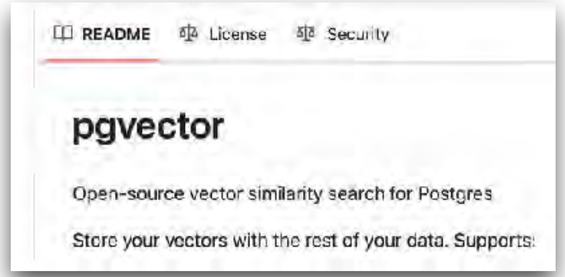
Rank Feb 2025	Rank Jan 2025	Rank Feb 2024	DBMS	Database Model	Score Feb 2025	Score Jan 2025	Score Feb 2024
1.	1.	1.	Oracle	Relational, Multi-model	1254.82	-3.93	+13.38
2.	2.	2.	PostgreSQL	Relational, Multi-model	659.62	-3.79	+30.21
3.	3.	3.	MongoDB	Document, Multi-model	396.63	-5.87	-23.73
4.	4.	4.	Redis	Key-value, Multi-model	157.91	+4.55	-2.80
5.	5.	5.	Elasticsearch	Multi-model	134.63	-0.29	-1.11
6.	6.	6.	Apache Cassandra	Wide column, Multi-model	102.58	+3.39	-6.69
7.	7.	↑ 9.	OpenSearch	Search engine, Multi-model	16.85	+0.06	+2.01
8.	8.	↓ 7.	Couchbase	Multi-model	15.63	-0.38	-4.85
10.	10.	10.	Amazon Aurora	Relational, Multi-model	8.28	+0.17	-1.75
11.	11.	11.	Kdb	Multi-model	6.79	-0.16	-1.49
12.	12.	↑ 15.	Microsoft Azure AI Search	Search engine, Multi-model	5.27	-0.18	-0.73
13.	13.	13.	Aerospike	Multi-model	5.09	+0.05	-2.00
14.	14.	↓ 12.	Datastax Enterprise	Wide column, Multi-model	4.22	-0.26	-3.43
15.	15.	↓ 14.	SingleStore	Relational, Multi-model	4.19	+0.29	-1.92
16.	16.	↑ 17.	Pinecone	Vector	2.94	-0.26	-0.42
17.	17.	↑ 21.	Milvus	Vector	2.77	+0.01	+1.30
18.	18.	↓ 16.	DolphinDB	Multi-model	2.38	-0.09	-1.68
19.	↑ 20.	↑ 22.	Qdrant	Vector	1.94	+0.00	+0.59
20.	↓ 19.	↓ 18.	Chroma	Vector	1.84	-0.11	-0.29
21.	21.	↓ 19.	Weaviate	Vector	1.65	+0.15	-0.43
22.	22.	↓ 20.	ObjectBox	Multi-model	1.06	0.00	-0.45
23.	23.	23.	AllegroGraph	Multi-model	0.70	-0.02	-0.39
24.	24.	↑ 26.	Vespa	Multi-model	0.69	+0.01	+0.09
25.	25.	↓ 24.	CrateDB	Multi-model	0.66	+0.02	-0.25
26.	26.	↑ 28.	Deep Lake	Vector	0.32	0.00	+0.07
27.	27.	27.	MyScale	Multi-model	0.27	-0.02	-0.02
28.	28.	↓ 25.	Vald	Vector	0.24	-0.03	-0.52
29.	29.	29.	JaguarDB	Multi-model	0.00	±0.00	±0.00
29.	29.		SvectorDB	Vector	0.00	±0.00	
29.	29.	29.	Transwarp Hippo	Vector	0.00	±0.00	±0.00



Reference VDBMS design

One size fits all

(vector as a data type in RDBMS)


README License Security

pgvector

Open-source vector similarity search for Postgres

Store your vectors with the rest of your data. Supports:

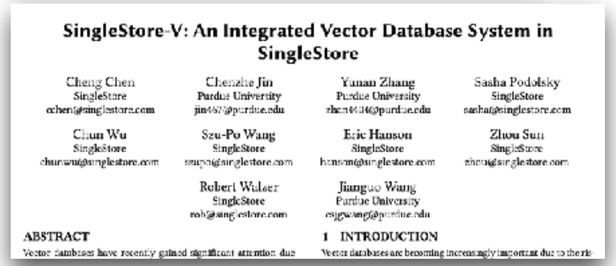

OCI About Services Solutions Pricing Partners Resources

Oracle United Kingdom > Database >

Oracle AI Vector Search

Easily bring AI-powered similarity search to your business data without managing and integrating multiple databases or compromising functionality, security, and consistency. AI Vector Search enables searching both structured and unstructured data by semantics or meaning, and by values, enabling ultra-sophisticated AI search applications. Native AI vector search capabilities can also help large language models (LLMs) deliver more accurate and contextually relevant results for enterprise use cases using retrieval-augmented generation (RAG) on your business data.

Try Oracle Database Free


SingleStore-V: An Integrated Vector Database System in SingleStore

Cheng Chen, SingleStore, cchen@singlestore.com
Chenzhe Jin, Purdue University, jin667@purdue.edu
Yunan Zhang, Purdue University, zhan4046@purdue.edu
Sasha Podolsky, SingleStore, sasha@singlestore.com
Chun Wu, SingleStore, chunwu@singlestore.com
Szu-Po Wang, SingleStore, szupo@singlestore.com
Eric Hanson, SingleStore, hanson@singlestore.com
Zhou Sun, SingleStore, zhou@singlestore.com
Robert Walzer, SingleStore, rob@singlestore.com
Jianguo Wang, Purdue University, csjgwang@purdue.edu

ABSTRACT

Vector databases have recently gained significant attention due

1 INTRODUCTION

Vector databases are becoming increasingly important due to their

One size does not fit all

(specialized design)


Manu: A Cloud Native Vector Database Management System

Rentong Guo[†*], Xiaofan Luan[*†], Long Xiang[‡*], Xiao Yan[‡*], Xiaomeng Yi[†*], Jigao Luo[†§]
Qianya Cheng[†], Weizhi Xu[†], Jiarui Luo[‡], Frank Liu[†], Zhenshan Cao[†], Yanliang Qiao[†], Ting Wang[†]
Bo Tang[‡], Charles Xie[†]

[†]Zilliz [*]Southern University of Science and Technology [§]Technical University of Munich
[†]{firstname.lastname}@zilliz.com
[‡]{xiangl3@mail., yanx@, 11911419@mail., tangb3@}sustech.edu.cn [§]jigao.luo@tum.de

ABSTRACT

PVLDB Artifact Availability:


Industrial Track Paper SIGMOD '21, June 20–25, 2021, Virtual Event, China

Milvus: A Purpose-Built Vector Data Management System

Jianguo Wang*, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, Charles Xie

* Zilliz & Purdue University
* *csjgwang@{zilliz.com; purdue.edu}*

Zilliz
{firstname.lastname}@zilliz.com

ABSTRACT

Recently, there has been a pressing need to manage high-dimensional vector data in data science and AI applications. This trend is fueled

KEYWORDS

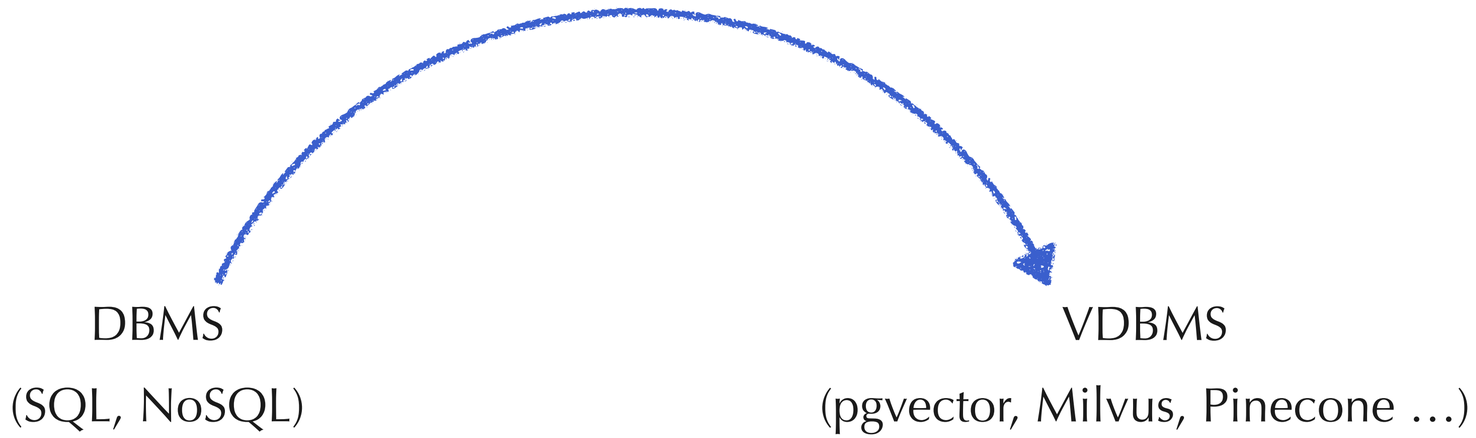
Vector database; High-dimensional similarity search; Heterogeneous computing; Data science; Machine learning


AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data

Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou,
Chaoqun Zhan, Feifei Li, Yuanzhe Cai

Alibaba Group
{chuangxian.wcx,binwu.wb,sh.wang,json.lrj,lizhe.zcq,lifeifei,yuanzhe.cyz}
@alibaba-inc.com

connecting the dots



1. updates
2. query optimisation

Case 1: dealing with updates — learning from past (DBMS)

“Traditional” SQL DBMS:

data: tuples

access pattern: in-order traversal

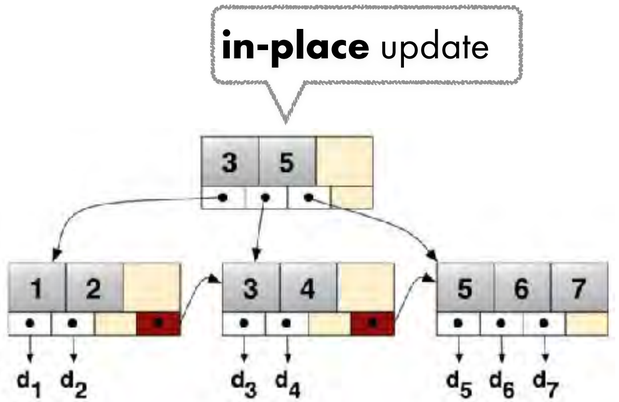
ORACLE

IBM DB2

Microsoft SQL Server

MySQL

PostgreSQL



B+-tree (credit: wiki)

“Modern” SQL+NoSQL DBMS:

data: column-family, key-value

access pattern: write-intensive (random, large)

snowflake

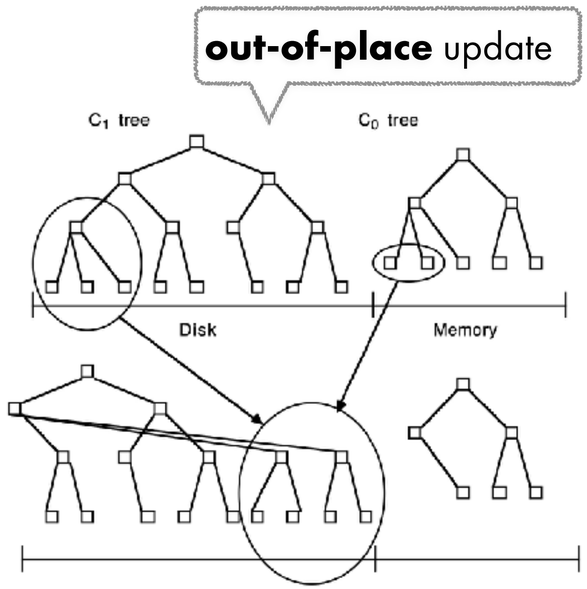
APACHE HBASE

cassandra

Google BigTable

levelDB

MyRocks



O'Neil et al. *The Log-Structured Merge-Tree*. (1996)

Case 1: dealing with updates — **Out-of-place** update via LSM-tree

Log Structured Merge Tree (LSM-tree)

avoid frequently random writes

flexible trade-off: read efficiency & consistency

How does LSM-tree achieve these, conceptually?

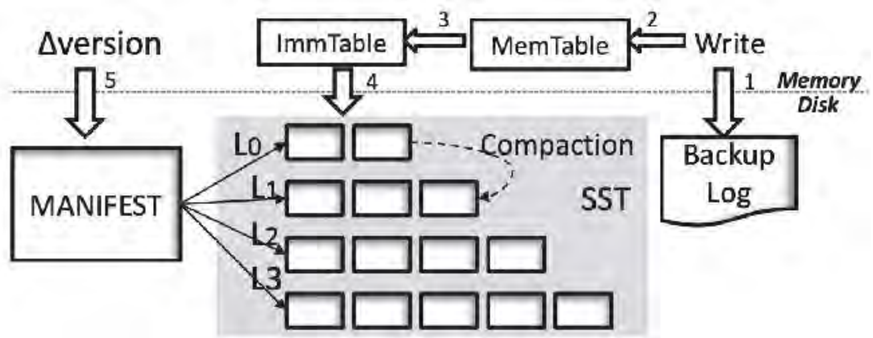
buffer writes in-memory (MemTable)

flash MemTable periodically to disk as a segment

merge segments periodically into larger ones

Three types of disk writes

- write-ahead-log (WAL)
- compaction
- metadata update (LSM index)



[Mei et al. SoCC'17]

Case 1: dealing with updates — **Out-of-place** update via LSM-tree

Read in LSM-tree

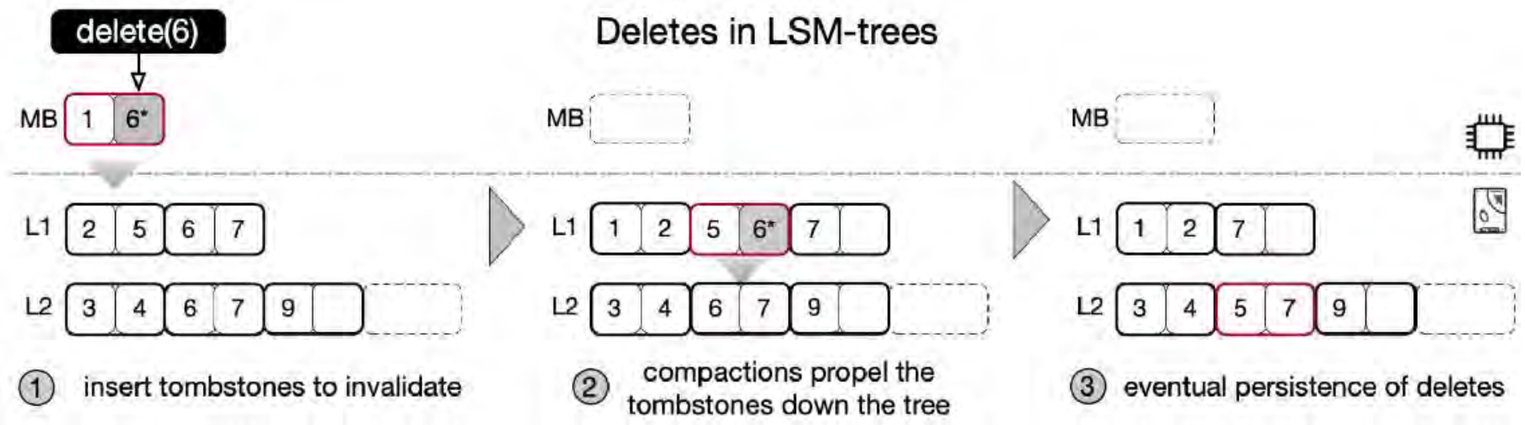
starting from MemTable

look for read key through levels (optimized via BloomFilter)

Deletion in LSM-tree

logically deleted by inserting **tombstones**, a special key-value entry

deletions are **physically** "implemented" during **compaction**



credit: https://disc-projects.bu.edu/lethe/

Connecting the dots: LSM for vector updates

Out-of-place update (batched updates)

maintain two data structures

updates: fast-writeable slow-readable structure (FWSR)

search: slow-writeable fast-readable structure (SWFR)

Reconcile FWSR into SWFR at a convenient time

MemTable in LSM-tree

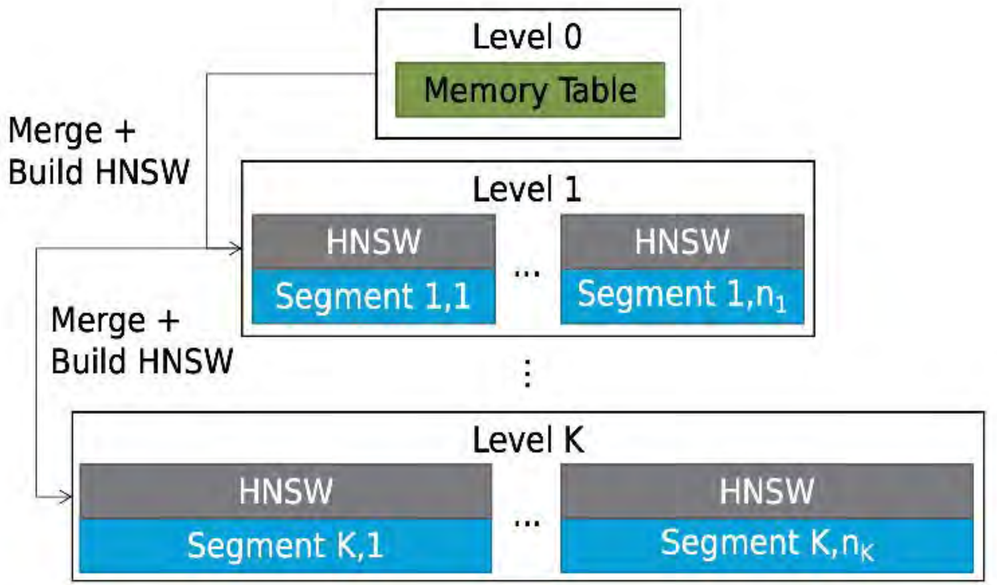
segment file (sorted runs)

compaction in LSM

During segment compaction

- build **per-seg** HNSW
- reconcile tombstones
- index merger [SingleStore-V]
- snapshot isolation

can replace HNSW with other index



index update in Milvus (credit: Pan et al. SIGMOD'24 tutorial)

Case 2: query optimisation (when vector search does not suffice)

Hybrid (predicate) search queries

items:

pname	pid	price	stock	category_id	embedding

```
SELECT * FROM items WHERE category_id = 123 ORDER BY embedding <-> '[3,1,2]' LIMIT 5;
```

Multi-vector search queries

items:

pname	pid	price	stock	category_id	embedding1	embedding2

```
SELECT * FROM items WHERE category_id = 123 ORDER BY
```

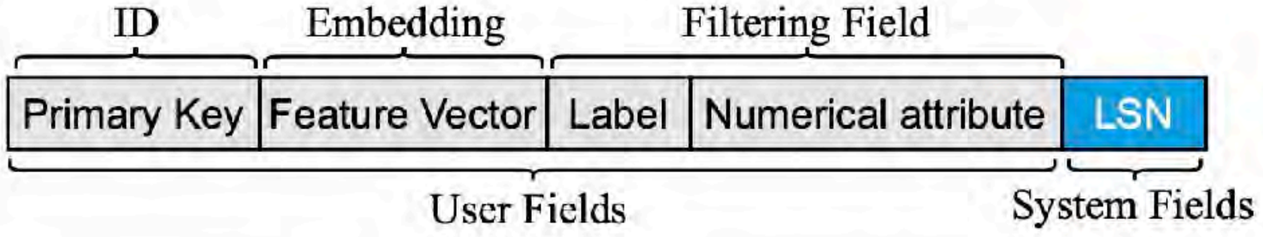
$g(\text{embedding1} <-> '[3,1,2]' \text{ AND } \text{embedding2} <-> '[4,3,1]')$ LIMIT 5

$g(\text{dist}_1, \text{dist}_2) = w_1 * \text{dist}_1 + w_2 * \text{dist}_2$

Predicate vector search



Hybrid query in Alibaba [Wei et al. VLDB'20]



Manu's schema [Guo et al. VLDB'22]

Predicate vector search

items:	pname	pid	price	stock	category_id	embedding

```
SELECT * FROM items WHERE category_id = 123 ORDER BY embedding <-> '[3,1,2]' LIMIT 5;
```

pre-filering

1) retrieve the set S of all tuples with category_id = 123 first
2) vector search over S

post-filtering

1) vector search first, find a set S of tuples with vectors near '[3,1,2]'
2) return tuples with category_id = 123

Which approach shall we pick?

Query planning in RDBMS



$Q = R \bowtie_B S \bowtie_C T$

1000 1000 ⋈ 100

join order 1: $D_1 = R \bowtie_B S, \quad D_2 = D_1 \bowtie_C T$

join order 2: $D_1 = S \bowtie_C D, \quad D_2 = D_1 \bowtie_B R$

10 10 ⋈ 100

nested-loop (via index scans)

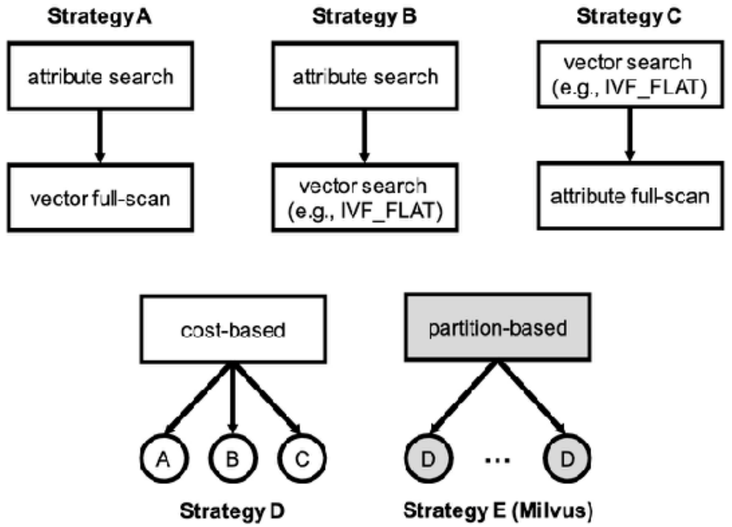
hash join

merge join

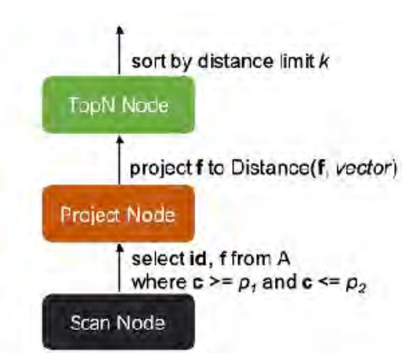
cost-based plan selection in RDBMS

- join ordering
- physical operator selection
- based on selectivity estimation

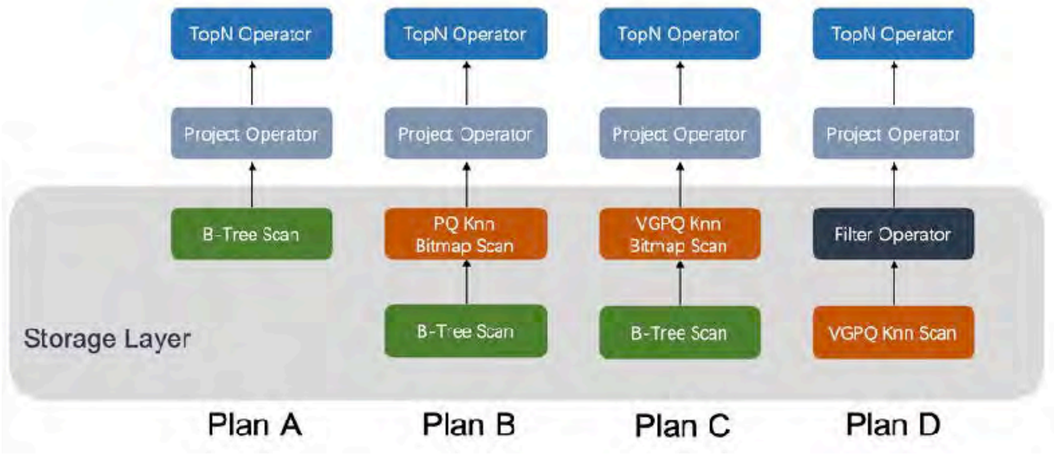
Connecting the dots: cost-based plan selection for vector search



plan selection in Milvus [Wang et al. SIGMOD'21]



query



query plans

cost-based plan selection in Alibaba's ADBV [Wang et al. VLDB'20]

extended reading (in-search filtering):

ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data

LIANA PATEL, Stanford University, USA
PETER KRAFT, DBOS, Inc., USA
CARLOS GUESTRIN, Stanford University, USA
MATEI ZAHARIA, UC Berkeley, USA

Multi-vector search

items:

pname	pid	price	stock	category_id	embedding1	embedding2

```
SELECT * FROM items WHERE category_id = 123 ORDER BY
```

$g(\text{embedding1} <-> '[3,1,2]' \text{ AND embedding2} <-> '[4,3,1]')$ LIMIT 5

$$g(\text{dist}_1, \text{dist}_2) = w_1 * \text{dist}_1 + w_2 * \text{dist}_2$$

Solution 1:

top-k search over embedding1 and embedding2 independently; join the results

bad recall: top-k(embedding1) $\bowtie$ top-k(embedding2) $\neq$ top-k(embedding1$\bowtie$embedding2)

Solution 2:

concatenate vectors first (embedding1$\bowtie$embedding2); top-k over concatenated vectors

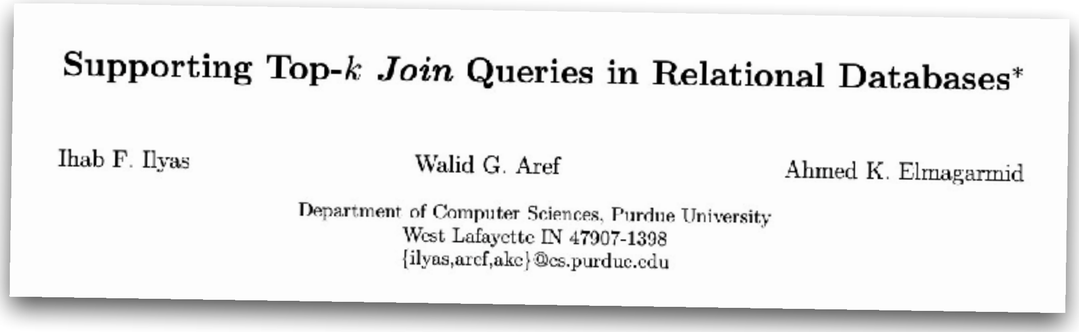
dot-product only: $g(\text{dist}_1, \text{dist}_2) = w_1 * \text{dist}_1 + w_2 * \text{dist}_2$

$$w_1(x_1, x_2) \cdot (x_1', x_2') + w_2(y_1, y_2) \cdot (y_1', y_2') = (w_1x_1, w_1x_2, w_2y_1, w_2y_2) \cdot (x_1', x_2', y_1', y_2')$$

Top-k join queries in RDBMS

Top-k join queries

the rank-join operator



Supporting Top-k *Join* Queries in Relational Databases*

Ihab F. Ilyas Walid G. Aref Ahmed K. Elmagarmid

Department of Computer Sciences, Purdue University
West Lafayette IN 47907-1398
{ilyas,aref,ake}@cs.purdue.edu

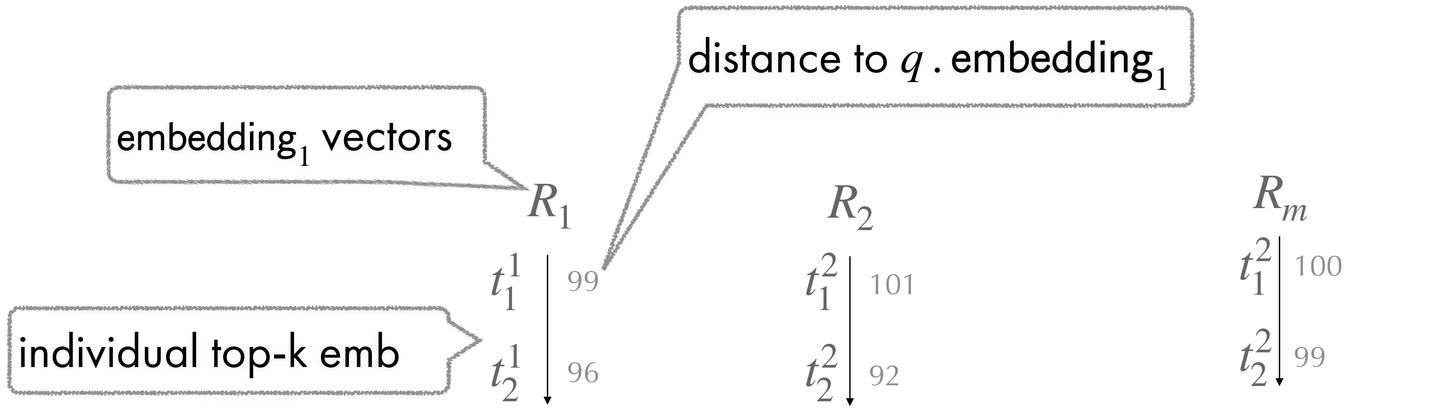
```
SELECT *
FROM R_1, R_2, ..., R_m
WHERE join_condition(R_1, R_2, ..., R_m)
ORDER BY f(R_1.score, R_2.score, ..., R_m.score)
STOP AFTER k;
```

tuple	R_1	score	R_2		$\cdots$	R_m	
t_1^1	↓	99	t_1^2	101		t_1^2	100
t_2^1	↓	96	t_2^2	92		t_2^2	99

terminate if $f(\text{ans}_k) \geq \min(f(t_i^1, t_1^2, \ldots, t_1^m), f(t_1^1, t_i^2, \ldots, t_1^m), \ldots, f(t_1^1, t_1^2, t_i^m))$

early termination property: #-tuples accessed is independent of dataset size

Connecting the dots: top-k join for multi-vector search



terminate if $f(\text{ans}_k) \geq \min(f(t_i^1, t_1^2, \ldots, t_1^m), f(t_1^1, t_i^2, \ldots, t_1^m), \ldots, f(t_1^1, t_1^2, t_i^m))$

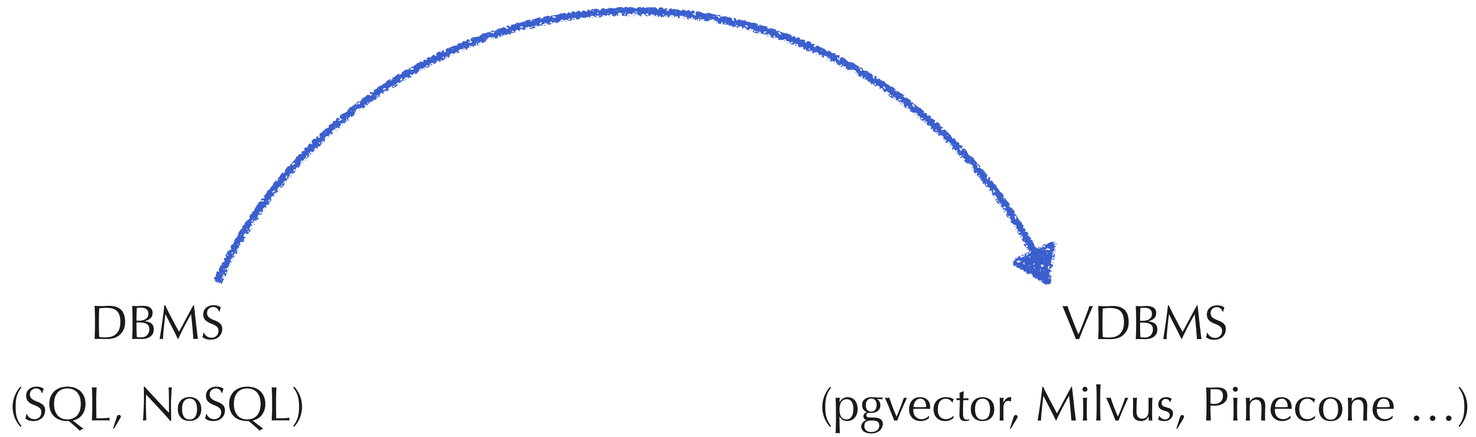
A little catch:

big overhead for **getNext(embedding$_i$)** w.r.t. distance to q . **embedding$_i$**

remedy: from **getNext(embedding$_i$)** to **getNext^K(embedding$_i$)**

exactly how multi-vector search is implemented in Milvus [Wang et al. SIGMOD'21]

1. LSM-based vector index updates
2. cost-based predicate vector search
3. rank-join based multi-vector search



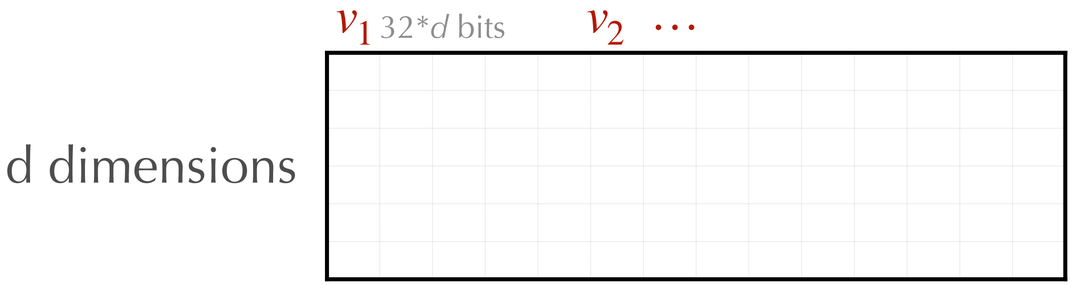
Anything not mimicking a traditional DBMS?

Vector quantisation

Quantisation:

aim: reduce the size of vectors (and consequently, their indices)

idea: bin (groups of) dimensions' values separately into discrete buckets



PCA: reduce height (d)

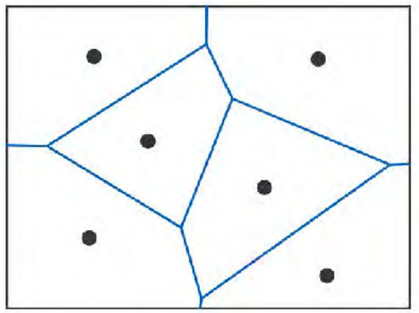
quantisation: reduce width

k-means as a quantiser: intended application of Lloyd's algorithm in 1957 at Bell Labs

all vectors in a cluster are encoded by the same centroid

replace centroid vectors with **centroid IDs**

codebook: map centroid IDs to centroid vectors

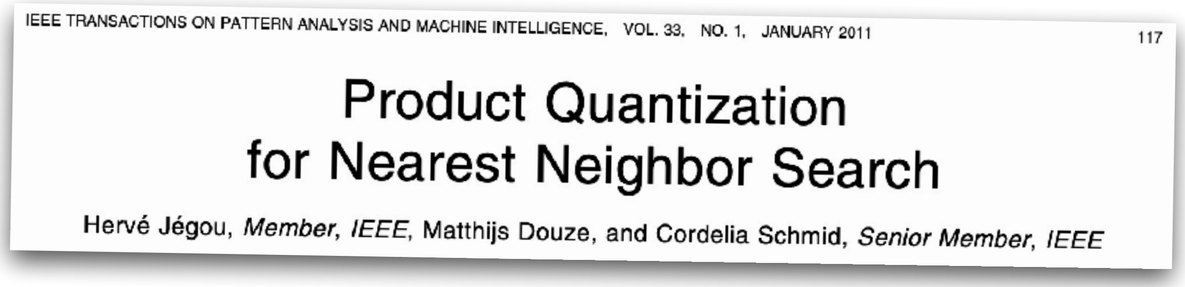


size reduction: D (of size nd) vs codebook (of size kd) + $k \log_2 k$

Product quantisation

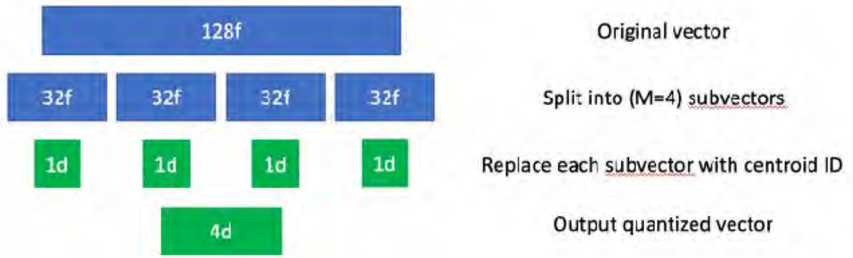
Product quantisation:

- partition vectors into subvectors
- quantise subvectors independently via k-means
- concatenate centroid IDs into compressed (quantised) vectors
- can be incorporated into vector index, e.g., IVF-PQ

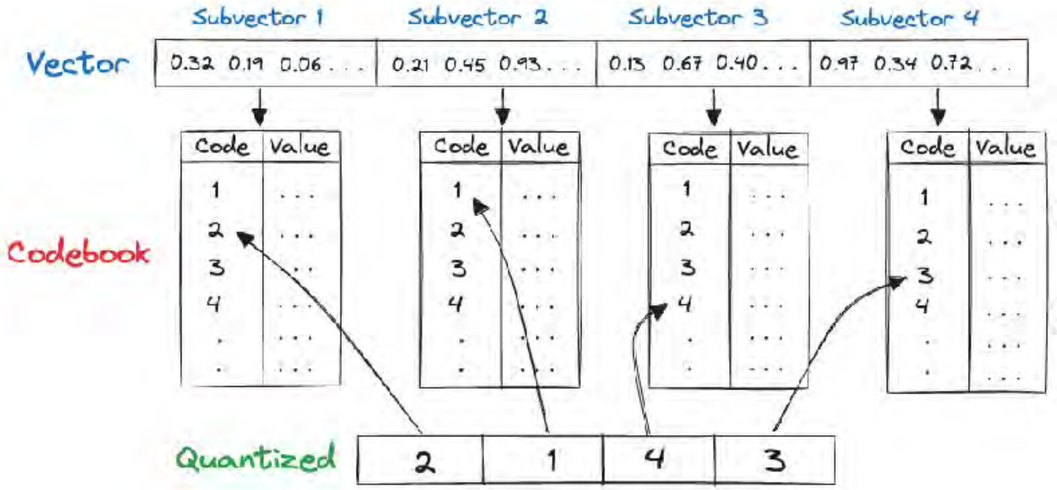

IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 33, NO. 1, JANUARY 2011 117

Product Quantization
for Nearest Neighbor Search

Hervé Jégou, *Member, IEEE*, Matthijs Douze, and Cordelia Schmid, *Senior Member, IEEE*



credit: https://zilliz.com/learn/scalar-quantization-and-product-quantization



credit: https://lancedb.github.io/lancedb/concepts/index_ivfpq/#product-quantization

More quantisations

Further reading on vector quantisation:

- Faiss: Billion-Scale Similarity Search with GPUs
- Optimized Product Quantization (OPQ) [Ge et al., 2014]
- Anisotropic Vector Quantization (ScaNN) [Guo et al., 2019]
- Anisotropic Additive Quantization [Zhang et al., 2022]
- RaBitQ / Extended RaBitQ [Gao & Long, 2024]

Reading on quantisation applied to LLMs & KV Cache

- SmoothQuant (Xiao et al., ICML 2023), QuaRot (Ashkboos et al., NeurIPS 2024),
- SpinQuant (Liu et al., ICLR 2025), KVQuant (Hooper et al., NeurIPS 2024), TurboQuant (Zandieh et al., ICLR 2026), LLM.int8 (Dettmers et al., NeurIPS 2022)