DHBW Stuttgart Campus Horb

Vector Databases: A Beginner's Guide

Olaf Herden
Baden-Wuerttemberg Cooperative State University
Stuttgart Campus Horb
o.herden@hb.dhbw-stuttgart.de

The 2025 IARIA Annual Congress on Frontiers in Science, Technology, Services, and Applications
IARIA Congress 2025
July 06, 2025 to July 10, 2025 - Venice, Italy

Speaker

- Olaf Herden
- Cooperative State University Baden-Wuerttemberg
- Since 2002 Professor for Computer Science
- Research interests:
  - Database Systems
  - Data Analysis
  - Data Mining/Machine Learning
  - Learning to code

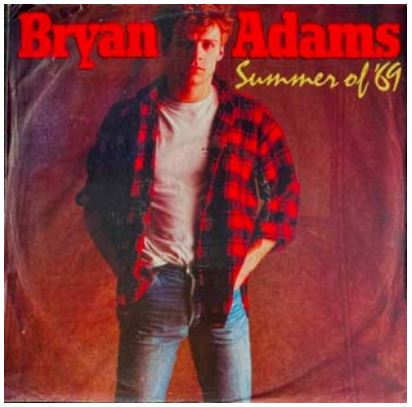
- LinkedIn: https://www.linkedin.com/in/olaf-herden-896a14157/
- Email: o.herden@hb.dhbw-stuttgart.de

Overview

- Introduction
- Similarity
- Searching
- Indexing
- Vector Database Solutions
- Summary and Outlook

Motivation (I)

GreMuSt - Great Music Streaming



Summer Of 69
Bryan Adams

Recommended for you:

Born to run
Bruce Springsteen

Run to you
Bryan Adams

The boys of summer
Don Henley

You give love a bad name
Bon Jovi

[Images from https://www.discogs.com/]

Motivation (II)

TGIS - **T**he **G**reat **I**mage **S**earch

Search

Result:

[All images on this slide are generated by ChatGPT.]

Motivation (III)

My company's RAG system for databases

I need a SQL statement to get the top 5 customers by total purchase amount this year.

"Based on company database structure (`customers`, `orders`, `order_items` tables), here is the SQL statement you can use:"

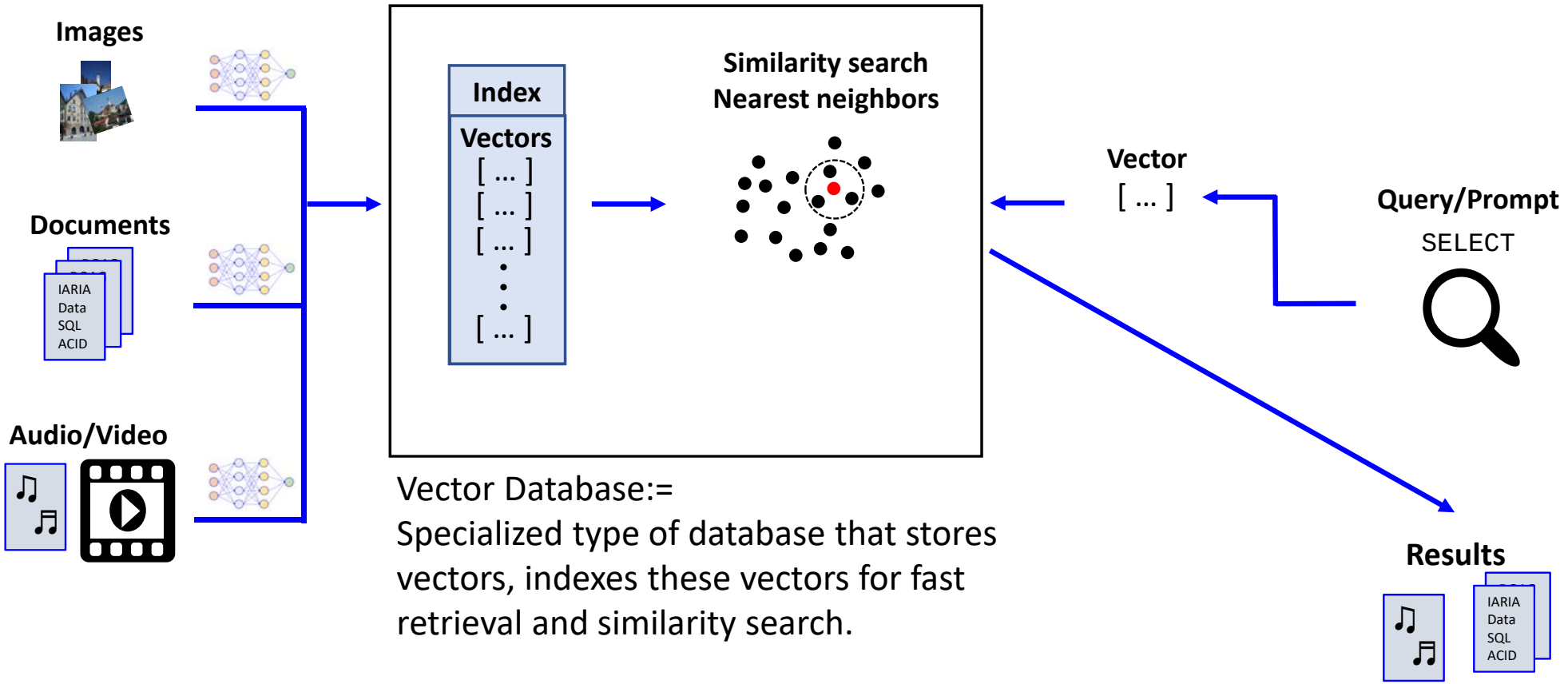
sql Kopieren Bearbeiten

```sql
SELECT c.customer_id, c.name, SUM(oi.quantity * oi.unit_price) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
WHERE o.order_date >= DATE_TRUNC('year', CURRENT_DATE)
GROUP BY c.customer_id, c.name
ORDER BY total_spent DESC
LIMIT 5;
```

"This query calculates the total purchase amount by summing product of quantity and unit price per order item, filtering only for orders placed this year."

[Example on this slide is generated by ChatGPT.]

Architecture overview



Vector Database:=
Specialized type of database that stores vectors, indexes these vectors for fast retrieval and similarity search.

Overview

- Introduction
- Similarity
- Searching
- Indexing
- Vector Database Solutions
- Summary and Outlook

Notation

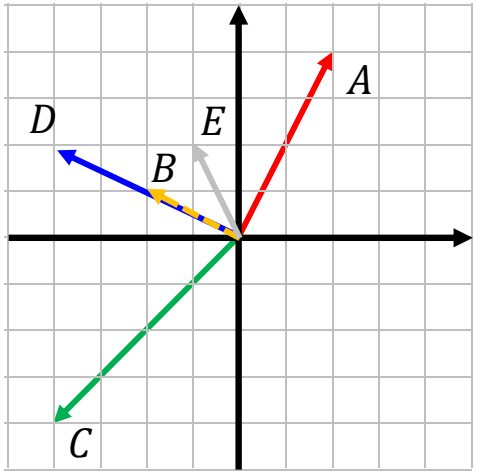
Vector

Components of the vector

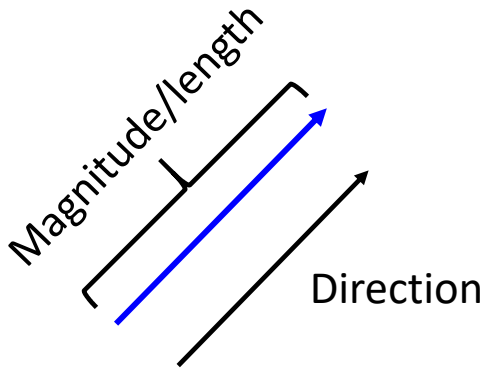
$$A = (a_1, \dots, a_n)$$

n: dimension of the vector

- Examples: $A = (2,4)$, $B = (-2,1)$, $C = (-4,-4)$, $D = (-4,2)$ and $E = (-1,2)$
- Illustrative: visualization in the plane



- Important properties:



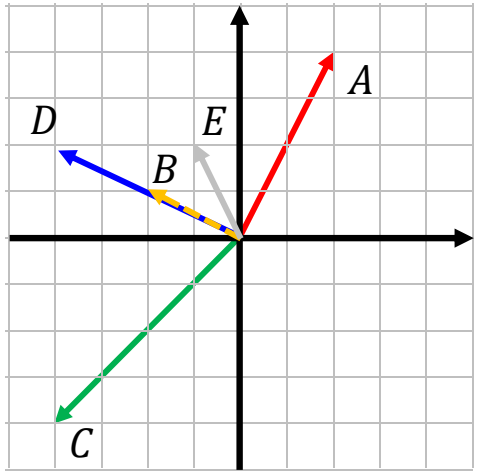
- A and D: same magnitude, different directions
- B and D: different magnitude, same direction

Definition: Euclidean norm

- Given vector $A = (a_1, \dots, a_n)$
- The Euclidean norm (L2 norm)

$$\|A\| := \sqrt{\sum_{i=1}^{n} {a_i}^2}$$

- Euclidean norm gives the magnitude of vector



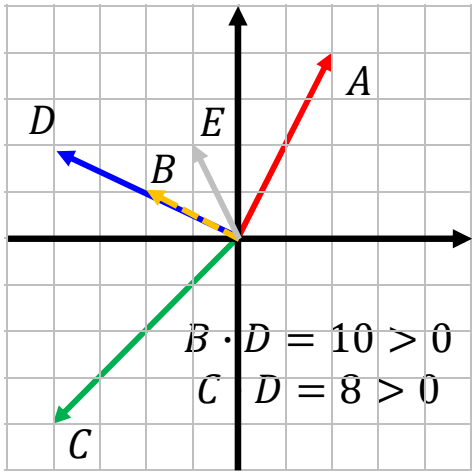
$$\|A\| \approx 4.47$$
$$\|B\| \approx 2.24$$
$$\|C\| \approx 5.66$$
$$\|D\| \approx 4.47$$
$$\|E\| \approx 2.24$$

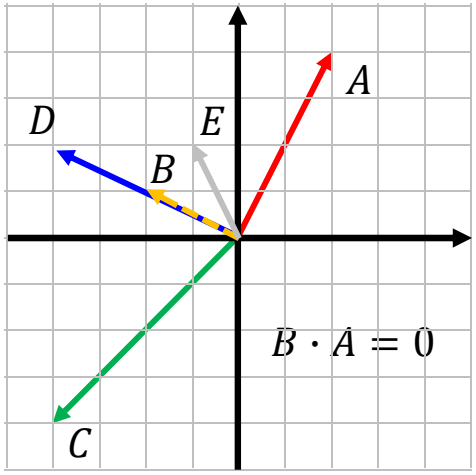
Definition: dot product

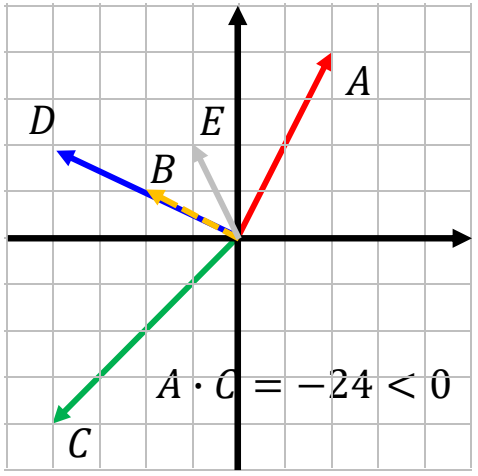
- Given two vectors $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$
- The dot product (synonym: inner product, scalar product)

$$A \cdot B := \sum_{i=1}^{n} a_i \cdot b_i$$

- Dot product is
  - positive when A and B show in same direction (angle $< 90°$)
  - 0 when A and B are orthogonal (angle $= 90°$)
  - negative when A and B point in mostly opposite directions ($90° <$ angle $\leq 180°$)

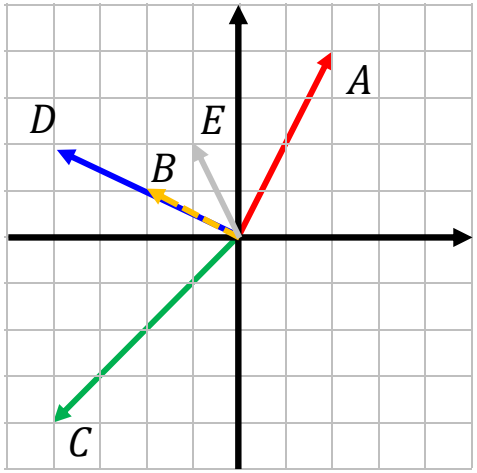






Similarity problem

- Problem:
  - Given two or more vectors
  - How similar (or different) are they?
- Example:
  - Is B more similar to D or to E?



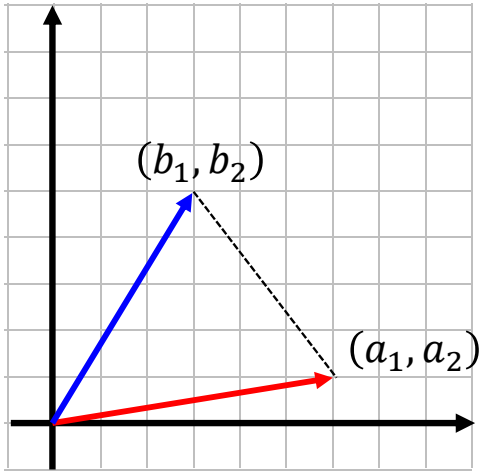
Euclidean distance (I)

- „Straight-line" distance
- Calculation:

$$euDist(A,B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

- Proof: Pythagorean theorem
- Example:
  - $A = (6,1)$ and $B = (3,5)$

$$euDist(A,B) = \sqrt{(3-6)^2 + (5-1)^2} = 5$$



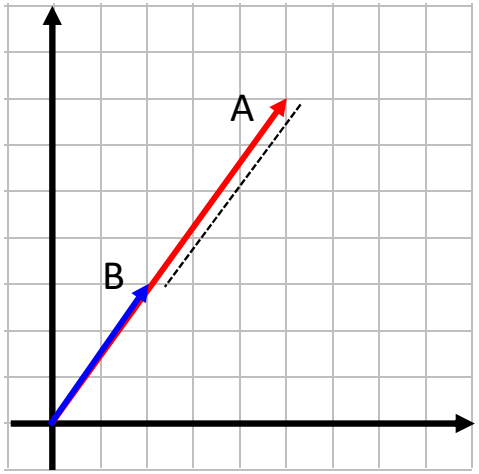
- n-dimensional case:
  - Let A, B vectors of dimension n $\Rightarrow euDist(A,B) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$

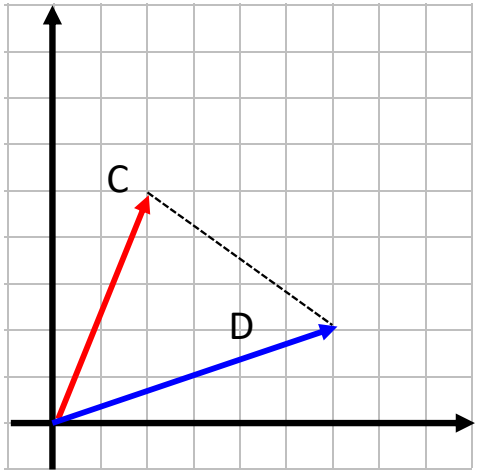
Euclidean distance (II)

- Works well, when direct distance is needed
- Problems/limits:
  - Sensitive to scale:
    - Components with different ranges $\Rightarrow$ Components with large scale dominate
    - Example: $\mathrm{A} = (2,15,8,6)$ and $\mathrm{B} = (4,75,7,7) \Rightarrow$ The second component dominates computation
  - Sensitive to outliers:
    - Outliers can disproportionately affect Euclidean distance calculations
    - Resulting in misleading results (especially problematic in clustering or nearest neighbor analyses)
    - Example: $\mathrm{A} = (1,2,3,4)$ and $\mathrm{B} = (2,3,4,100) \Rightarrow B$'s fourth component (outlier) distorts calculation

Euclidean distance (II)

- Considers only direct distance, e.g.:



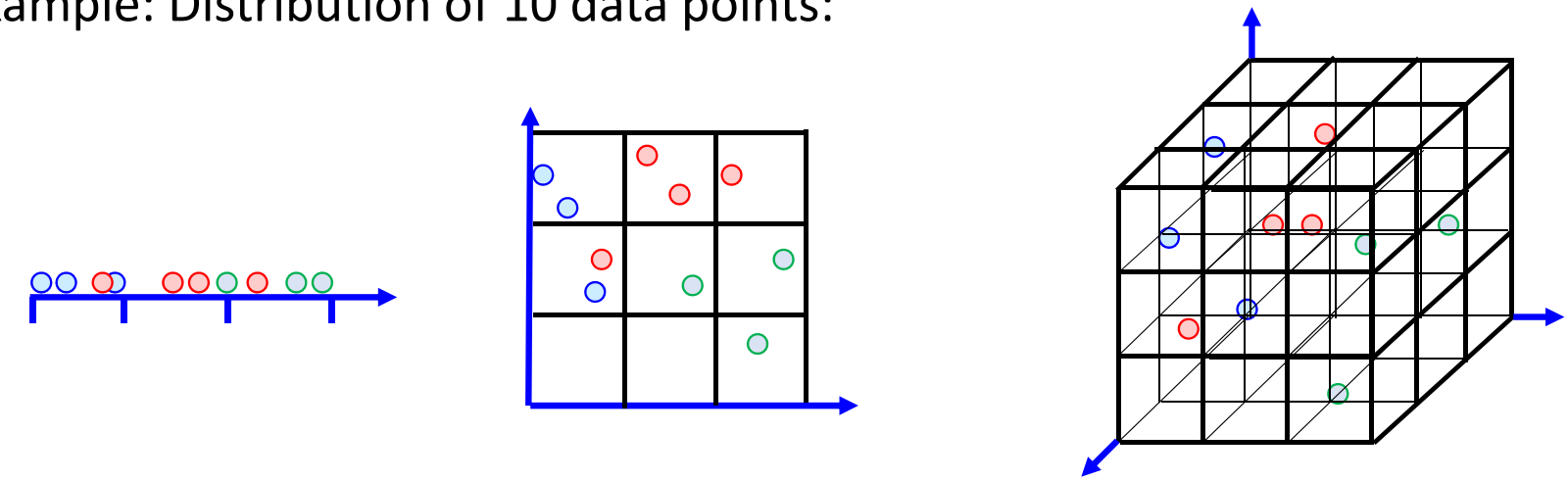


  - $euDist(A, B) = euDist(C, D)$
  - But: same similarity?

- Not suitable for high-dimensional data:
  - In high-dimensional spaces the distance between points becomes less meaningful as the number of dimensions increases
  - Leads to difficulties in distinguishing between points
  - “Curse of dimensionality"

Digression: Curse of dimensionality

- Example: Distribution of 10 data points:



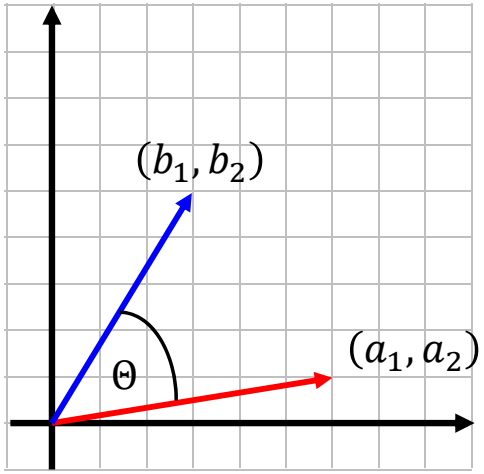
- In numbers:
  - One dimension: space with 100 data points „well“ covered
  - 10 dimensions $\Rightarrow$ 100 data points are isolated points, space not covered sufficiently
  - For covering analogous to one-dimensional case: $100^{10} = 10^{20}$ data points needed

Richard Bellman. Adaptive control processes: a guided tour. Princeton University Press, 1961.

Cosine similarity (I)

- Cosine of angle between two vectors
- Calculation:

$$cosSim(A,B) := \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} {a_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {b_i}^2}}$$



- Example:
  - $A = (6,1)$ and $B = (3,5)$

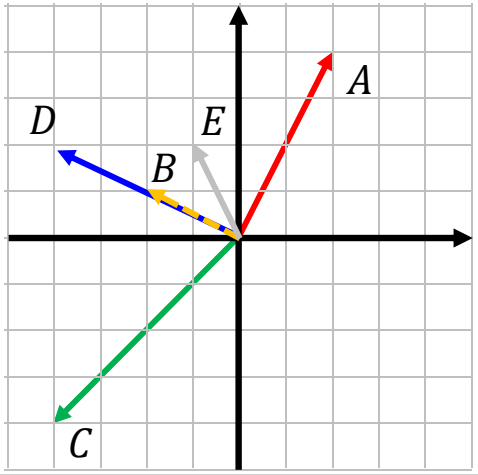
$$cosSim(A,B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{a_1 \cdot b_1 + a_2 \cdot b_2}{\sqrt{{a_1}^2 + {a_2}^2} \cdot \sqrt{{b_1}^2 + {b_2}^2}} = \frac{6 \cdot 3 + 1 \cdot 5}{\sqrt{6^2 + 1^2} \cdot \sqrt{3^2 + 5^2}} \approx 0.648466$$

- Cosine distance $cosDist$ is defined as $cosDist(A,B) := 1 - cosSim(A,B)$

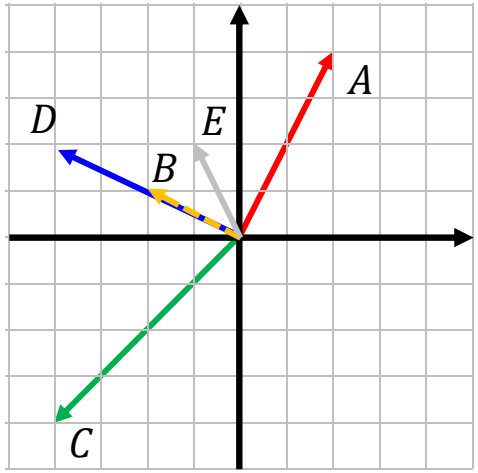
Cosine similarity (II)

- Cosine similarity
  - only considers direction of vectors
  - neglects magnitude of vectors
- Cosine similarity is
  - close to 1 when the angle between vectors is close to $0°$
  - Close to 0 when the angle between vectors is close to $90°$
  - Close to -1 when the angle between vectors is close to $180°$

e.g. $cosSim(B, E) = 0.8$
e.g. $cosSim(A, D) = 0$
e.g. $cosSim(A, C) \approx -0.9847$



Euclidean distance vs. cosine similarity



- Is B more similar to D or to E?
  - $euDist(B, D) = 2.24$, $euDist(B, E) = 1.41$
  - B is more similar to E when using Euclidean distance
  - $cosSim(B, D) = 1$, $cosSim(B, E) = 0.8$
  - B is more similar to D when using cosine similarity

Overview

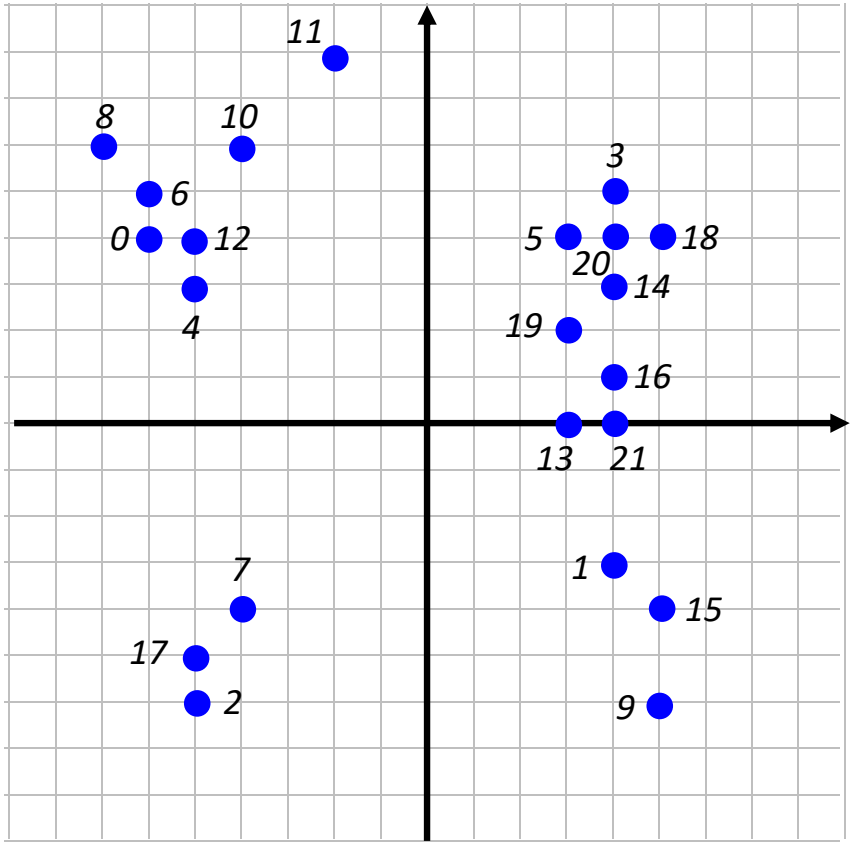
- Introduction
- Similarity
- Searching
- Indexing
- Vector Database Solutions
- Summary and Outlook

Sample data

- 22 two-dimensional integer vectors

no.	x	y
0	−6	4
1	4	−3
2	−5	−6
3	4	5
4	−5	3
5	3	4
6	−6	5
7	−4	−4
8	−7	6
9	5	−6
10	−4	6

no.	x	y
11	−2	8
12	−5	4
13	3	0
14	4	3
15	5	−4
16	4	1
17	−5	−5
18	5	4
19	3	2
20	4	4
21	4	0



Brute force k-NN-search (I)

- Input: data set V of vectors, search vector Q, k desired nearest neighbours, distance/similarity function
- Output: list L of k vectors, most similar to Q, ordered by similarity to Q
- Algorithm:
  - Initialize list L with k entries
  - Foreach $v \in V$:
    - Compute distance d between v and Q
    - If $d <$ k-th entry in $L \Rightarrow$ update L

Brute force k-NN-search (II)

- Example: search vector $Q = (-1, -1)$, $k = 3$, Euclidean distance

no.	x	y
0	−6	4
1	4	−3
2	−5	−6
3	4	5
4	−5	3
5	3	4
6	−6	5
7	−4	−4
8	−7	6
9	5	−6
10	−4	6

no.	dist
1	5.39
4	5.66
2	6.40

$d(Q, V_7) = 4.24$

$d < 3rd\ entry\ in\ L$

Update L:

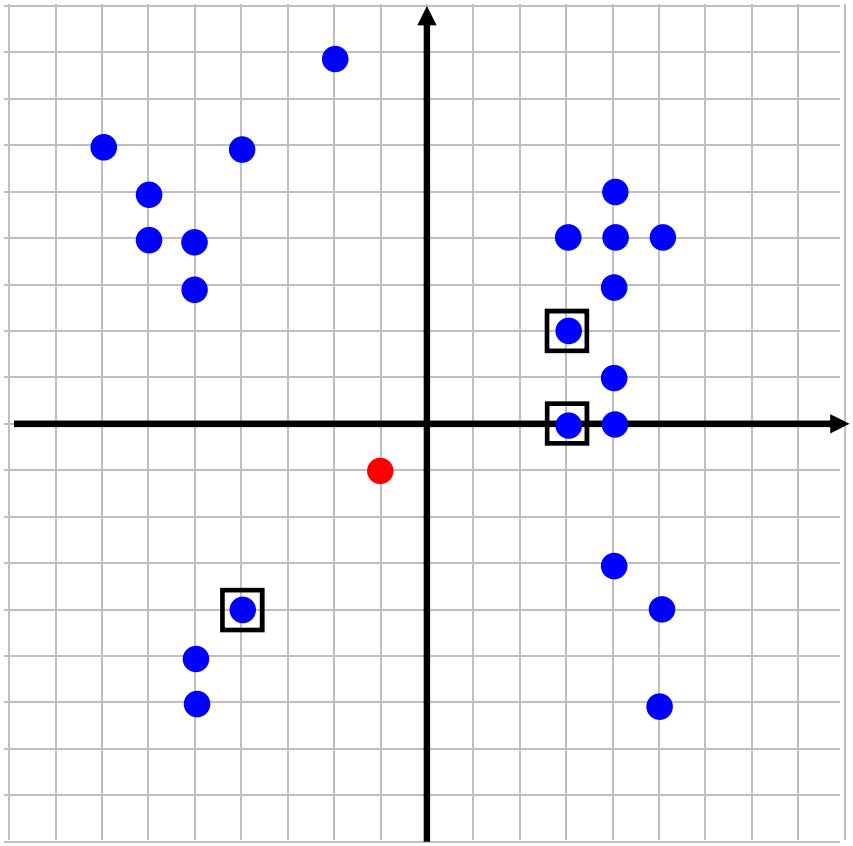
no.	dist
7	4.24
1	5.39
4	5.66

no.	x	y
11	−2	8
12	−5	4
13	3	0
14	4	3
15	5	−4
16	4	1
17	−5	−5
18	5	4
19	3	2
20	4	4
21	4	0

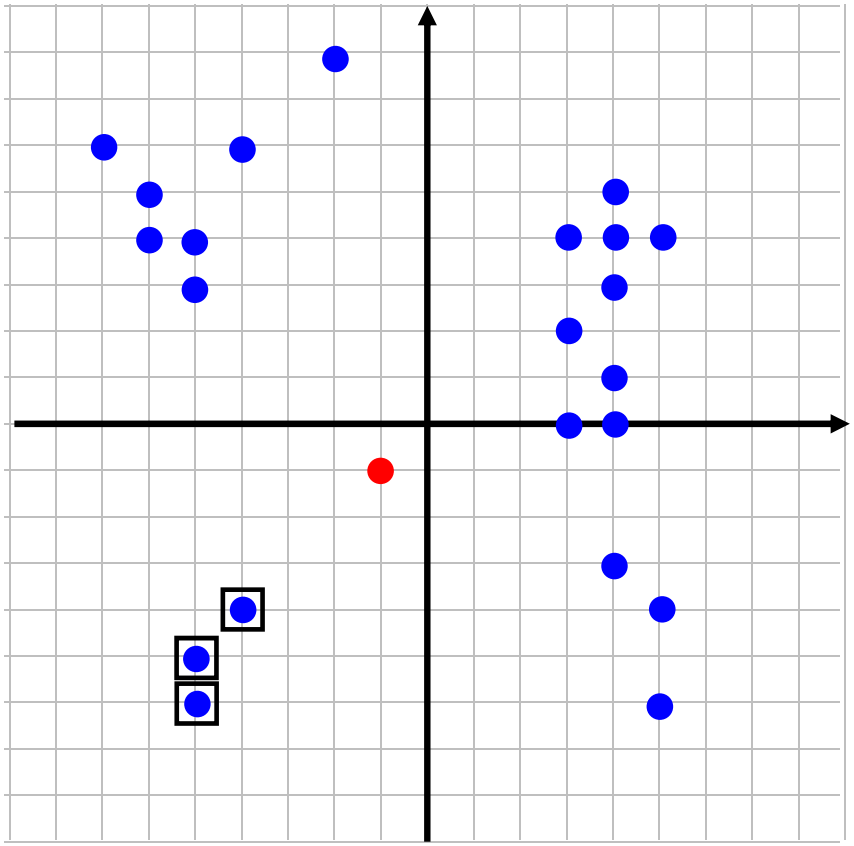
no.	dist
13	4.12
7	4.24
19	5.0

Brute force k-NN-search (III)

- Result Euclidean distance



- Result cosine similarity



Conclusion

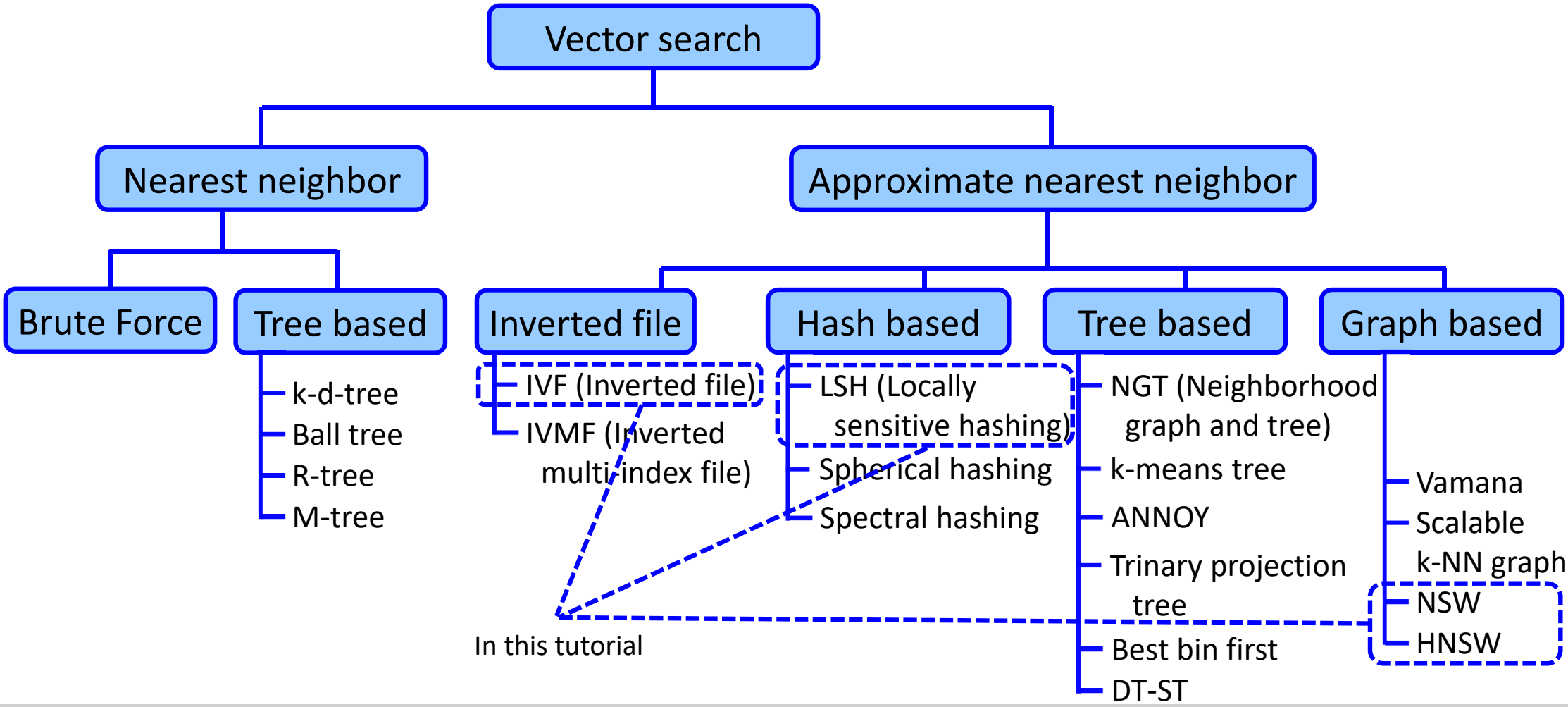
- Result is correct
- But computationally too complex
- Question: Do there exist other options searching faster, probably under loss of accuracy

- Answer: Yes, there are several index techniques!

Overview

- Introduction
- Similarity
- Searching
- Indexing
- Vector Database Solutions
- Summary and Outlook

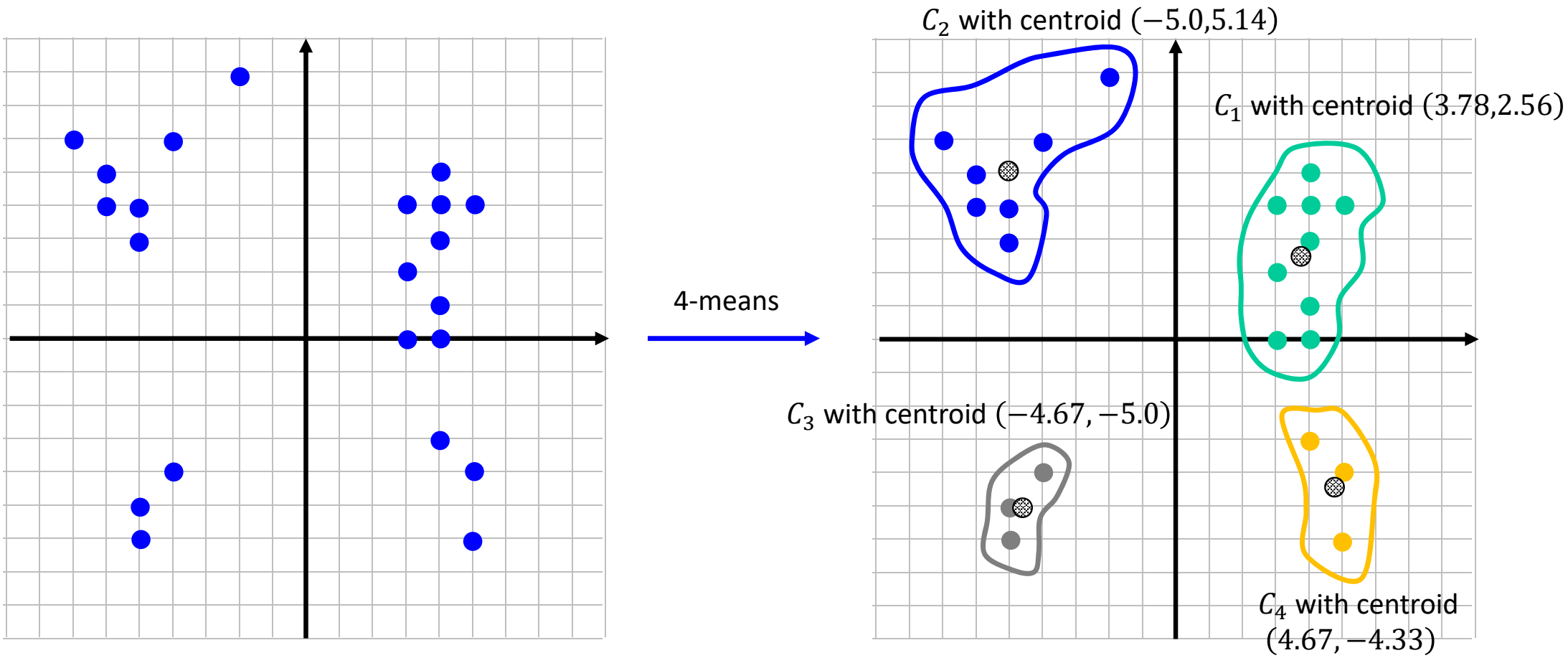
Overview


Vector search
Nearest neighbor
Approximate nearest neighbor
Brute Force
Tree based
Inverted file
Hash based
Tree based
Graph based
k-d-tree
Ball tree
R-tree
M-tree
IVF (Inverted file)
IVMF (Inverted multi-index file)
LSH (Locally sensitive hashing)
Spherical hashing
Spectral hashing
NGT (Neighborhood graph and tree)
k-means tree
ANNOY
Trinary projection tree
Best bin first
DT-ST
Vamana
Scalable k-NN graph
NSW
HNSW
In this tutorial

Inverted file (I)

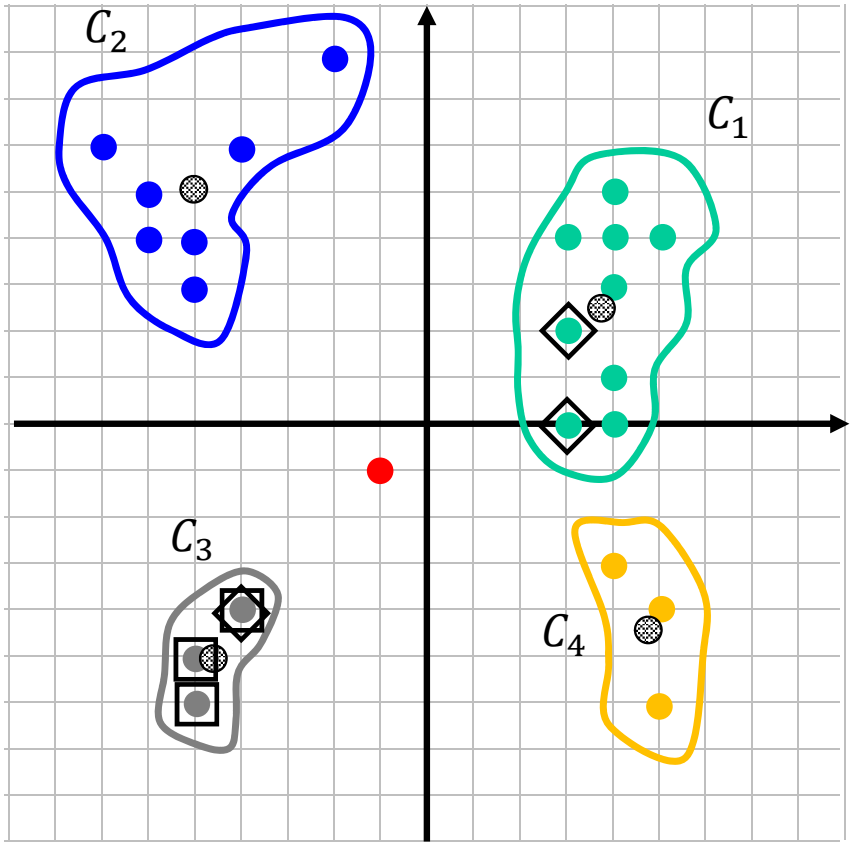
- Build:
  - Divide vectors into clusters k clusters $C_1, \ldots, C_k$ (e.g. using k-means-algorithm)
  - Calculate centroid c_i for each cluster
  - Create "inverted list" for each centroid, which stores references to all vectors belonging to that cluster
- Query:
  - Calculate m nearest cluster(s) to Q
  - Search within these clusters l nearest neighbours

Inverted file (II): Example (I)



Inverted file (III): Example (II)

- Let $Q = (-1, -1)$



Cluster	Centroid c_i	$euDist(c_i, Q)$
C_1	(3.78,2.56)	5.96
C_2	(−5.0,5.14)	7.33
C_3	(−4.67, −5.0)	5.43
C_4	(4.67, −4.33)	6.58

- Let $m = 1, \mathrm{l} = 3$:
  - Nearest cluster: C_3
  - Result: vectors in C_3 □
- Let $m = 2, \mathrm{l} = 3$:
  - Nearest clusters: C_1 and C_3
  - Result: vectors in C_1 and C_3 ◇

- Conclusion:
  - IVF reduces computational complexity
  - Reduction of accuracy
  - Results depend on parameters

LSH (I)

- Transform high dimensional vectors into lower dimensional hash codes
- Basic idea: input vectors close together collide with high probability (different to hashing for searching or crytographic hashing!)

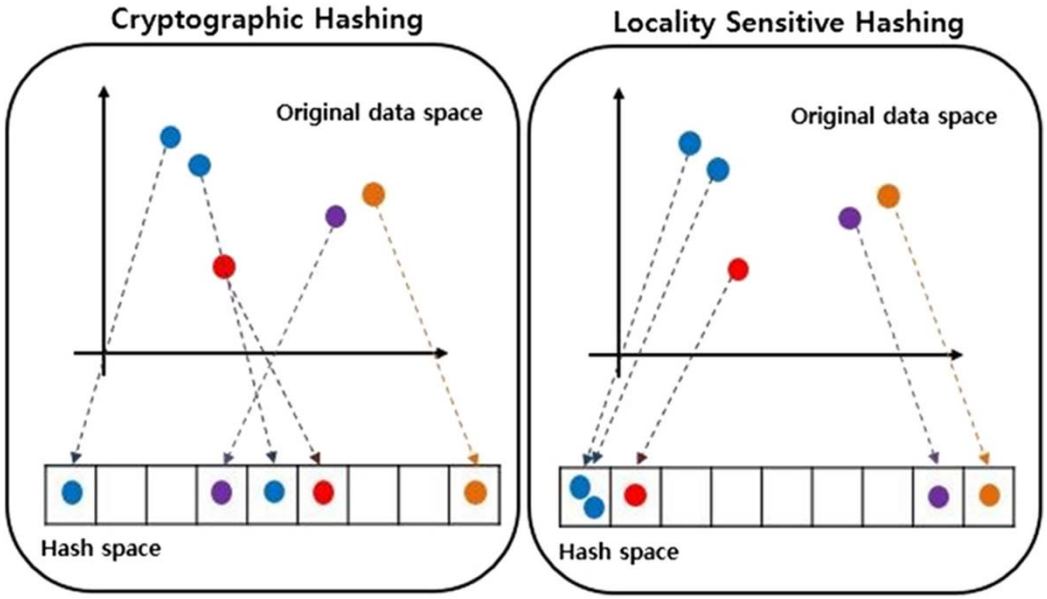


Figure from: Kim, Jihyeon & Park, Jaewoo & Low, Cheng-Yaw & Teoh, Andrew. (2024). Cancellable biometrics based on the index-of-maximum hashing with random sparse binary encoding. Multimedia Tools and Applications. 83. 1-28. DOI: 10.1007/s11042-023-17711-w.

LSH (II)

- Build time:
  - Choose LSH-family $\mathcal{H} = \{h_1, \dots, h_n\}$
  - Build L independent hash tables $HT_1, \dots, HT_L$
    - Each table uses k independently selected hash functions $h_1, \dots, h_k \in \mathcal{H}$
    - Combine them into one $g_i(x) = (h_1(x), \dots, h_k(x))$ with $i \in \{1, \dots, L\}$
  - Store all vectors in all HT_i using g_i
- Query time:
  - Calculate $g_i(Q)$ for all $i \in \{1, \dots, L\}$
  - Collect all vectors from bucket
  - Calculate similarity to Q for all these vectors

Piotr Indyk, Rajeev Motwani: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. STOC 1998: 604-613. DOI: 10.1145/276698.276876

Alexandr Andoni, Piotr Indyk: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM 51(1): 117-122 (2008). DOI: 10.1145/1327452.1327494

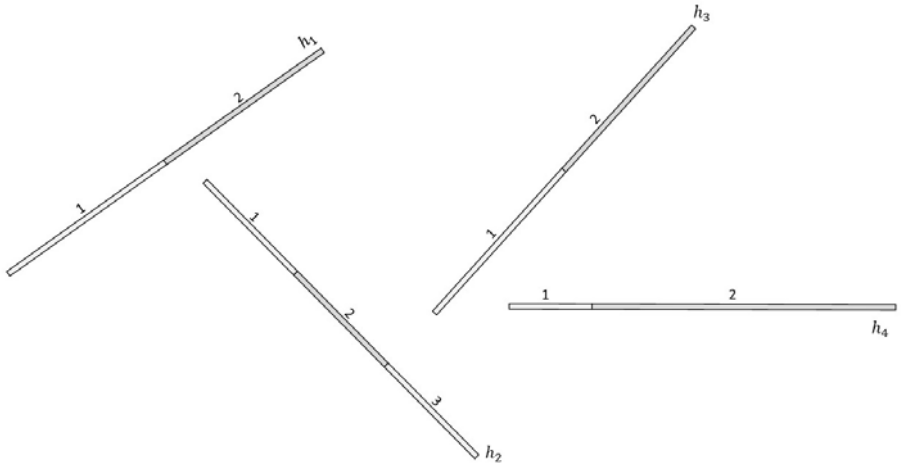
LSH (III): Example (I)

- One approach: E2LSH
  - Use hyperplanes as hash functions

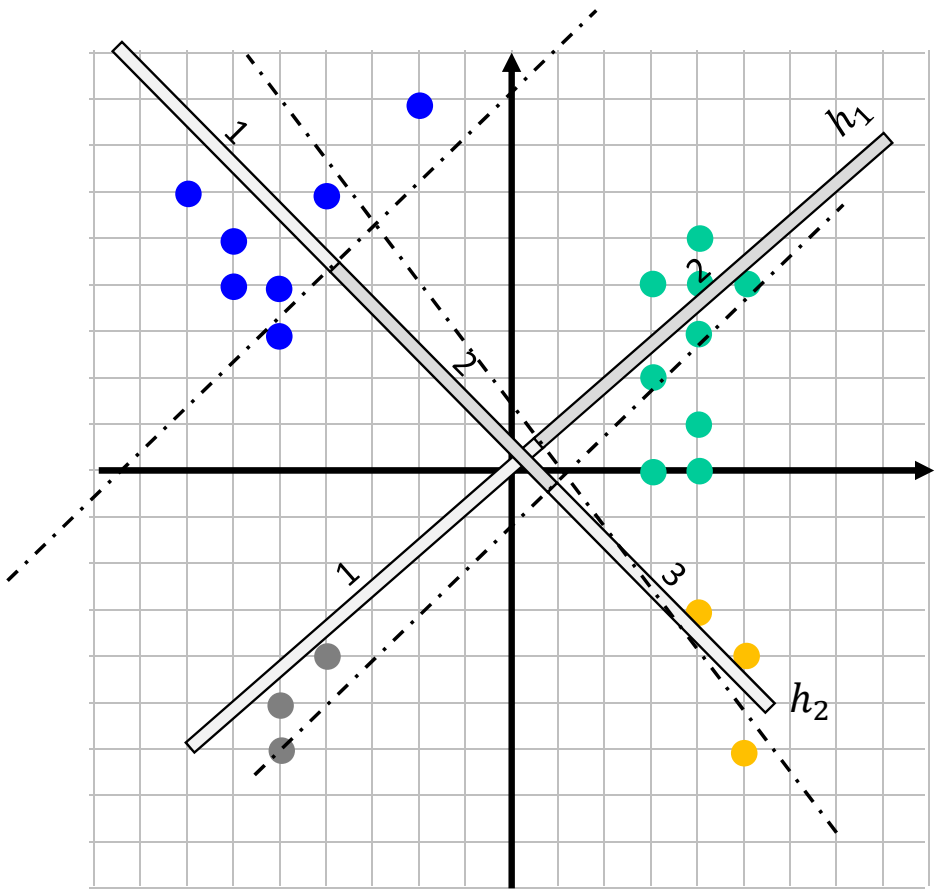
Mayur Datar, Nicole Immorlica, Piotr Indyk, Vahab S. Mirrokni: Locality-sensitive hashing scheme based on p-stable distributions. SCG 2004: 253-262. DOI: 10.1145/997817.997857

Alexandr Andoni and Piotr Indyk. 2005. LSH Algorithm and Implementation (E2LSH). https://www.mit.edu/~andoni/LSH/

- Example:
  - Sample data set
  - Four hyperplanes
  - Three hash tables
  - Search for $Q = (-1, -1)$



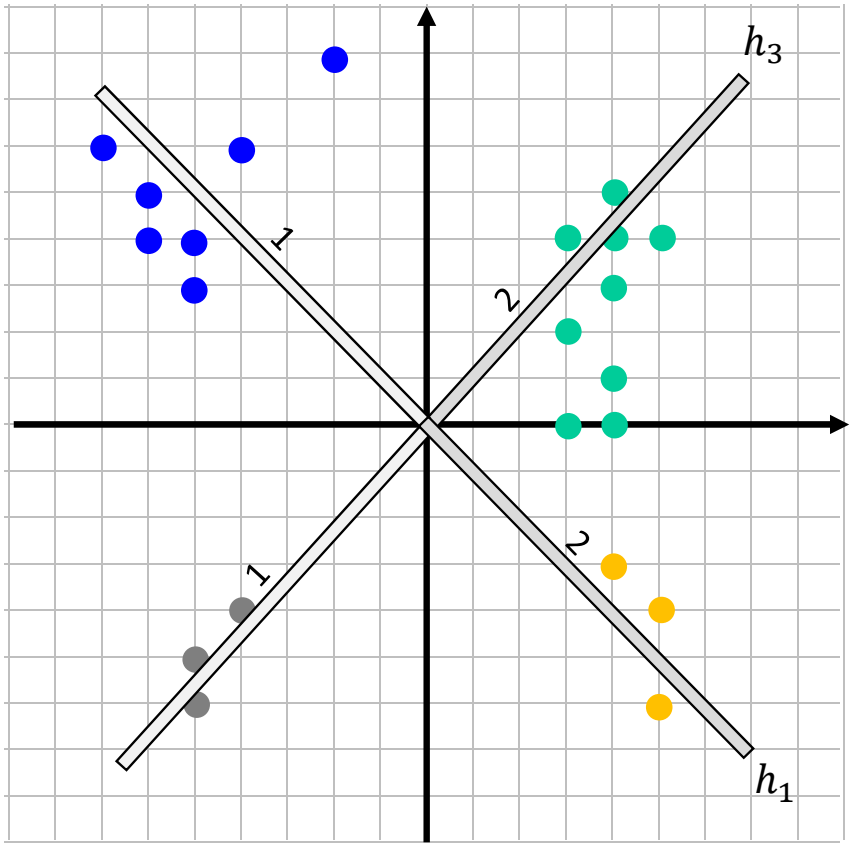
LSH (IV): Example (II)



- Let $g_1 = (h_1, h_2)$
- Hash each vector into HT_1:

Bucket	Vectors
11	$(-7,6)$ $(-4,6)$ $(-6,5)$ $(-6,4)$ $(-5,4)$
12	$(-5,3)$ $(-4,-4)$ $(-5,-5)$
13	$(-5,-6)$ $(5,-6)$
21	$(-2,8)$
22	$(3,2)$ $(4,3)$ $(3,4)$ $(4,4)$ $(5,4)$ $(4,5)$
23	$(3,0)$ $(4,0)$ $(4,1)$ $(4,-3)$ $(5,-4)$

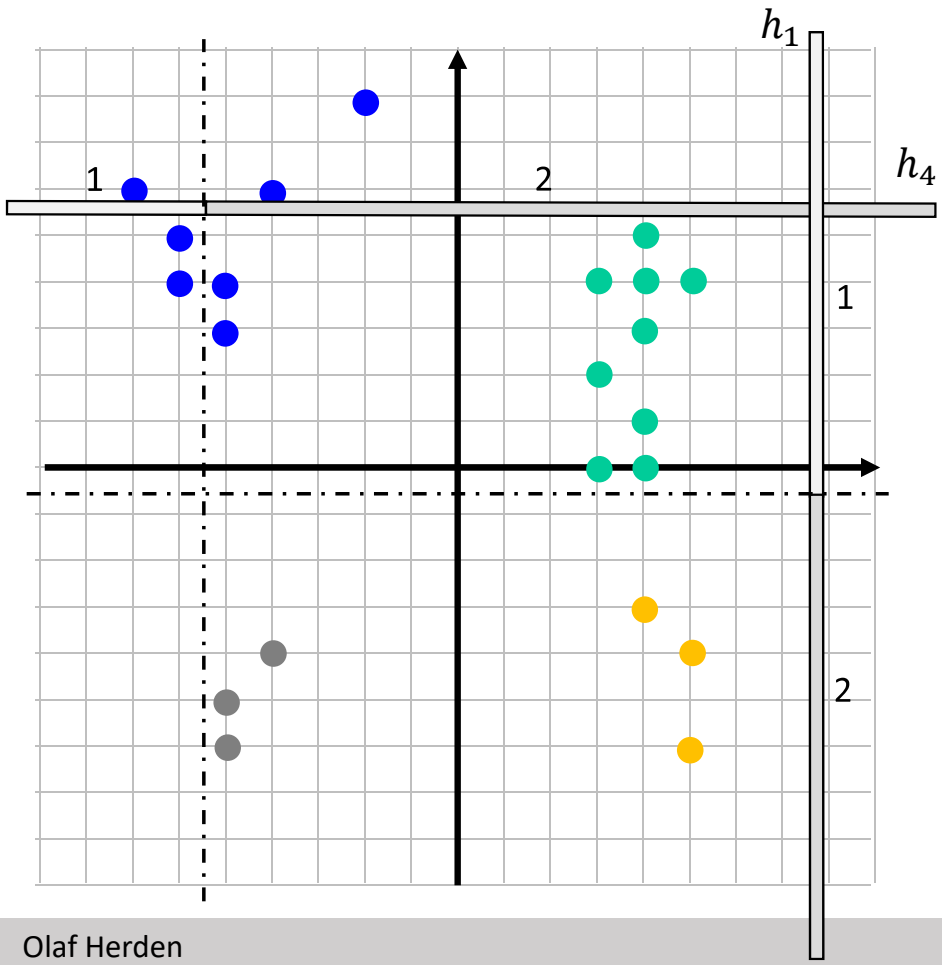
LSH (V): Example (III)



- Let $g_2 = (h_3, h_1)$
- Hash each vector into HT_2:

Bucket	Vectors
11	$(-5,-5)$ $(-4,-4)$ $(-6,4)$ $(-5,3)$ $(-6,5)$ $(-7,6)$ $(-5,4)$
12	$(-5,-6)$ $(5,-6)$
21	$(-2,8)$ $(-4,6)$ $(3,4)$ $(4,5)$
22	$(4,-3)$ $(5,-4)$ $(3,0)$ $(4,0)$ $(4,1)$ $(3,2)$ $(4,3)$ $(4,4)$ $(5,4)$

LSH (VI): Example (IV)



- Let $g_3 = (h_4, h_1)$
- Hash each vector into HT_3:

Bucket	Vectors
11	$(-6,4)$ $(-6,5)$ $(-7,6)$
12	---
21	$(-2,8)$ $(-4,6)$ $(-5,3)$ $(-5,4)$ $(3,0)$ $(4,0)$ $(4,1)$ $(3,2)$ $(4,3)$ $(3,4)$ $(4,4)$ $(5,4)$ $(4,5)$
22	$(-4,-4)$ $(-5,-5)$ $(-5,-6)$ $(4,-3)$ $(5,-4)$ $(5,-6)$

LSH (VII): Example (V)

- Calculate 3 NN of $Q = (-1, -1)$
- $g_1(Q) = 12$

Bucket	Vectors
11	(−7,6) (−4,6) (−6,5) (−6,4) (−5,4)
12	(−5,3) (−4, −4) (−5, −5)
13	(−5, −6) (5, −6)
21	(−2,8)
22	(3,2) (4,3) (3,4) (4,4) (5,4) (4,5)
23	(3,0) (4,0) (4,1) (4, −3) (5, −4)

- $g_2(Q) = 11$

Bucket	Vectors
11	(−5, −5) (−4, −4) (−6,4) (−5,3) (−6,5) (−7,6) (−5,4)
12	(−5, −6) (5, −6)
21	(−2,8) (−4,6) (3,4) (4,5)
22	(4, −3) (5, −4) (3,0) (4,0) (4,1) (3,2) (4,3) (4,4) (5,4)

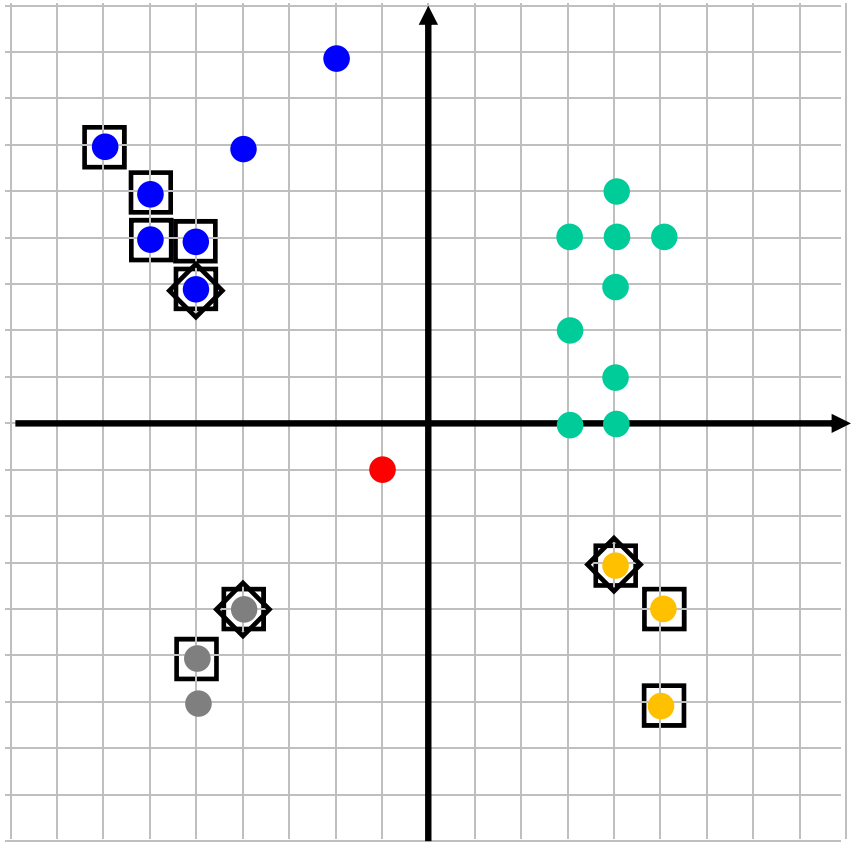
- $g_3(Q) = 22$

Bucket	Vectors
11	(−6,4) (−6,5) (−7,6)
12	---
21	(−2,8) (−4,6) (−5,3) (−5,4) (3,0) (4,0) (4,1) (3,2) (4,3) (3,4) (4,4) (5,4) (4,5)
22	(−4, −4) (−5, −5) (−5, −6) (4, −3) (5, −4) (5, −6)

LSH (VIII): Example (VI)

12	● $(-5,3)$ ● $(-4,-4)$ ● $(-5,-5)$
11	● $(-5,-5)$ ● $(-4,-4)$ ● $(-6,4)$ ● $(-5,3)$ ● $(-6,5)$ ● $(-7,6)$ ● $(-5,4)$
22	● $(-4,-4)$ ● $(-5,-5)$ ● $(-5,-6)$ ● $(4,-3)$ ● $(5,-4)$ ● $(5,-6)$

- Search in buckets
- Result: ● $(-4,-4)$
  ● $(4,-3)$
  ● $(-5,3)$

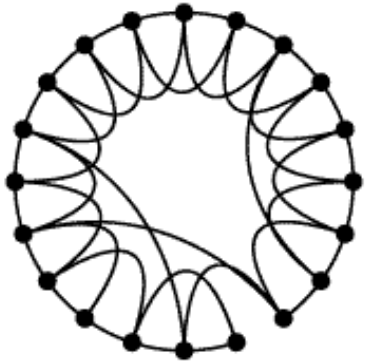


□ : vectors from selected buckets

◇ : result

Navigable Small World (NSW)

- Small World Graph:
  - High cluster coefficient
  - Low distances
  - Characteristic path grows in $O(\log(n))$



Watts, D., Strogatz, S. Collective dynamics of 'small-world' networks. Nature 393, 440–442 (1998). https://doi.org/10.1038/30918

- Navigable graph:
  - Length of search path found by best-first-search algorithm grows in $O(\log(n))$

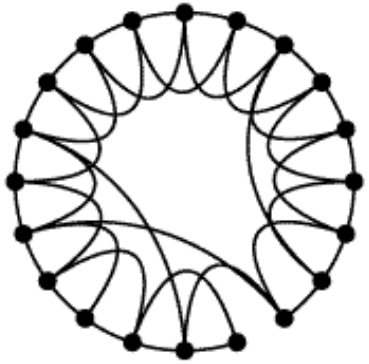
Kleinberg, J. Navigation in a small world. *Nature* 406, 845 (2000). https://doi.org/10.1038/35022643

- NSW:
  - Graph with both properties
  - Search complexity is logarithmic
  - But: out-degree of nodes tends to scale in $\log(n) \Rightarrow O(\log(n)^k)$ with $k > 1$

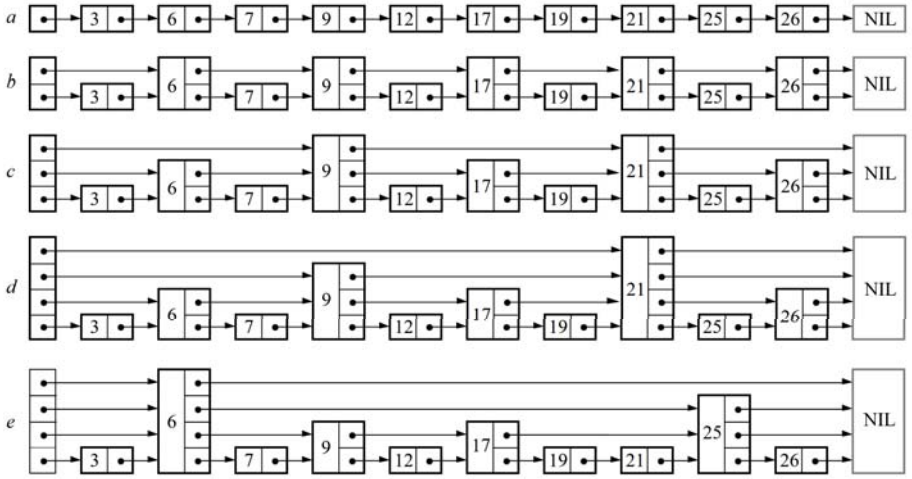
Hierarchical Navigable Small World (HNSW)

William W. Pugh: Skip Lists: A Probabilistic Alternative to Balanced Trees. Commun. ACM 33(6): 668-676 (1990).

- Concept 1: NSW graph

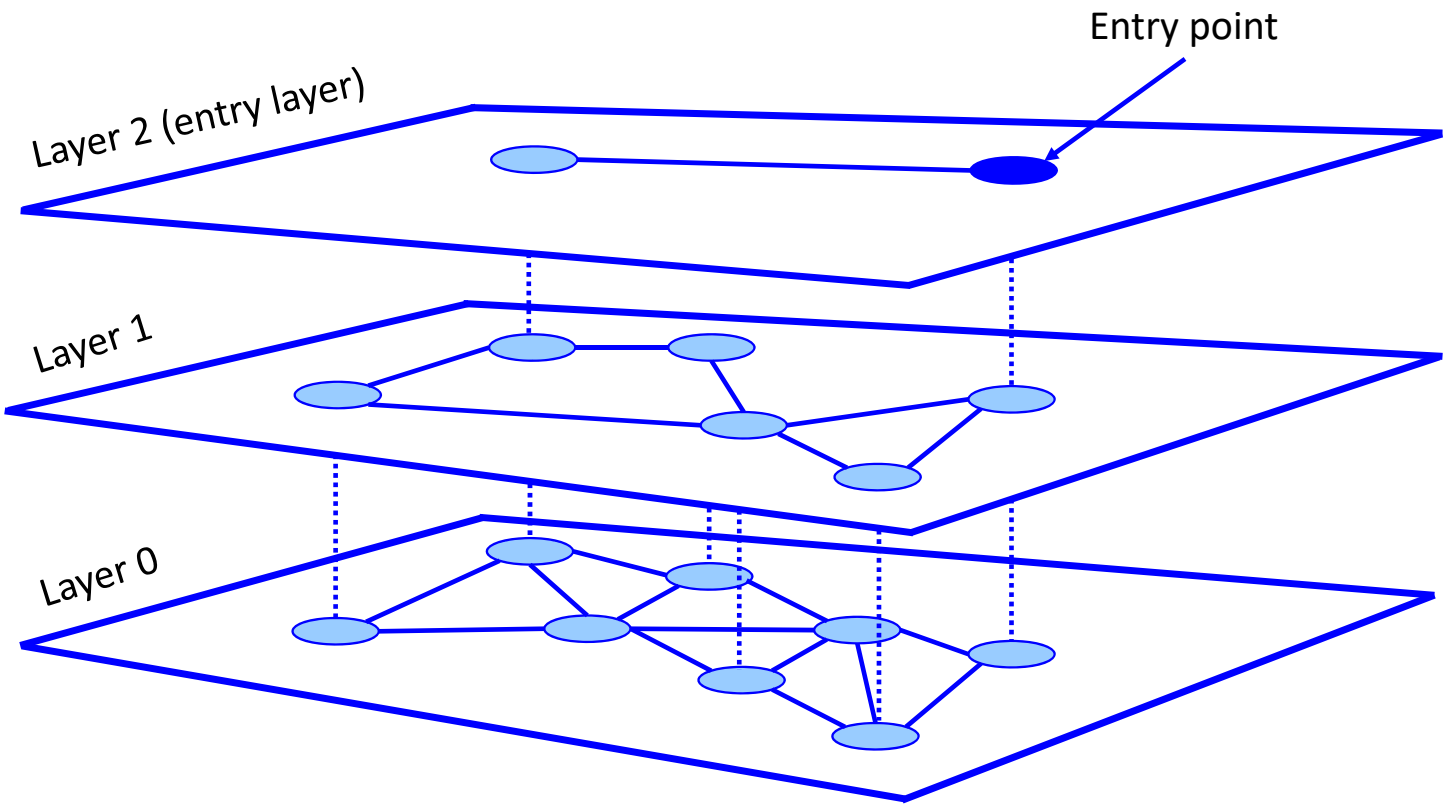


- Concept 2: Skip list



Combining these two ideas
Aim: keep logarithmic complexity

HNSW (II): Structure

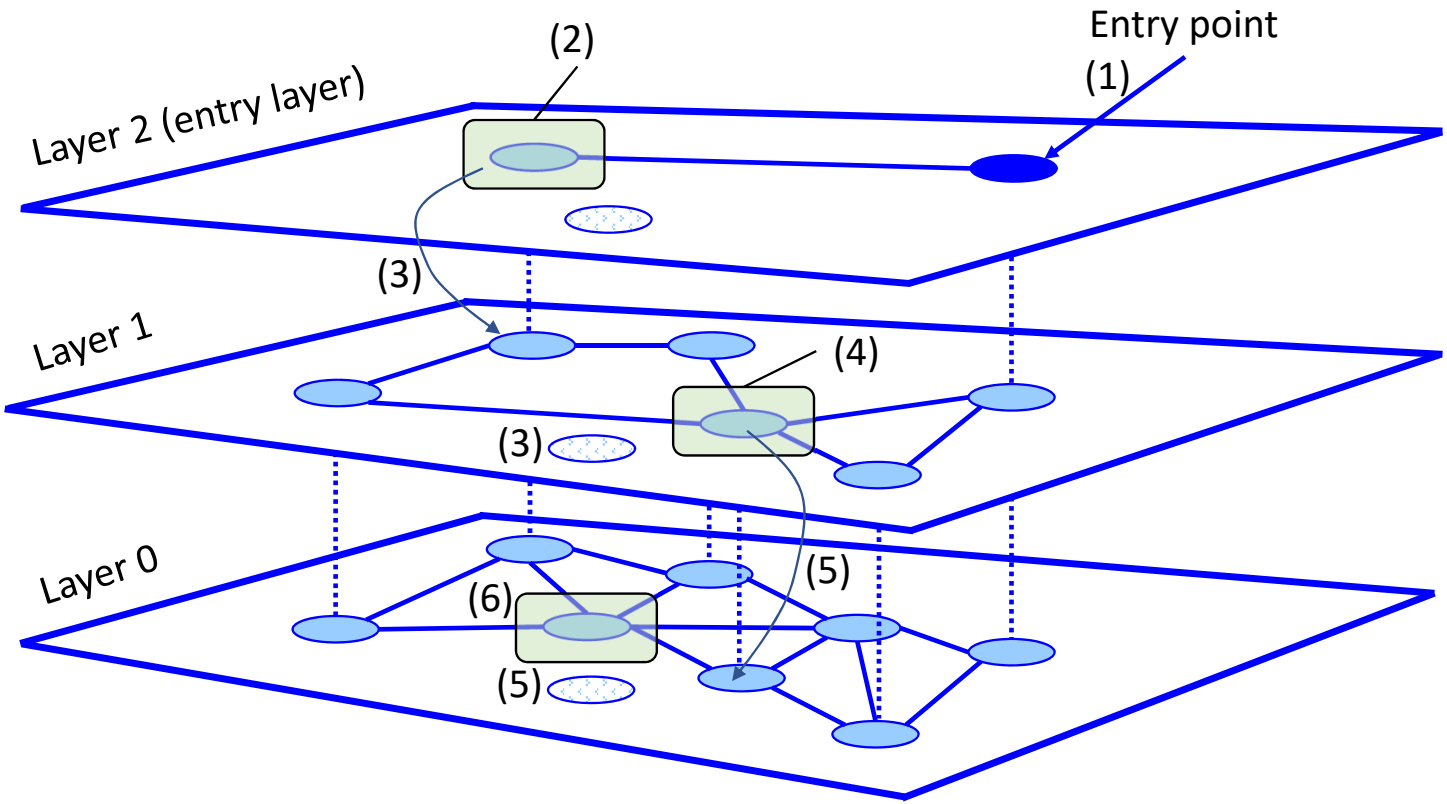


- Structure:
  - M layers
  - Entry/Top layer (M) to bottom layer (0)
  - Subset of nodes from layer N are duplicated to layer $N + 1$ (for $0 \leq N \leq M - 1$)
  - From M to 0:
    - Increasing number of nodes
    - Increasing number of links
    - Decreasing magnitude of links
  - One top layer node is entry point

Yury A. Malkov, Dmitry A. Yashunin: Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42(4): 824-836 (2020)

HNSW (III): Searching

- Given: search vector Q ( )

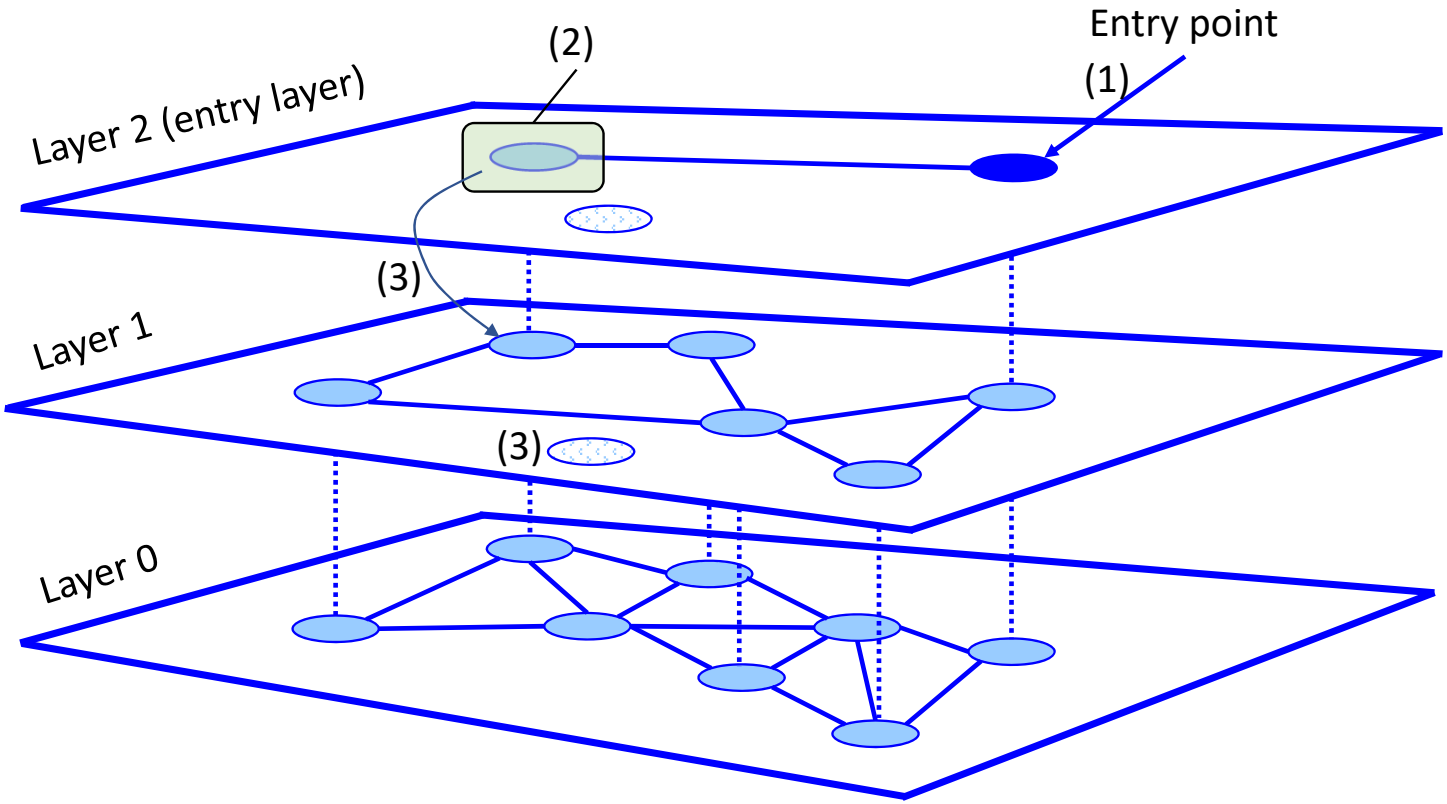


(1) Enter top layer
(2) Search NN of Q (call it X)
(3) Enter layer $M - 1$
(4) Search NN of Q starting with X
- Repeat this procedure until bottom layer is reached
- In the example:

(5) Enter layer 0
(6) Search NN of Q

HNSW (IV): Inserting (I)

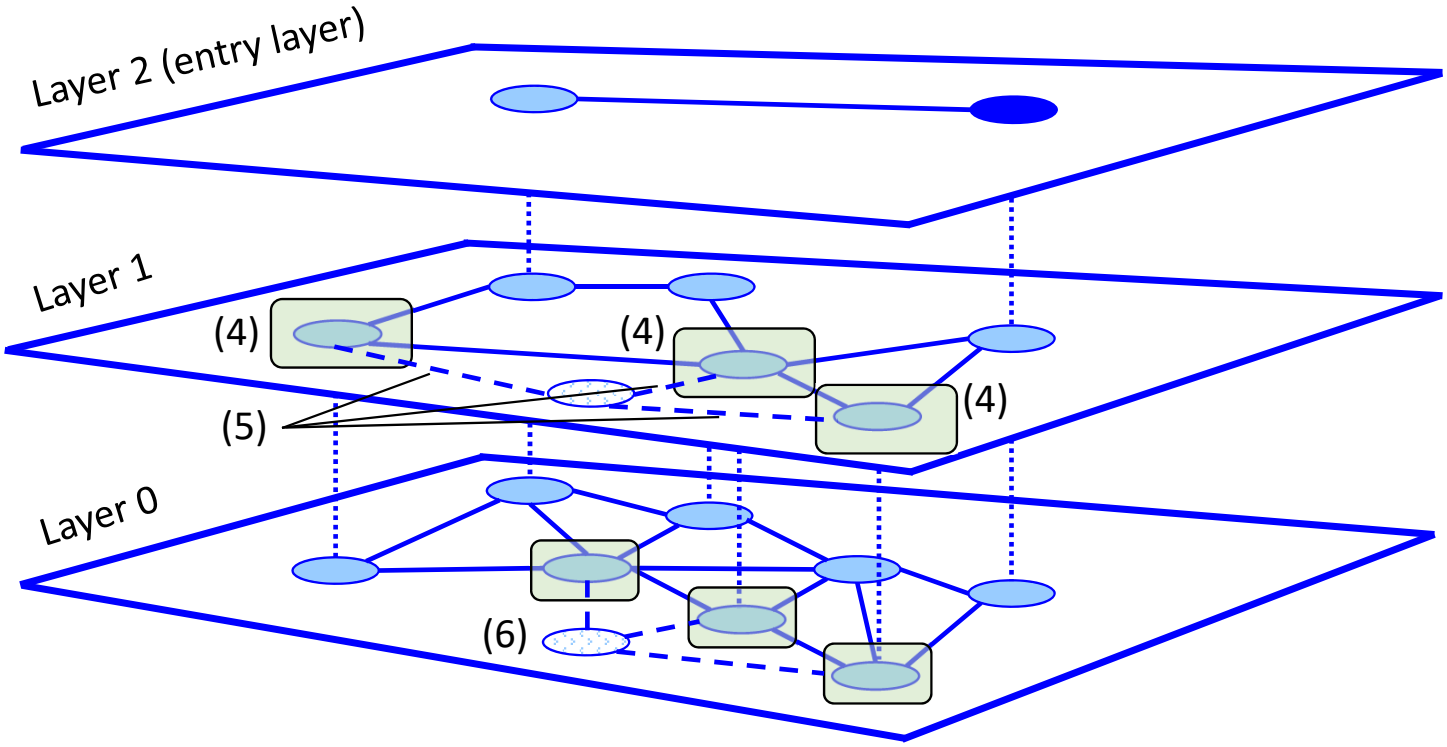
- Given:
  - vector to be inserted ( )
  - parameter k (NN to be considered in insertion layers)



- Random function f_{rand} delivers insertion layer l
- Let $l = 1$
- Phase 1:

  (1) - (3): Start in top layer and traverse to l (analogous to search)

HNSW (V): Inserting (II)

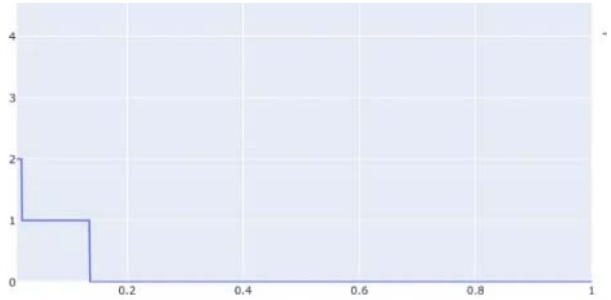


- Phase 2:
  - (4) Calculate k NN of Q in l (=: X)
  - (5) Insert links from Q to each $x \in X$ (- - - -)
  - (6) Repeat (4) and (5) until bottom layer is reached

HNSW (VI): Inserting (III)

- Property of f_{rand}:
  - Use exponentially decreasing probability distribution
  - Use non-zero multiplier m_L
  - $l = [-ln(uniform(0,1)) \cdot m_L]$

$m_L = 0.5$



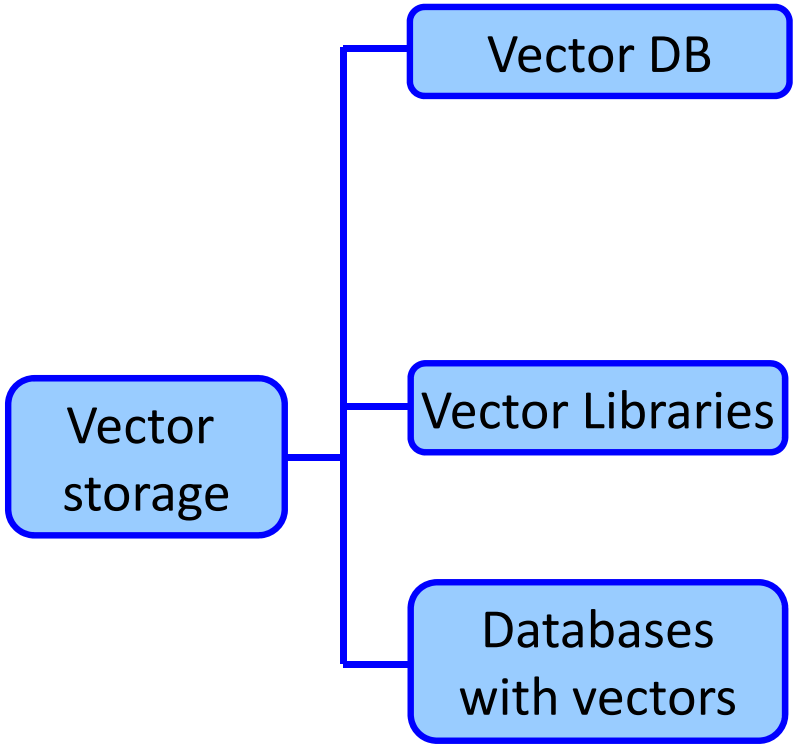
- Algorithm was in simplified representation
- Problem: Number of edges is growing very fast
- Solution: Limit node degree by two parameters:
  - M_{MAX} : maximum number of links a node can have in layers M to 1
  - M_{MAX0}: maximum number of links a node can have in layer 0

[https://medium.com/data-science/similarity-search-part-4-hierarchical-navigable-small-world-hnsw-2aad4fe87d37]

Overview

- Introduction
- Similarity
- Searching
- Indexing
- Vector Database Solutions
- Summary and Outlook

Solutions



- Built only for vectors
- Good performance in vector indexing and searching
- Often not mature wrt database features (e.g. recovery, security, ACID)
- Original data must also be retrieved
- Examples: Pinecone, Weaviate, Milvus

- Can be used directly in application code
- Typically use main memory ⇒ not scalable
- Examples: ANNOY, FAISS, ScaNN

- Extend databases with vector capabilities
- All data in one place: structured, original data and vectors can retrieved together
- Systems are mature w.r.t. database features
- Examples:
  - Relational: Oracle, PostgresQL
  - NoSQL: MongoDB (Document Store), Neo4j (Graph)

Vector databases

- Many overviews in literature and in the web, e.g.

Name	Type	Sub-type	Vector query			Query variant			Vector index			
			Ex	Ap	Rng	Pr	Mul	Bat	Tab	Tr	Gr	Opt
EuclidesDB (2018) [12]	Nat	Vec	✓	✓	✗	✗	✗	✗	✓	✓	✓	✗
Vearch (2018) [15, 77]	Nat	Vec	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗
Pinecone (2019) [2]	Nat	Vec	✗	✓	✗	✓	✓	✗	Proprietary			U
Vald (2020) [8]	Nat	Vec	✓	✓	✓	✗	✗	✓	✗	✗	✓	✗
Chroma (2022) [10]	Nat	Vec	✗	✓	✗	✓	✗	✗	✗	✗	✓	U
Weaviate (2019) [1]	Nat	Mix	✗	✓	✓	✓	✓	✗	✗	✗	✓	✗
Milvus (2021) [16, 122]	Nat	Mix	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
NucliaDB (2021) [18]	Nat	Mix	✗	✓	✓	✓	U	✗	✗	✗	✓	✗
Qdrant (2021) [9]	Nat	Mix	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓
Manu (2022) [61]	Nat	Mix	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
Marqo (2022) [19]	Nat	Mix	✗	✓	✗	✓	✓	✓	✗	✗	✓	U
Vespa (2020) [17]	Ext	NoSQL	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
Cosmos DB (2023)	Ext	NoSQL	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗
MongoDB (2023)	Ext	NoSQL	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗
Neo4j (2023)	Ext	NoSQL	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗
Redis (2023)	Ext	NoSQL	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗
AnalyticDB-V (2020) [130]	Ext	Rel	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
PASE+PG (2020) [136]	Ext	Rel	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
pgvector+PG (2021) [7]	Ext	Rel	✓	✓	✓	✓	✓	✗	✓	✗	✓	✓
SingleStoreDB (2022) [11, 99]	Ext	Rel	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓
ClickHouse (2023) [20]	Ext	Rel	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓
MyScale (2023) [21]	Ext	Rel	✓	✓	✓	✓	✗	✗	✓	✗	P	✓

Dates are estimated from paper publication, earliest Github release, blog post, or other indications. Dates indicate year when vector search capability first appeared in the product. Abbreviations: *Ex.* exact k-NN, *Ap.* ANN, *Rng.* range, *Pr.* predicated, *Mul.* multi-vector, *Bat.* batched, *Tab.* table, *Tr.* tree, *Gr.* graph, *Opt.* query optimizer, *Nat.* native, *Ext.* extended, *Vec.* mostly vector, *Rel.* relational, *U* unknown, *P* proprietary

James Jie Pan, Jianguo Wang, Guoliang Li: Survey of vector database management systems. VLDB J. 33(5): 1591-1615 (2024)

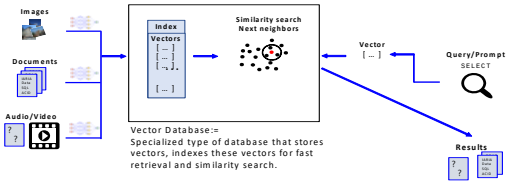
Overview

- Introduction
- Similarity
- Searching
- Indexing
- Vector Database Solutions
- Summary and Outlook

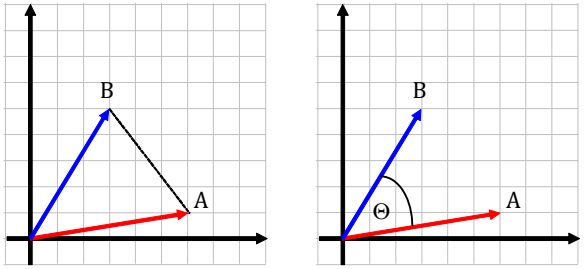
Summary

Use cases

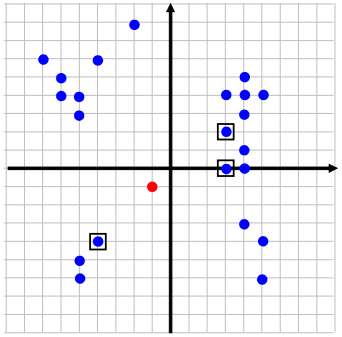
Architecture



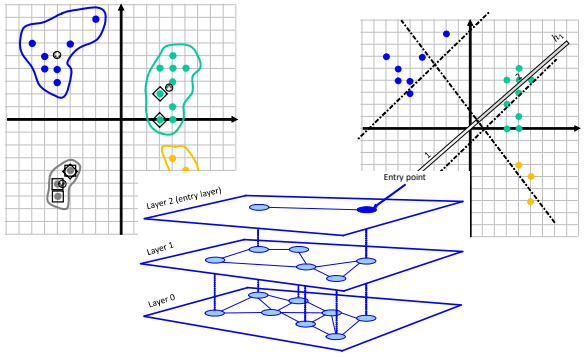
Similarity



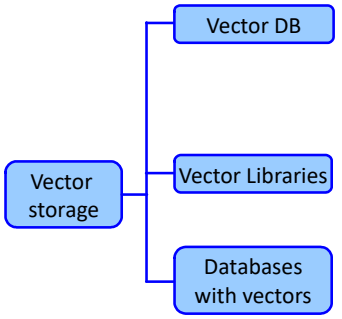
Searching



Indexing



Solutions



Topics not covered

- More search/index technologies:
  - Vector quantization
  - Metadata storage
  - Pre- and post-processing (filtering), refinement/re-ranking
- Query execution and optimization
- Architecture, e.g. (horizontal) scaleability
- (Real time) updates
- Performance tests

Outlook (I)

- New similarity metrics:
  - E.g. sensitive Euclidean
  - Combining metrics
- Index design:
  - Scaleability with ever growing data volumes
  - Concurrency
- System design:
  - Query prediction and caching
  - Distributed solutions, e.g. partitioning strategies
  - Security and data privacy

Outlook (II)

- New applications:
  - Incremental k-NN-search
  - Multi-vector search
- Performance tests

Thank you for for your attention!

Questions?