Acknowledgments

The research project is implemented in the framework of H.F.R.I call “3rd Call for H.F.R.I.” Research Projects to Support Faculty Members & Researchers” (H.F.R.I. Project Number: 25146).

Graph-based Approximate Nearest Neighbor Search

What I learned from

"A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search"[1]

(and other sources)

Enriched with examples

Components

1. Preliminaries
2. Basic graphs
3. Algorithms
4. Metrics
5. Competitors
6. Datasets
7. References

Definitions

- **Nearest Neighbors Search**: Given a finite dataset $\mathcal{S}$ in Euclidean space $\mathbb{R}^d$ and a query q, NNS obtains k nearest neighbors $\mathcal{R}$ of q by evaluating $\delta(x, q)$, where $x \in \mathcal{R}$. $\mathcal{R}$ is described as follows:

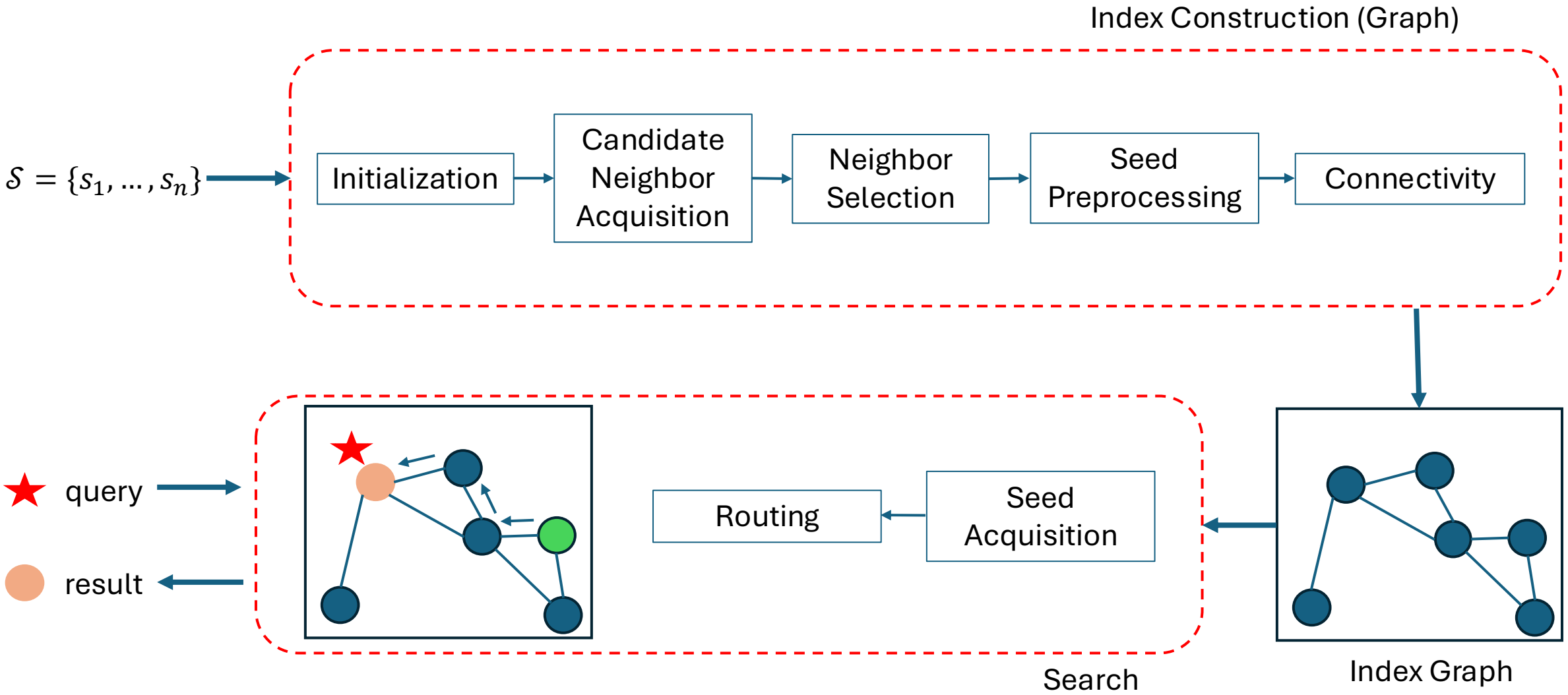
$$\mathcal{R} = argmin_{\mathcal{R} \subseteq \mathcal{S}, |\mathcal{R}|=k} \sum_{x \in \mathcal{R}} \delta(x, q)$$

- **Approximate Nearest Neighbor Search**: Given a finite dataset $\mathcal{S}$ in Euclidean space $\mathbb{R}^d$, and a query q, ANNS builds an index $\mathcal{I}$on $\mathcal{S}$. It then gets a subset $\mathcal{C}$ of $\mathcal{S}$ by $\mathcal{I}$, and evaluates $\delta(x, q)$ to obtain the approximate k nearest neighbors $\tilde{\mathcal{R}}$ of q, where $x \in \mathcal{C}$.

Graph-based ANNS

Given a finite dataset $\mathcal{S}$ in Euclidean space $\mathbb{R}^d$, $G(V, E)$denotes a graph (the index $\mathcal{I}$) constructed on $\mathcal{S}$, $v \in V$that uniquely corresponds to a point $x \in \mathcal{S}$. Here $\forall(v, u) \in E$ represents the neighbor relationship between v and u, and $v, u \in V$. Given a query q, seeds $\hat{\mathcal{S}}$, routing strategy, and termination condition, the graph-based ANNS initializes approximate k nearest neighbors $\tilde{\mathcal{R}}$ of q with $\mathcal{S}$, then conducts a search from $\hat{\mathcal{S}}$ and updates $\tilde{\mathcal{R}}$ via a routing strategy. Finally, it returns the query result $\tilde{\mathcal{R}}$ once the termination condition is met.

The workflow



The workflow – Index Construction

1. Initialization: The methods used to select a subset of $\mathcal{S}$ to use as graph's vertices.
2. Candidate neighbor acquisition: Given a finite dataset $\mathcal{S}$, a point $\mathrm{p} \in \mathcal{S}$, the candidate neighbor acquisition gets a subset $\mathcal{C} \subset \mathcal{S} \setminus \{p\}$ as p's candidate neighbors $N(p) \subset \mathcal{C}$.
3. Neighbor selection: Given a point p and its candidate neighbors $\mathcal{C}$, the neighbor selection obtains a subset of $\mathcal{C}$ to update $N(p)$.
4. Seed preprocessing: The methods used to initialize a set $\hat{\mathcal{S}}$ of initial points, from which the search strategy will start from.
5. Connectivity: The methods used to ensure graph's connectivity.

The workflow - Search

1. Seed acquisition: The strategies used to obtain a set of points $\hat{S}$, from which the routing starts.
2. Routing: Given a graph $G(V, E)$, a query q, seed set $\hat{S}$, the routing starts from the vertices in $\hat{S}$ and then converges to q by neighbor propagation along the neighbor n of the visited point with smaller $\delta(n, q)$, until the vertex r so that $\delta(r, q)$ reaches a minimum.

Disclaimer

- Not all authors of the following papers, discuss in detail every component identified earlier.

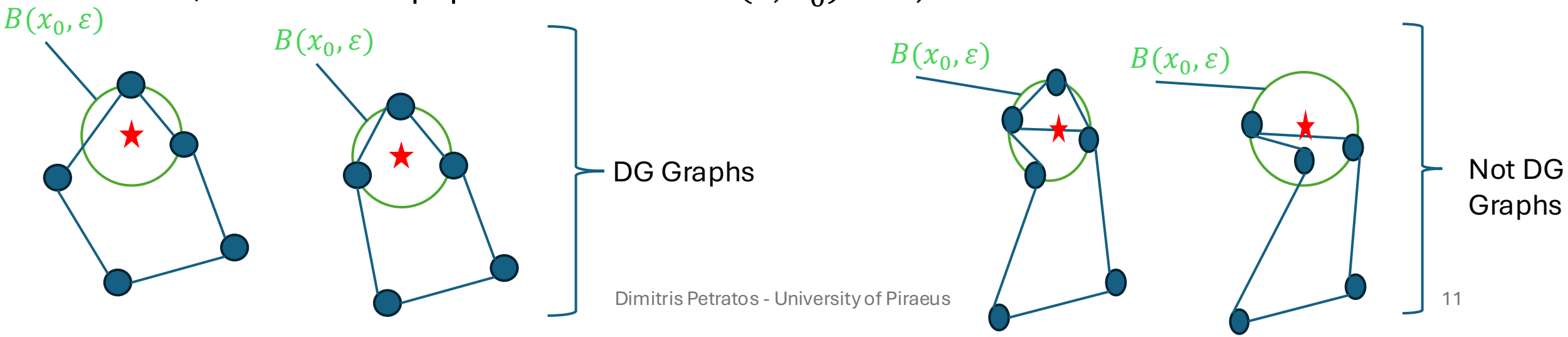
Key Components – Base Graphs

- Base graphs for ANNS:
  - Delaunay Graph (DG)
  - Relative Neighborhood Graph (RNG)
  - K-Nearest Neighbor Graph (KNNG)
  - Minimum Spanning Tree (MST)
  - +Other variants described in the papers

Delaunay Graph (DG)

Properties:

- For every $e = (x, y) \in E, \exists B(x_o, \varepsilon)$ where $x_0 \in \mathbb{R}^d, \varepsilon > 0$ such that:
  - $\delta(x, x_0) = \varepsilon$
  - $\delta(y, x_0) = \varepsilon$
  - $\nexists z: \delta(z, x_0) < \varepsilon$
  - $\nexists A \subset \mathcal{S}$ with $|A| \geq 2$ such that $\delta(z, x_0) = \varepsilon, \forall z \in A$.



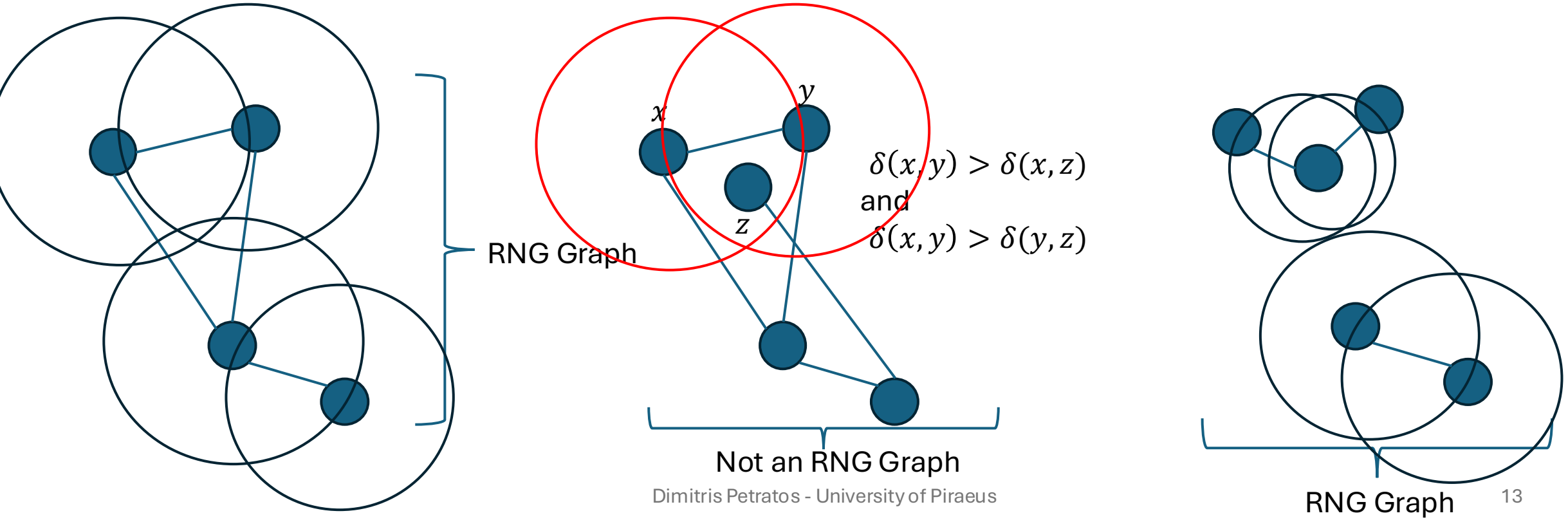
Delaunay Graph (DG)

- ANNS with Delaunay Graph (DG) always returns precise results.
- When d is extremely high, DG is almost fully connected, leading to a large search space.
- DGs are undirected.
- DGs ensure global connectivity.

Relative Neighborhood Graph (RNG)

Properties

- $\forall (x, y) \in E, \forall z \in V: \delta(x, y) < \delta(x, z) \wedge \delta(x, y) < \delta(y, z)$



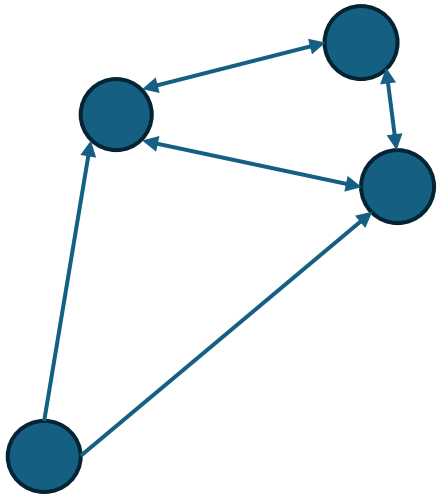
Relative Neighborhood Graph (RNG)

- Constructing cost: $O(|\mathcal{S}|^3)$
- RNGs are undirected.
- RNGs do not ensure global connectivity.

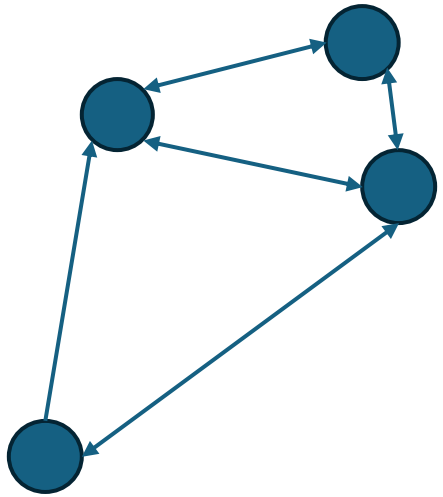
K-Nearest Neighbor Graph (KNNG)

Properties:

- Each $v \in V$ is connected to its k nearest neighbors



KNNG(k=2) Graph



Not a KNNG(k=2) Graph

K-Nearest Neighbor Graph (KNNG)

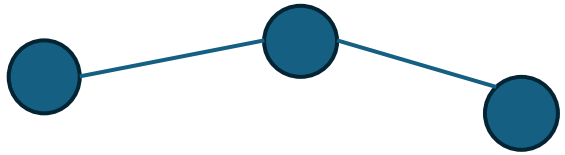
- Construction cost: $O(|\mathcal{S}|^2)$
- KNNGs are directed.
- KNNGs do not ensure global connectivity.

Minimum Spanning Tree (MST)

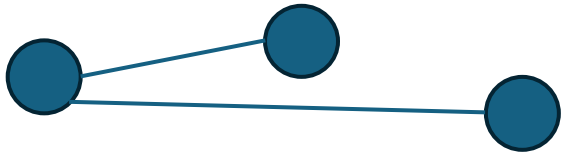
Properties:

- MST graph is the graph that satisfies

$$E^* = argmin_E \sum_{(x,y)\in E} \delta(x,y)$$



MST



Not an MST

Minimum Spanning Tree (MST)

- MSTs uses the least number of edges to ensure global connectivity.

Comment on graph construction

- Due to high construction cost of exact graph types (DG, RNG, KNNG) through brute force, authors propose their own way to get approximations of graphs.

Greedy Search Algorithm (the basic search algorithm)

- $GreedySearch(s\text{: } start\ node, q\text{: } query\ point, k\text{: } result\ size, l \geq k\text{: } search\ list\ size)$

1. $L \leftarrow \{s\}, V \leftarrow \emptyset$
2. $while\ L \setminus V \neq \emptyset$:
3. $\quad p^* \leftarrow argmin_{p \in L \setminus V} \delta(p, q)$
4. $\quad L \leftarrow L \cup N_{out}(p^*)$ # out-edges incident on p^*
5. $\quad V \leftarrow V \cup \{p^*\}$
6. $\quad if\ |L| > l$:
7. $\quad\quad update\ L\ to\ retain\ closest\ l\ points\ to\ q$
8. $return\ [closest\ k\ points\ from\ L; V]$

SNG (Sparse Neighborhood Graphs)

- An SNG is a graph $G = (V, E)$ is a graph with the following property:

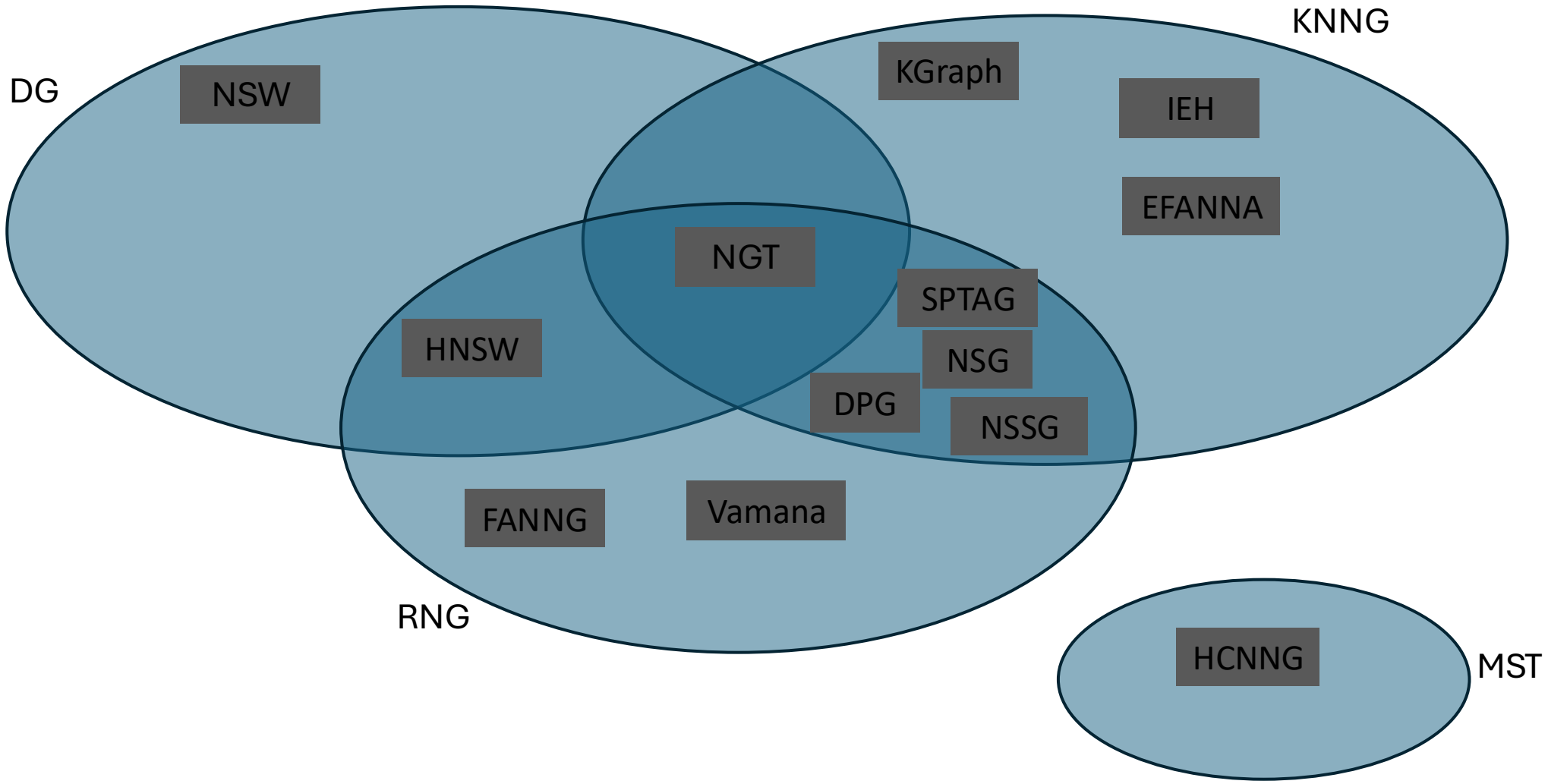
For every query point $q \in V$, the $GreedySearch(v, q, 1,1)$ algorithm converges to q.

Key Components – Graph-based ANNS Algorithms

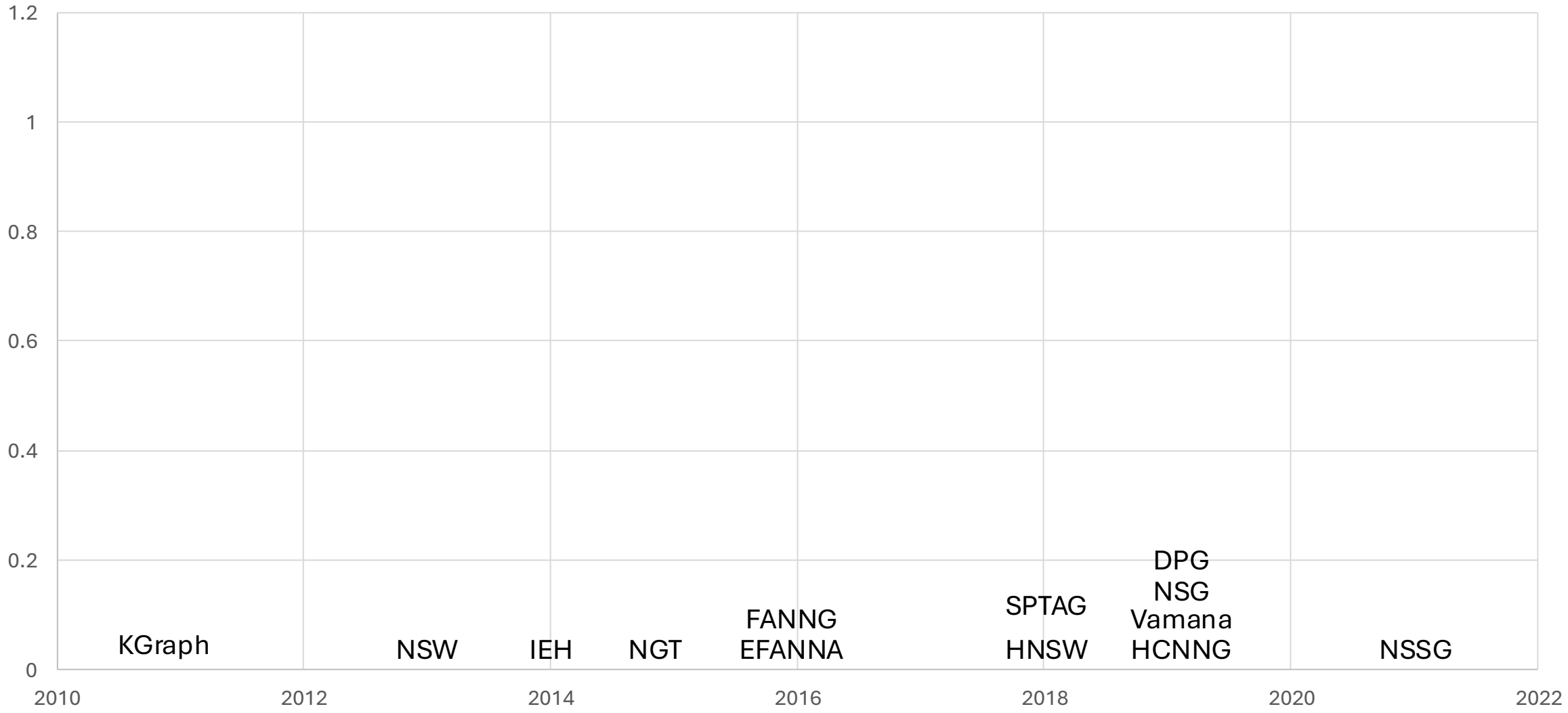
- Navigable Small World graph (NSW)
- Hierarchical Navigable Small World graphs (HNSW)
- Fast Approximate Nearest Neighbor Graph (FANNG)
- Neighborhood Graph and Tree (NGT)
- Space Partition Tree and Graph (SPTAG)
- KGraph
- EFANNA
- IEH
- Diversified Proximity Graph (DPG)
- Navigating Spreading-out Graph (NSG)
- Navigating Satellite System Graph (NSSG)
- Vamana
- HCNNG

**Taxonomy based on [1]

Taxonomy of algorithms

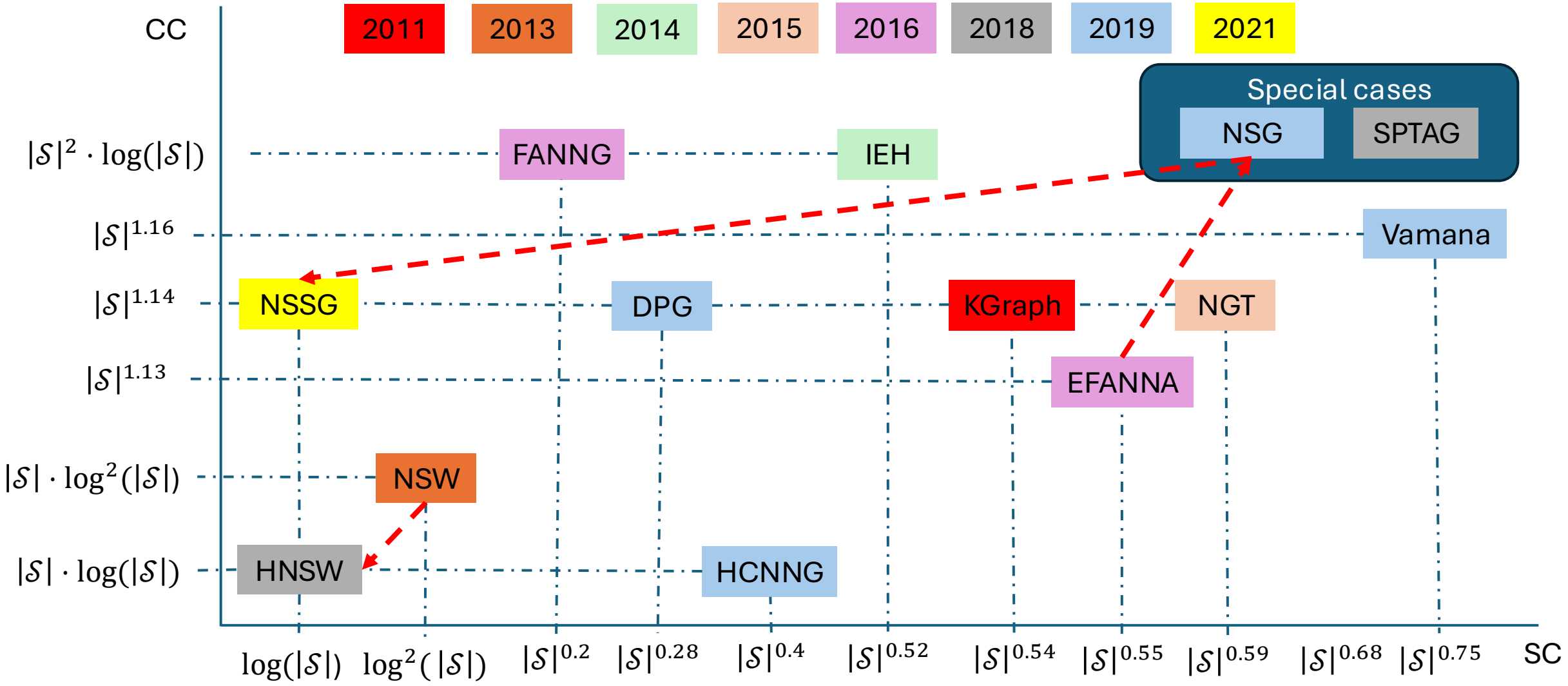


Year published



**Complexities were obtained from [1]

Construction Complexity vs Search Complexity



They share same authors

Special cases

Algorithm	Construction	Search
NSG (2019)	$\lvert\mathcal{S}\rvert^{\frac{1+c}{c}} \cdot \log(\lvert\mathcal{S}\rvert) + \lvert\mathcal{S}\rvert^{1.14}$	$\log(\lvert\mathcal{S}\rvert)$
SPTAG (2018)	$\lvert\mathcal{S}\rvert \cdot \log(\lvert\mathcal{S}\rvert^{c} + t^{t})$	$\lvert\mathcal{S}\rvert^{0.68}$

Navigable Small World graph (NSW) - 2013

- Original paper: https://publications.hse.ru/pubs/share/folder/x5p6h7thif/128296059.pdf [2]
- Graph: Approximate DG, by continuous insertions of elements to ensure global connectivity, presented in the next slides.
- Search Algorithm: The Greedy_Search algorithm presented in the next slides.

Navigable Small World graph (NSW) - 2013

- Search Algorithm (the classic one):
  - Greedy_Search(q: query point, v_{entry}: entry point)

1. $v_{curr} \leftarrow v_{entry}$
2. $\delta_{min} \leftarrow \delta(q, v_{curr})$
3. $for\ each\ v_{friend} \in N(v_{curr})$: # $N(v_{curr})$ is the neighborhood of v_{curr}
4. $\quad \delta_{friend} \leftarrow \delta(q, v_{friend})$
5. $\quad if\ \delta_{friend} < \delta_{min}$:
6. $\quad\quad \delta_{min} \leftarrow \delta_{friend}$
7. $\quad\quad v_{next} \leftarrow v_{friend}$
8. $if\ termination_condition$: # if v_{next} has no neighbors closer to q than v_{next}
9. $\quad return\ v_{curr}$
10. $else$:
11. $\quad return$ Greedy_search(q, v_{curr})

Navigable Small World graph (NSW) - 2013

- Search Algorithm (used by Malkov et al. to create DG):
  - Modified_KNN_Search(q: query point, m: search iterations, k: number of k-nn)
  - $initialize: tempRes = \{\}, candidates = \{\}, visitedSet = \{\}$

1. $for\ i\ in\ range(m)$:
2. $\quad v_{entry} \leftarrow put\ a\ random\ entry\ point\ in\ candidates$
3. $\quad tempRes \leftarrow Null$
4. $\quad while$:
5. $\quad\quad get\ element\ c\ closest\ from\ candidates\ to\ q$
6. $\quad\quad remove\ c\ from\ candidates$
7. $\quad\quad if\ c\ is\ further\ than\ k^{th}\ element\ from\ results$: # check stopping condition
8. $\quad\quad\quad break;$
9. $\quad\quad for\ every\ element\ e\ from\ N(c)$: # update list of candidates
10. $\quad\quad\quad if\ e\ is\ not\ in\ visitedSet$:
11. $\quad\quad\quad\quad add\ e\ to\ visitedSet, candidates, tempRes$
12. $\quad\quad\quad break;$
13. $\quad add\ objects\ from\ tempRes\ to\ results$
14. $return\ best\ k\ elements\ from\ results$

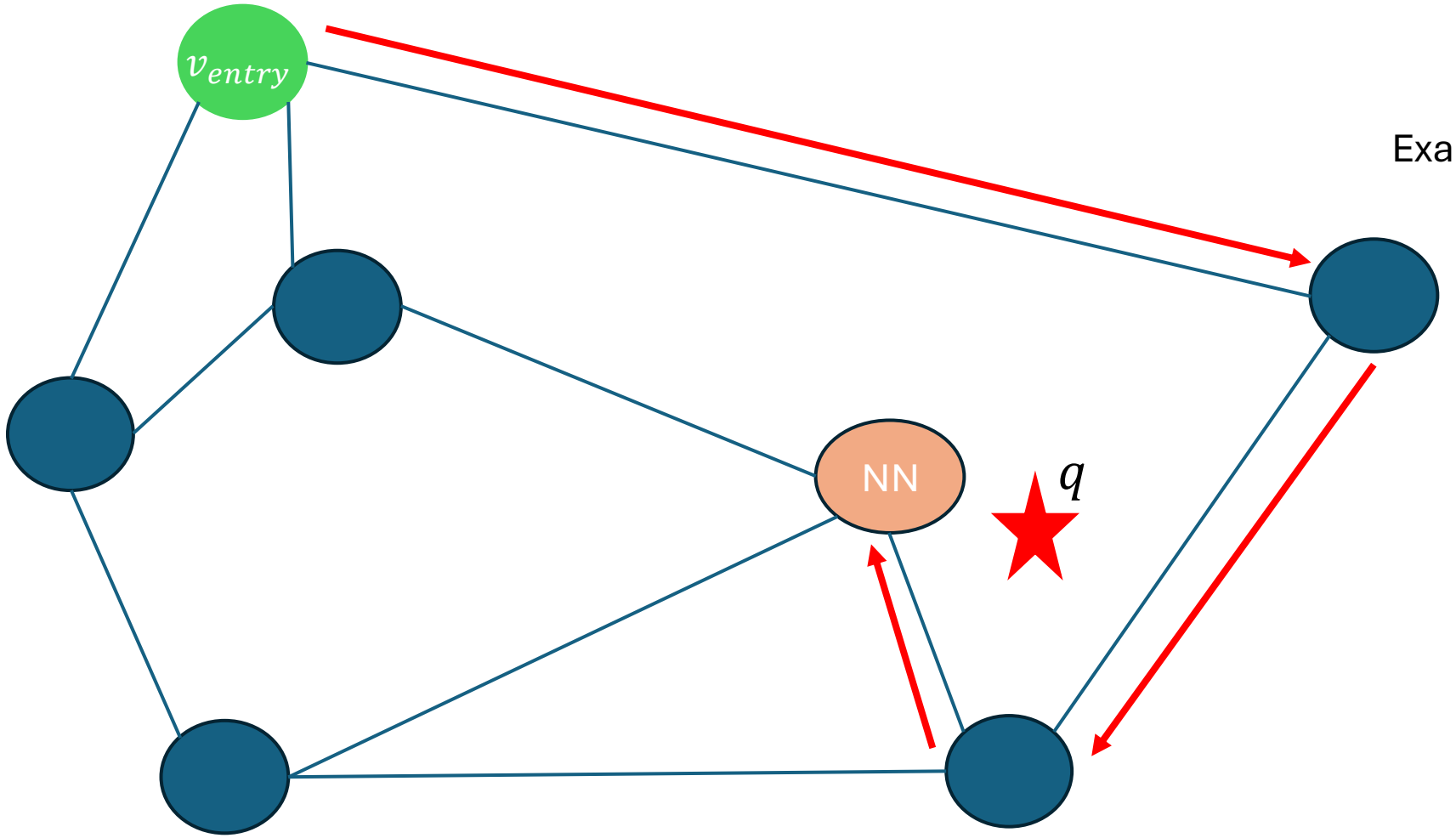
Navigable Small World graph (NSW) - 2013

Graph Construction:

Authors create and use a DG graph, based on the following algorithm

- Nearest_Neighbor_Insert(v: new vertex, f: number of closest vertices, w: number of iterations)
  - $initialize: neighbors \leftarrow$ Modified_KNN_Search(v, w, f)
  1. $for\ i\ in\ range(f)$:
  2. $\quad neighbors[i].connect(v)$
  3. $\quad v.connect(neighbors[i])$

Navigable Small World graph (NSW) -Example



Example taken from [3]

Hierarchical Navigable Small World graphs (HNSW) -2018

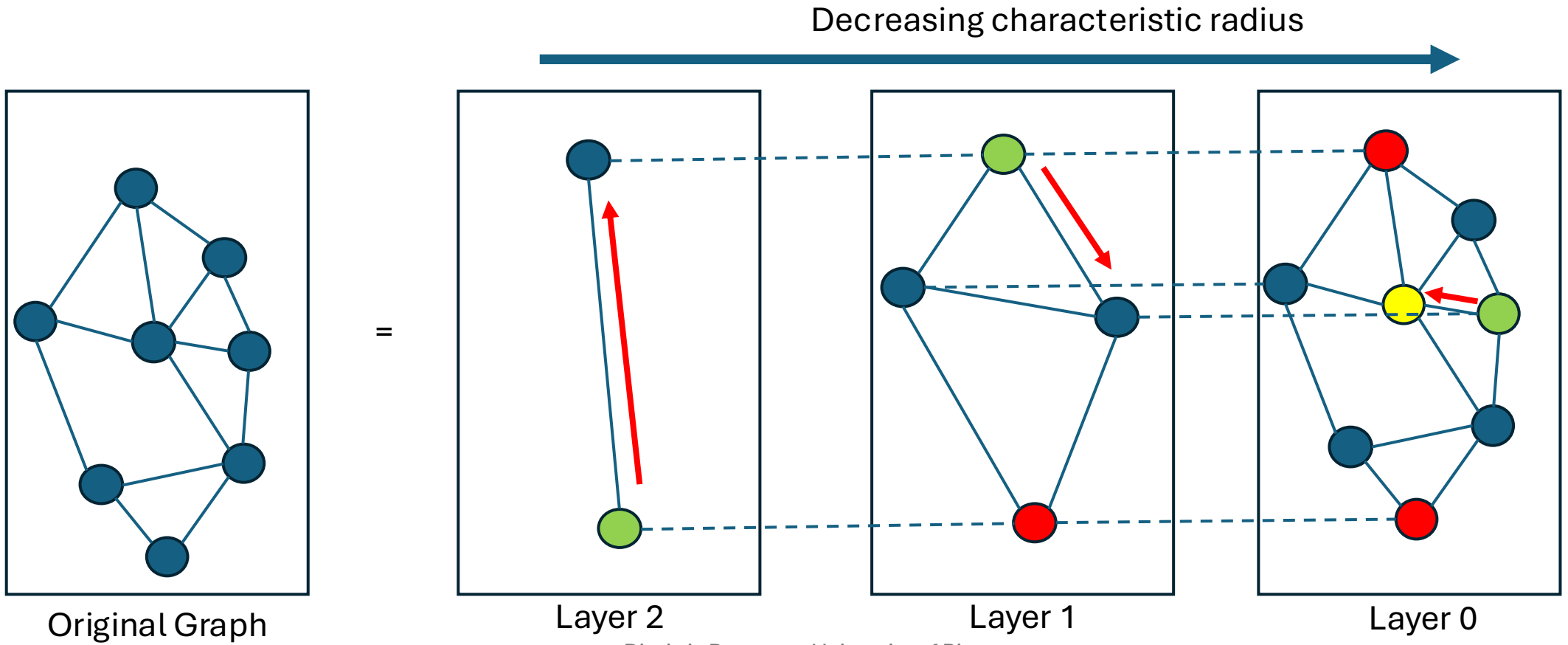
- Original paper: https://arxiv.org/pdf/1603.09320 [4]
- It expands NSW, by adding hierarchy to NSW.
- Nice illustrative examples of NSW and HNSW can be found in [3]: https://www.pinecone.io/learn/series/faiss/hnsw/

HNSW – Revisiting the routing process

- Greedy algorithm's (GA) phases:
  - Zoom-out: Start searching from low-degree vertices.
    - High probability of stacking in a false local minimum.
  - Zoom-in: Start searching from higher-degree (maximum is preferred) vertices.
    - Better search performance at low-dimensional data.
    - Polylogarithmic complexity.

HNSW – The basic idea (from “Zoom-in” to “Zoom-out”)

- Separate the links according to their length scale, into different layers and then search in a multi-layer graph.



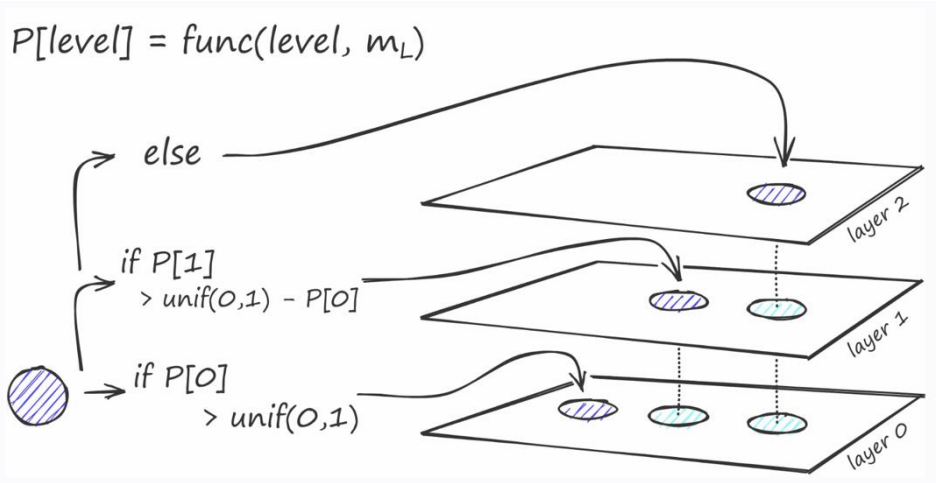
HNSW – Search Algorithm

- We traverse the graph, just like we did in NSW from the top layer to the bottom. At each layer, when the search is terminated, we move on to the next layer.

**Knowledge/images gotten from [3]

HNSW – Graph Construction

- Vectors are iteratively inserted one-by-one.
- L: number of layers
- m_L: the probability of vector insertion at level L ($m_L \sim 0$ means vectors are only inserted at layer 0)



The probability function is repeated for each layer (other than layer 0). The vector is added to its insertion layer and every layer below it.

**Knowledge/images gotten from [3]

HNSW – Graph Construction

- 2 phases:
  1. Starting from the top layer, for a selected vector q, we find ef nearest neighbors of q (here $ef = 1$). After finding a local minimum, we move to the next layer (just like we would do during search). This process is repeated until reaching out layer 0.
  2. The ef parameter is increased to efConstruction (a parameter we set), meaning more nearest neighbors will be returned. In phase two, these nearest neighbors are candidates for the links to the new inserted element q and as entry points to the next layer.

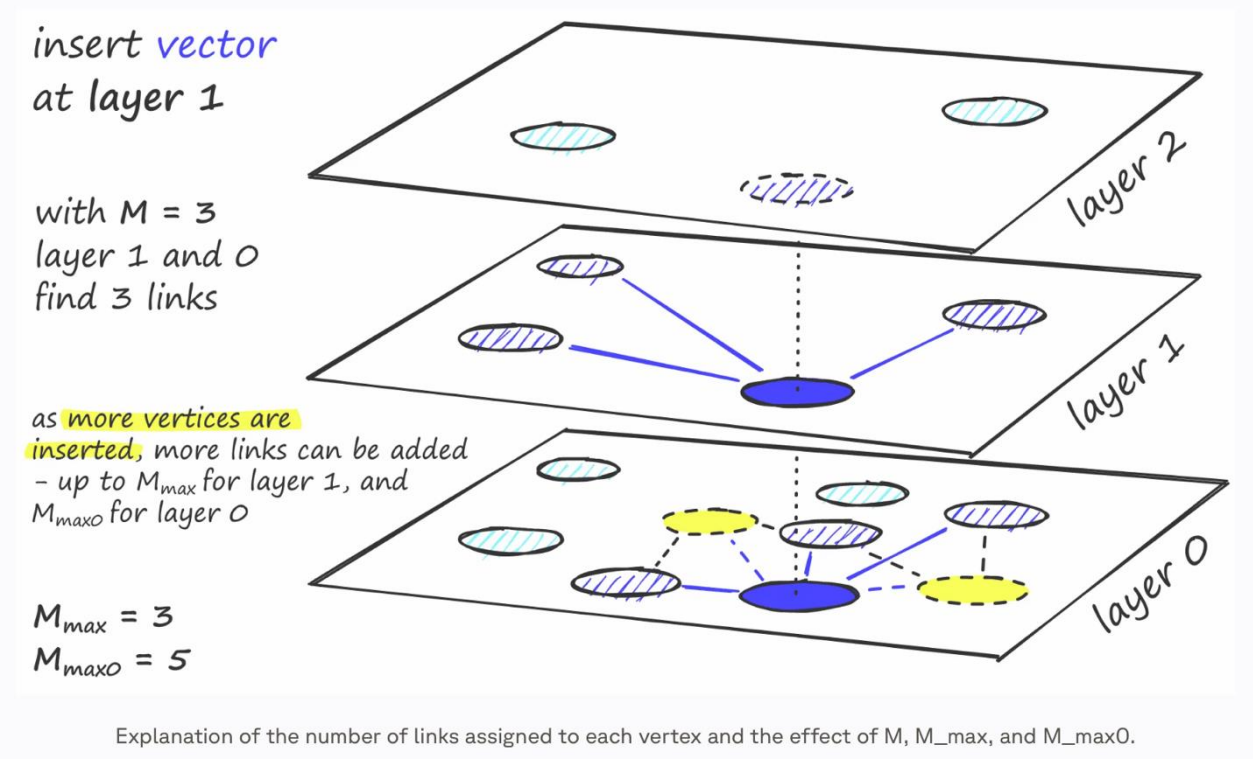
     M neighbors are added as links from these candidates – the most straightforward selection criteria are to choose the closest vectors.

     After working through multiple iterations, there are two more parameters that are considered when adding links. M_max, which defines the maximum number of links a vertex can have, and M_max0, which defines the same but for vertices in layer 0.

**Knowledge/images gotten from [3]

HNSW – Graph Construction

- The stopping condition for insertion is reaching the local minimum in layer 0.



Explanation of the number of links assigned to each vertex and the effect of M, M_max, and M_max0.

Fast Approximate Nearest Neighbor Graph (FANNG) -2016

- Original paper: https://openaccess.thecvf.com/content_cvpr_2016/papers/Harwood_FANNG_Fast_Approximate_CVPR_2016_paper.pdf [5]

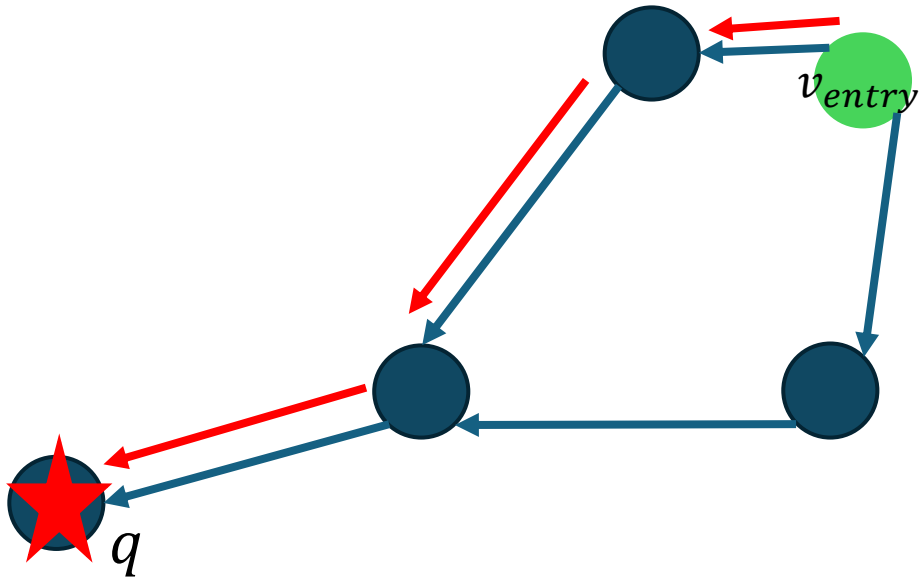
Fast Approximate Nearest Neighbor Graph (FANNG) – The ideal graph

Authors define as the ideal graph, the minimal graph for which the following algorithm always finds the correct solution when the query point matches a vertex of the graph.

- Naïve_downhill_search(V, E, q: query point, v_{entry}:entry point)

1. $for\ each\ (v_{entry}, v) \in E$:
2. $\quad u \leftarrow v$
3. $\quad if\ \delta(q,u) < \delta(q, v_{entry})$:
4. $\quad\quad v_{entry} \leftarrow u$
5. $return\ v_{entry}$

So the ideal graph is just an SNG graph (slide 20)



Fast Approximate Nearest Neighbor Graph (FANNG) – The ideal graph

- The property that the ideal graph must have, is the property of occlusion:

$$\forall s_1, s_2, s_3 \in \mathcal{S}: (s_1, s_2) \text{ occludes } (s_1, s_3) \text{ if } \delta(s_1, s_2) < \delta(s_1, s_3) \; and \; \delta(s_1, s_2) < \delta(s_1, s_3)$$

RNG Graph!

Fast Approximate Nearest Neighbor Graph (FANNG) – The creation of the ideal (RNG) graph

- Naïve_ideal_graph_costruction($\mathcal{S}$):

1. $for\ each\ s_i \in \mathcal{S}$:
2. $\quad e \leftarrow empty\ sorted\ list\ of\ edges$
3. $\quad for\ each\ vertex\ s_j \neq s_i$:
4. $\quad\quad add\ edge\ (s_i, s_j)\ sorted\ by\ \delta(s_i, s_j)$
5. $\quad for\ each\ edge\ (v,u) \in E$:
6. $\quad\quad L \leftarrow length\ of\ (v,u)$
7. $\quad\quad occluded \leftarrow False$
8. $\quad\quad for\ each\ (s_i, k) \in E$:
9. $\quad\quad\quad if\ \delta(k,u) < L$:
10. $\quad\quad\quad\quad occluded \leftarrow True$
11. $\quad\quad if\ not\ occluded$:
12. $\quad\quad\quad add\ edge\ (v,u)\ to\ E$
13. $return\ E$

Fast Approximate Nearest Neighbor Graph (FANNG) – The ideal graph's properties

- Properties of the RNG created by the algorithm presented above:
  1. The graph has relatively low degree.
  2. The angle between edges, must be at least 60 degrees, and thus the edges have to be well spread out (sparse graph).

Fast Approximate Nearest Neighbor Graph (FANNG)

- If the query point does not belong to $\mathcal{S}$, then authors modified the occlusion criterion as:

$$\forall s_1, s_2, s_3 \in \mathcal{S}: (s_1, s_2) \text{ occludes } (s_1, s_3) \text{ if}$$
$$\delta(s_1, s_2) < \delta(s_1, s_3) \; and \; \delta(s_1, s_2)^2 < \delta(s_1, s_3)^2 - 2\tau\delta(s_1, s_2)$$

Where τ is a distance threshold for the NN-query.

They also use a modified version of the Naïve_downhill_search:

Fast Approximate Nearest Neighbor Graph (FANNG) – Search Algorithm 1

- Backtrack_search($\mathcal{S}, E, q, v_{entry}, M$: maximum distance calculations):

1. $X \leftarrow empty\ priority\ queue$
2. $add\ edge\ e_0\ with\ start\ vertex\ v_{entry}\ to\ X$
3. $m \leftarrow 1, n \leftarrow v_{entry}$
4. $while\ m < M$:
5. $\quad e_i = (v, u) \leftarrow remove\ top\ of\ X$
6. $\quad if\ u\ has\ not\ been\ visited\ yet$:
7. $\quad\quad add\ edge\ e_0 = (u, k)\ to\ X$
8. $\quad\quad m \leftarrow m + 1$
9. $\quad\quad if\ \delta(q, u) < \delta(q, n)$:
10. $\quad\quad\quad n \leftarrow u$
11. $\quad v_{entry} \leftarrow v$
12. $\quad if$ i < number of edges with start vertex v:
13. $\quad\quad add\ edge\ e_{i+1}\ with\ start\ vertex\ v\ to\ X$
14. $return\ n$

Fast Approximate Nearest Neighbor Graph (FANNG) – Search Algorithm 2

- Traverse_add($\mathcal{S}$, E, v_1: entry point 1, v_2: entry point 2)

1. $u \leftarrow Naive_downhill_search(\mathcal{S}, E, v_2, v_1)$
2. $if\ u \neq v_2$:
3. $\quad add\ edge\ (u, v_2)\ to\ E$
4. $\quad for\ each\ edge\ e_i = (u, v)$:
5. $\quad\quad if\ (u, v_2)\ occludes\ e_i$:
6. $\quad\quad\quad remove\ edge\ e_i\ from\ E$
7. $\quad\quad\quad k \leftarrow v$
8. $\quad\quad\quad E \leftarrow Traverse_add(\mathcal{S}, E, v_2, v)$
9. $\quad E \leftarrow Traverse_add(\mathcal{S}, E, v_2, u)$
10. $return\ E$

Neighborhood Graph and Tree (NGT) - 2015

- GitHub Repository (Yahoo library): https://github.com/yahoojapan/NGT?tab=readme-ov-file
- Paper: https://arxiv.org/pdf/1810.07355 [6]
- Two construction methods:
  1. Transform KNNG into a bi-directional KNNG (BKNNG), which adds reverse edges to each directed edge of the KNNG.
  2. Incremental construction, like NSW, but using a range search.

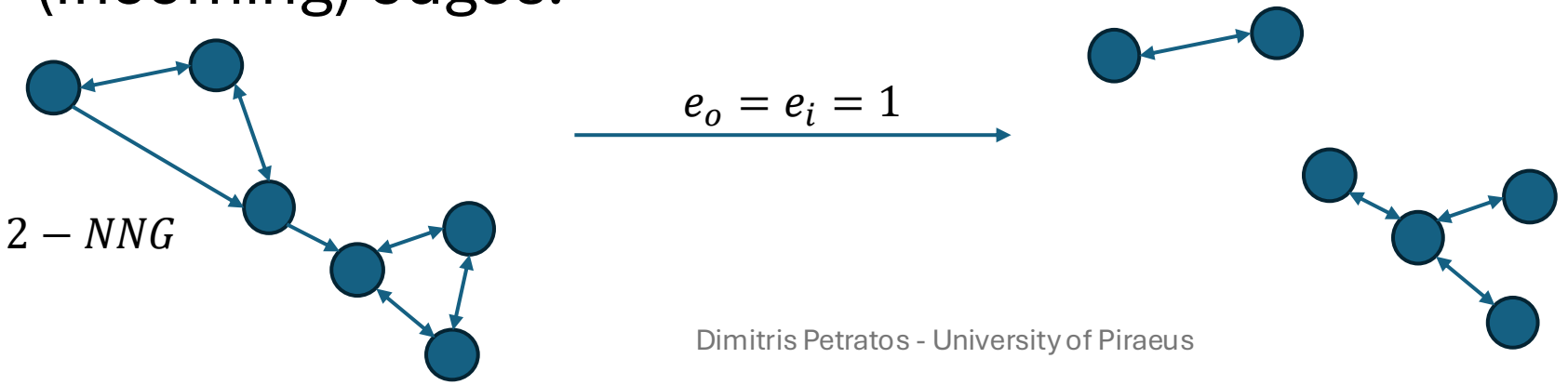
Neighborhood Graph and Tree (NGT) – Adjustment methods

- Authors propose four different types of adjustment methods:
    1. Static degree adjustment method for deriving an adjusted graph from the edges and reversed edges of a KNNG to roughly adjust the indegrees and outdegrees. This method causes some nodes to have a high outdegree, increasing the query time.
    2. Degree adjustment method with constraints for more precisely adjusting the indegrees and outdegrees.
    3. Path adjustment to remove edges.
    4. Dynamic degree adjustment in order to adjust accuracy/search time ratio, based on user preference.

NGT – Static degree adjustment

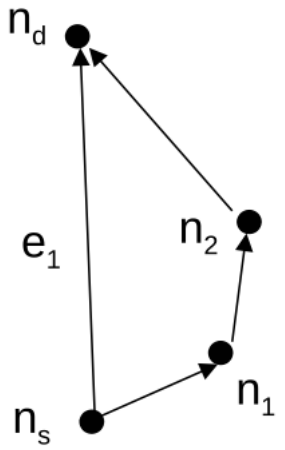
- Let a KNNG, e_o: expected out-degree, e_i: expected in-degree

1. Generate an adjusted graph that has all objects as nodes, without any edges.
2. For every node in the adjusted graph, consider the ascending order of e_o from the KNNG, in terms of length and add the corresponding edges in the adjusted graph. Reverse e_i and add the corresponding edges in the adjusted graph as reverse (incoming) edges.

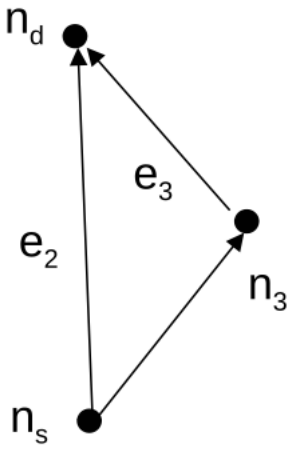


NGT – Path adjustment

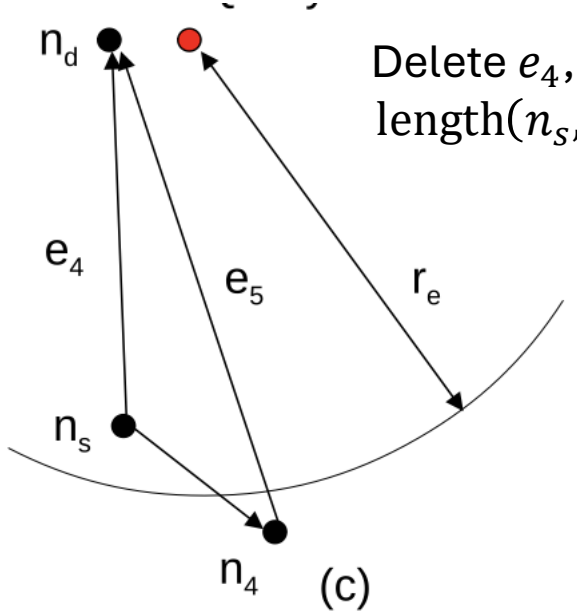
- Let a target edge $e = (u, v)$ (a candidate for removal). If an alternative path $(u, u_1, \dots, u_n, v)$ exists with length >2, then keep (u, v). If an alternative path (u, u_1, v) with length=2, delete (u, v), unless $\delta(u, u_1) > \delta(u, v)$ or $\delta(v, u_1) > \delta(u, v)$.



Keep e_1, since exists $\text{length}(n_s, n_1, n_2, n_d) = 3 > 2$



Delete e_2, since exists $\text{length}(n_s, n_3, n_d) = 2$



Delete e_4, since exists $\text{length}(n_s, n_3, n_d) = 2$

NGT – Graph Construction

- Graph construction includes a combination of applying degree and path adjustments over an approximate KNNG.

Space Partition Tree and Graph (SPTAG) - 2018

- A Microsoft's library: https://github.com/microsoft/SPTAG
- SPTAG hierarchically divides $\mathcal{S}$ into subsets and builds an exact KNNG over each subset.
- SPTAG uses K-Means trees to obtain

seeds near the query point.

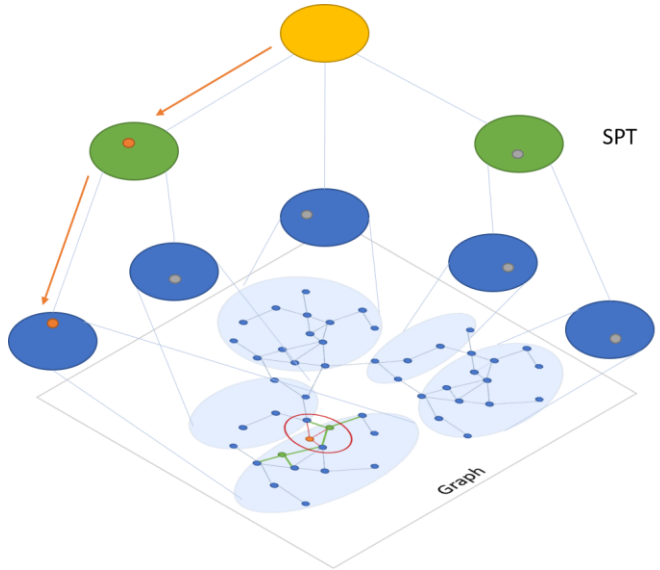


Image source: https://github.com/microsoft/SPTAG

KGraph - 2011

- GitHub Repository: https://github.com/aaalgo/kgraph?tab=readme-ov-file
- (Probably) original paper (I saw a commit to the Github repository from Wei Dong, an author of the following paper + I found that paper as reference to KGraph in [9]): https://www.cs.princeton.edu/cass/papers/www11.pdf [14]
- KGraph uses NN-Descent to reduce KNNG's construction complexity
- KGraph uses the Greedy Search Algorithm in order to search the constructed graph

KGraph – Definitions

- $B_k(v) = v's\ k - nearest\ neighbors, \forall v \in V$
- $R_k(v) = \{u \in V | v \in B_k(u)\}, \forall v \in V$
- $\widetilde{B_k}(v) = B_k(v) \cup R_k(v), \forall v \in V$
- $B(v, r) = \{u \in V | \delta(u, v) \leq r\}$
- A metric space V is said to be growth restricted iif

$$\exists c \in \mathbb{R}: |B(v, 2r)| \leq c \cdot |B(v, r)|, \forall v \in V$$

The smallest such c constant is called the growing constant of V.

Principle: *"a neighbor of a neighbor is also likely to be a neighbor"*

KGraph - Heuristic

- Let c be the growing constant of V and let $k = c^3$. Assume we already have an approximate KNNG B, and let $\forall v \in V, B'(v) = \cup_{v' \in B(v)} B(v')$ be the set of points we explore trying to improve B. If the accuracy of B is reasonably good, such that for certain fixed radius r, $\forall v \in V, B(v)$ contains k neighbors that are uniformly distributed in $B(v, r)$, then assuming independence of certain events and that $k \ll |B\left(v, \frac{r}{2}\right)|$, we can conclude that $B'(v)$ is likely to contain k neighbors in $B(v, \frac{r}{2})$.
- In other words, we expect to halve the maximal distance to the set of approximate k nearest neighbors by exploring $B'(v), \forall v \in V$.

KGraph - NN-Descent

- NNDescent($\mathcal{S}, \delta, k$):
- Output: K-NN list B

1. $B(v) \leftarrow Sample(\mathcal{S}, k) \times \{\infty\}, \forall v \in V$
2. $loop$:
3. $\quad R \leftarrow Reverse(B)$
4. $\quad \tilde{B}(v) \leftarrow B(v) \cup R(v), \forall v \in v$
5. $\quad c \leftarrow 0$
6. $\quad for\ v \in V$:
7. $\quad\quad for\ u_1 \in \tilde{B}(v), u_2 \in \tilde{B}(u_1)$:
8. $\quad\quad\quad l \leftarrow \delta(v, u_2)$
9. $\quad\quad\quad c \leftarrow c + UpdateNN\big(B(v), (u_2, l)\big)$
10. $return\ B\ if\ c = 0$

- Sample($\mathcal{S}$, n):
  1. $Return\ n\ samples\ from\ \mathcal{S}$
- Reverse(B):
  1. $R(v) \leftarrow \{u | < v, \ldots > \in B(u)\}, \forall v \in V$
  2. $return\ R$
- UpdateNN($H, < u, l, \ldots >$):
  1. Update k-nn heap H; return 1 if changed or 0 if not

KGraph – NNDescentFull (optimized)

- NNDescentFull($\mathcal{S}, k, \rho, \delta_0$):
- Output: KNN list B

1. $B[v] \leftarrow Sample(\mathcal{S}, k) \times \{< \infty, true >\}, \forall v \in \mathcal{S}$
2. $loop$
3. $\quad parallel\ for\ v \in \mathcal{S}$:
4. $\quad\quad old[v] \leftarrow all\ items\ in\ B[v]\ with\ a\ false\ flag$
5. $\quad\quad new[v] \leftarrow \rho \cdot k\ items\ in\ B[v]\ with\ a\ true\ flag$
6. $\quad\quad mark\ sampled\ items\ in\ B[v] as\ false$
7. $\quad old' \leftarrow Reverse(old), new' \leftarrow Reverse(new)$
8. $\quad c \leftarrow 0$
9. $\quad parallel\ for\ v \in \mathcal{S}$:
10. $\quad\quad old[v] \leftarrow old[v] \cup Sample(old'[v], \rho \cdot k)$
11. $\quad\quad new[v] \leftarrow new[v] \cup Sample(new'[v], \rho \cdot k)$
12. $\quad\quad for\ u_1, u_2 \in new[v], u_1 < u_2\ or\ u_1 \in new[v], u_2 \in old[v]$:
13. $\quad\quad\quad l \leftarrow \delta(u_1, u_2)$
14. $\quad\quad\quad c \leftarrow c + UpdateNN(B[u_1], < u_2, l >)$
15. $\quad\quad\quad c \leftarrow c + UpdateNN(B[u_2], < u_1, l >)$
16. $\quad if\ c < \delta_0 \cdot \rho \cdot k$:
17. $\quad\quad return\ B$

EFANNA - 2016

- Original paper: https://arxiv.org/pdf/1609.07228 [7]
- Index: Randomized truncated KD-tree + approximate KNNG

EFANNA – Tree Building Algorithm

- TreeBuildingAlgorithm(D: dataset, T: the number of trees, K: the number of points in a leaf node)
- Output: The randomized truncated KD-tree S

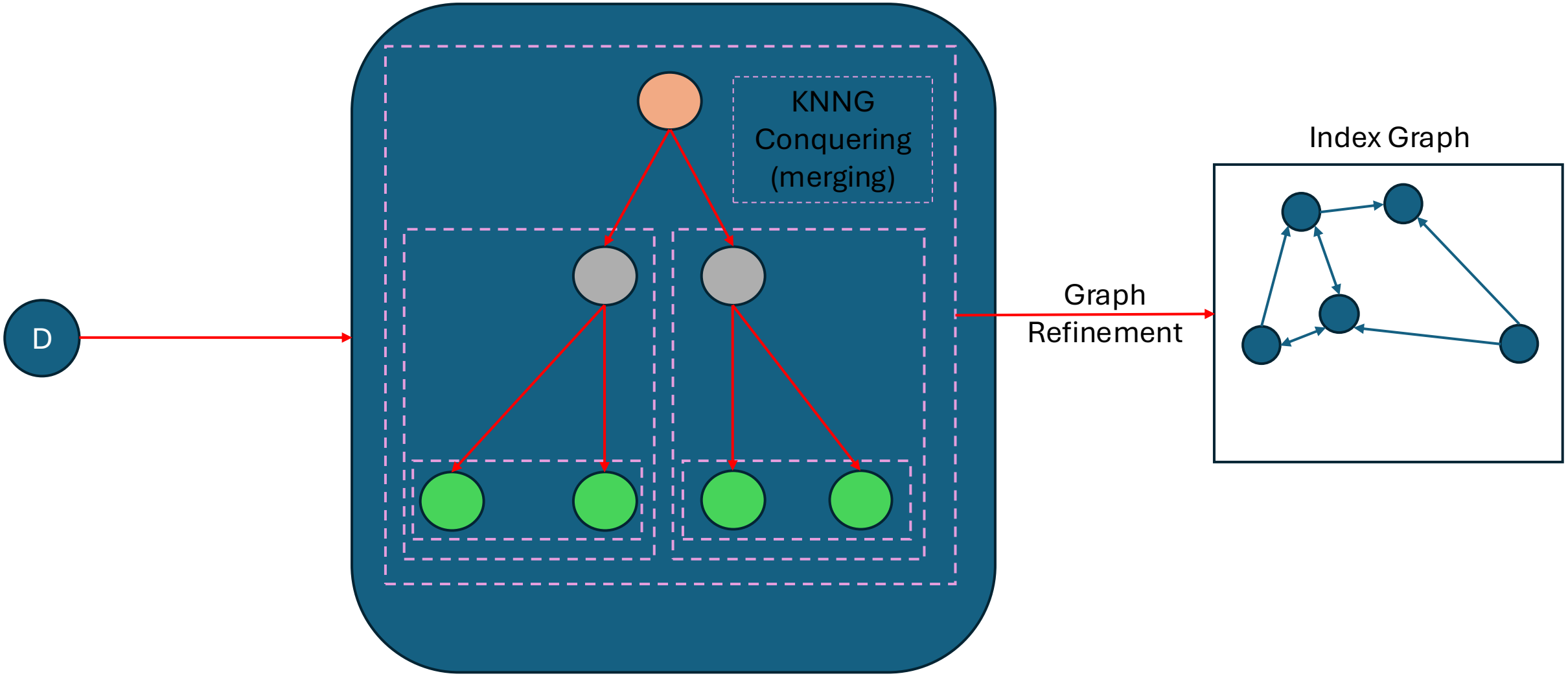
1. $S \leftarrow \emptyset$
2. $BuildTree(Node, PointSet)$:
3. $\quad if\ size\ of\ PointSet < K$:
4. $\quad\quad return$
5. $\quad else$:
6. $\quad\quad d \leftarrow Randomly\ choose\ dimension$
7. $\quad\quad mid \leftarrow calculate\ the\ mean\ over\ PointSet\ on\ dimension\ d$
8. $\quad\quad LeftHalf, RightHalf \leftarrow Divide\ PointSet\ according\ to\ mid$
9. $\quad\quad BuildTree(Node.LeftHalf, LeftHalf)$
10. $\quad\quad BuildTree(Node.RightHalf, RightHalf)$
11. $for\ i$ in range(T):
12. $\quad BuildTree(Root_i, D)$
13. $\quad add\ Root_i\ to\ S$

EFANNA – kNN Graph Initialization

- HierarchicalDivideAndConquerAlgorithm(D: dataset, k: the k in the kNN graph, S: randomized truncated KD-tree, Dep: conquer-to depth)
- Output: approximate kNN graph G

$G = \emptyset$
for all point i in D:
    candidate pool $C \leftarrow \emptyset$
    for all binary tree t in S:
        Search in tree t with point i to the leaf node
        Add all the points in the leaf node to C
        $d \leftarrow$ depth of the leaf node
        while $d > Dep$:
            $d \leftarrow d - 1$
            $Depth - first - search$ in the tree t with point i to depth d. Suppose N is the
            non $-$ leaf node on the search path with depth d. Suppose Sib is the child node
            of N. And Sib is not on the search path of point i.
            $Depth - first - search$ to the leaf node in the subtree of Sib with point i. Add
            all the points in the leaf node to C.
    Reverse K closest points to i in C
    Add C to G

EFANNA – Graph Construction



EFANNA – Search Algorithm

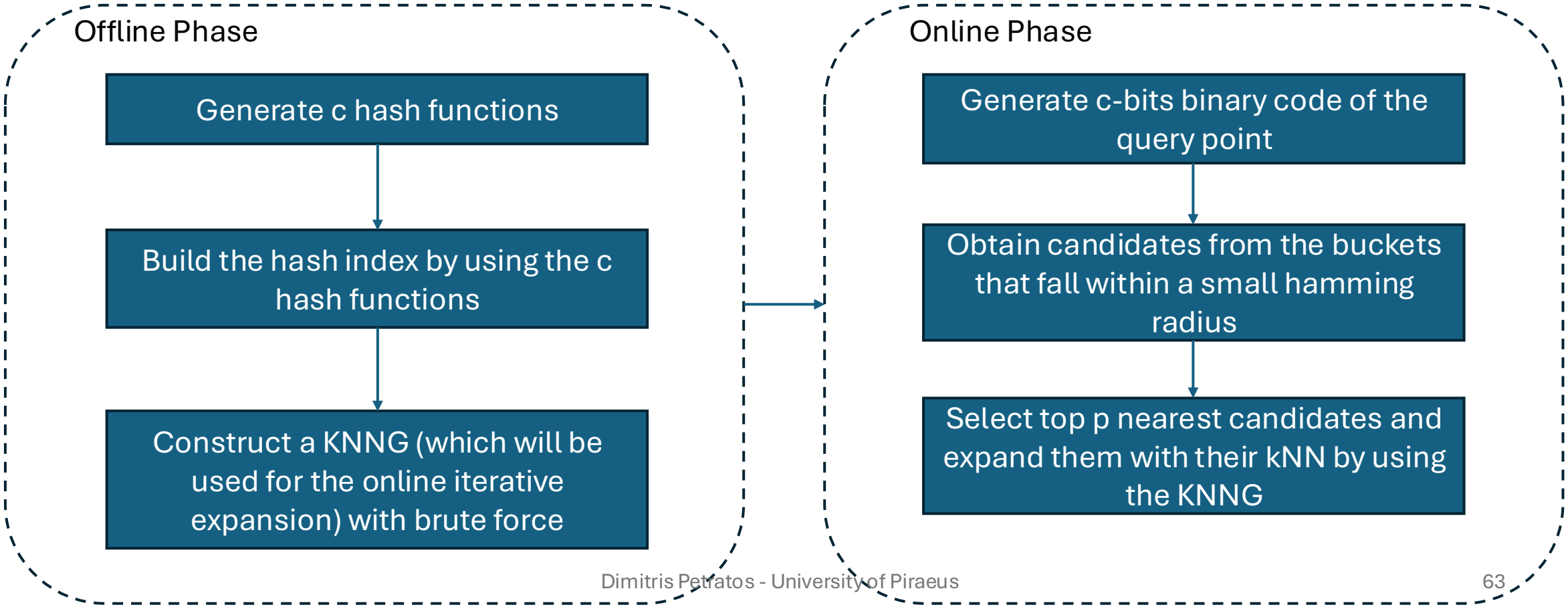
- EFANNASearchAlgorithm(D, q, k, G(V, E), S_{tree}, P: candidate pool size, E: expansion factor, I: iteration number)

1. $iter \leftarrow 0, NodeList \leftarrow \emptyset, candidate\ set\ C \leftarrow \emptyset$, S_{leaf}: maximal number of points of leaf node
2. $N_{tree}: number\ of\ trees, N_{node} \leftarrow \frac{P}{S_{leaf} \cdot N_{tree}} + 1$
3. $for\ all\ tree\ i\ in\ S_{tree}:$
4. $\quad Depth - first - search\ i\ for\ top\ N_{node}\ closest\ leaf\ nodes\ according\ to$
5. $\quad respective\ tree\ search\ criteria, add\ to\ NodeList$
6. $add\ the\ points\ belonging\ to\ the\ nodes\ in\ NodeList\ to\ C$
7. $keep\ E\ points\ in\ C\ which\ are\ closest\ to\ q$
8. $while\ iter < I:$
9. $\quad CC \leftarrow \emptyset$ #candidate set
10. $\quad for\ all\ points\ n\ in\ C:$
11. $\quad\quad S_n\ is\ the\ neighbors\ of\ point\ n\ based\ on\ G$
12. $\quad\quad for\ all\ points\ nn\ in\ S_n:$
13. $\quad\quad\quad if\ nn\ hasn't\ been\ checked: put\ nn\ to\ CC$
14. $\quad move\ all\ the\ points\ in\ CC\ to\ C\ and\ keep\ P\ points\ in\ C\ which\ are\ closest\ to\ q$
15. $\quad iter \leftarrow iter + 1$
16. $return\ ANNS\ as\ the\ closest\ k\ points\ to\ q\ in\ C$

IEH (Iterative Expanding Hashing) - 2014

Seed acquisition through hash functions

- Original paper: https://ieeexplore.ieee.org/document/6734715 [8]



Diversified Proximity Graph (DPG) - 2019

- Original paper: https://arxiv.org/pdf/1610.02455 [9]
- Index Graph: Diversified KNNG
- Search: The same as KGraph

DPG – Diversified Proximity Graph

- GreedyConstruction($\mathcal{S}, \mathrm{k}, \mathrm{K} = 2\mathrm{k}$):

1. $E \leftarrow \emptyset$
2. $for\ every\ p \in \mathcal{S}$:
3. $\quad L[p] \leftarrow find\ K\ nearest\ neightbors\ of\ p$
4. $\quad S = \{argmin_{x \in L[p]} \delta(x, p)\}$
5. $\quad for\ i\ in\ range(k-1)$:
6. $\quad\quad x \leftarrow argmin_{y \in L[p] \setminus S} \frac{1}{|S-1|} \sum_{y_i, y_j \in S \setminus \{y\}} \theta(y_i, y_j)$ # θ is the angular distance
7. $\quad\quad S \leftarrow S \cup \{x\}, L[p] \leftarrow L[p] \setminus \{x\}$
8. $\quad for\ each\ u \in S$:
9. $\quad\quad E \leftarrow E \cup \{(p, u), (u, p)\}$
10. $return\ E$

Navigating Spreading-out Graph (NSG) - 2019

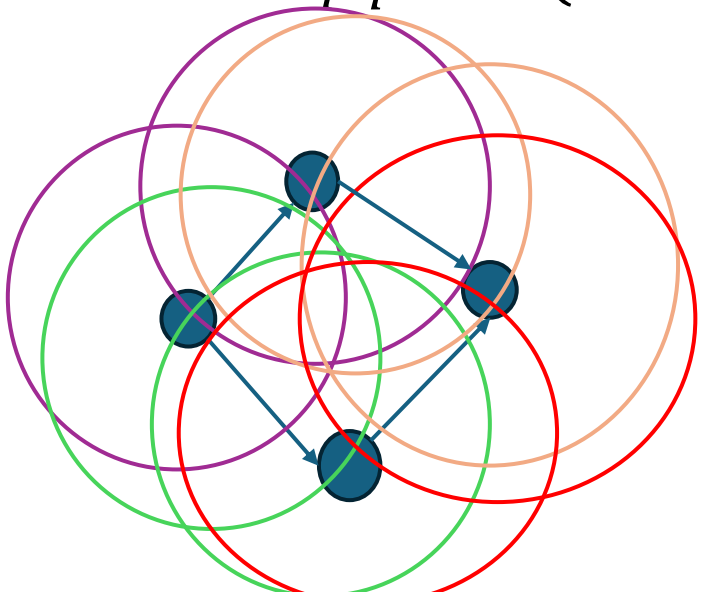
- Original paper: https://www.vldb.org/pvldb/vol12/p461-fu.pdf [10]
- Authors propose:
  - a new graph - Monotonic Relative Neighborhood Graph (MRNG)
    - MRNG guarantees a low average search time
  - An approximation for MRNG – Navigating Spreading-out Graph (NSG)

NSG - Monotonic Search Networks (MSNET)

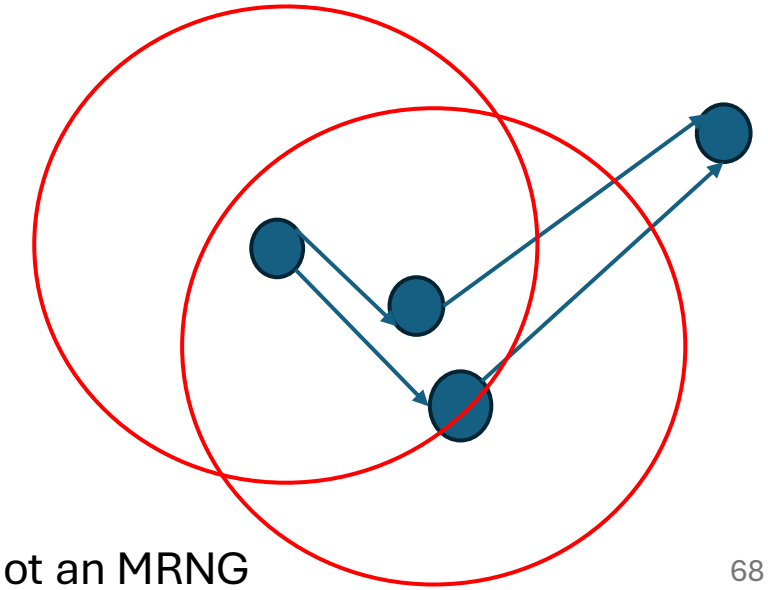
- Definition (Monotonic Path): Given a finite point set $\mathcal{S}$, $\mathrm{p}, \mathrm{q} \in \mathcal{S}$ and $G = (V, E)$ a graph defined on $\mathcal{S}$, let $v_1, \ldots, v_k$ (where $v_1 = p, v_k = q$) denote a path from p to q in G, such that ($v_i v_{i+1} \in E, \forall i = 1, \ldots, k-1$). We call $v_1, \ldots, v_k$ a monotonic path iff $\forall i = 1, \ldots, k-1$ $\delta(v_i, q) > \delta(v_{i+1}, q)$.
- Definition (Monotonic Search Network): Given a finite point set $\mathcal{S}$, a graph defined on $\mathcal{S}$ is said to be a monotonic search network iff there exists at least one monotonic path from p to q, $\forall p, q \in \mathcal{S}$.

NSG – Monotonic Relative Neighborhood Graph (MRNG)

- Definition (MRNG): Given a finite point set $\mathcal{S}$, an MRNG is a directed graph with the set of edges satisfying the following property: for any $(p,q) \in E, (p,q) \in MRNG$ iff $lune_{pq} \cap \mathcal{S} = \emptyset$ or $\forall r \in \left(lune_{pq} \cap \mathcal{S}\right), (p,q) \notin MRNG$, where
- $lune_{pq} = B\left(p, \delta(p,q)\right) \cap B\left(q, \delta(p,q)\right)$.

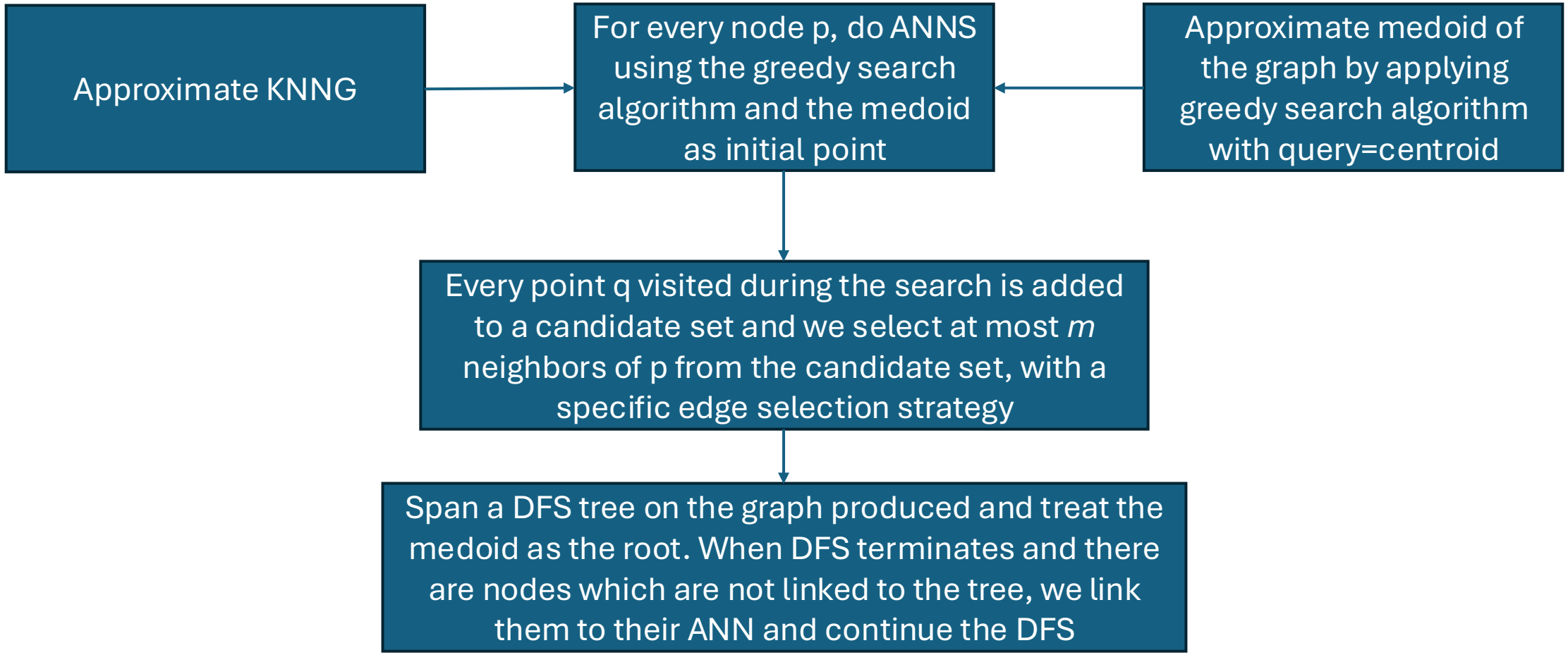


MNRG



Not an MRNG

NSG - Construction



NSG - Search

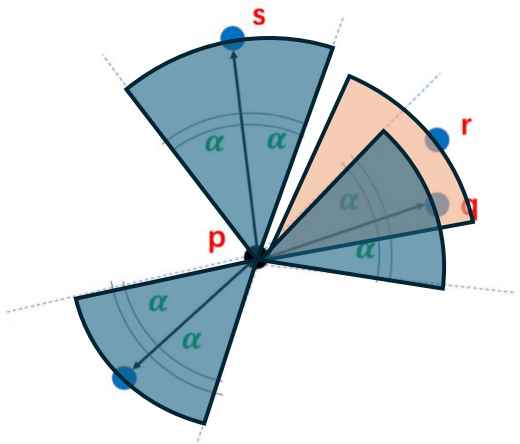
- Authors use the greedy search algorithm, starting from the approximate medoid of the graph.

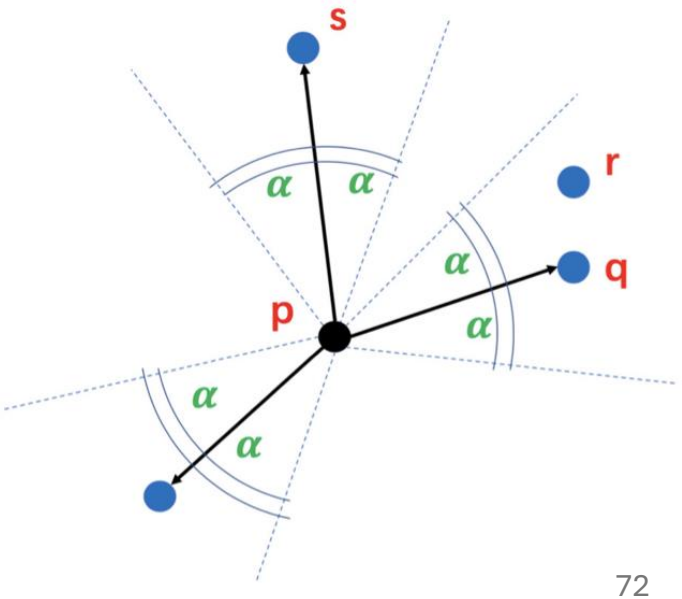
Navigating Satellite System Graph (NSSG) - 2021

- Original paper: https://ieeexplore.ieee.org/document/9383170 [11]
- Authors claim that NSGs (their own work), do not have theoretical guarantees on NNS when the query is not indexed in the database.
- Authors use an SSG, a special type of an MRNG (like the NSG of their previous work).
- Authors upgrade the NSG pruning algorithm, including angle-based information.

NSSG – Satellite System Graph (SSG)

- Definition (SSG): Given a graph $G = (V, E)$, in E, an SSG is defined as the set of directed edges satisfying the property: $\forall (p, q) \in E, (p, q) \in SSG$ iff $Cone((p, q), \alpha \cap B\big(p, \delta(p, q)\big) \cap \mathcal{S} = \emptyset$ or $\forall r \in Cone((p, q), \alpha \cap B\big(p, \delta(p, q)\big) \cap \mathcal{S}, (p, q) \notin SSG$, where $0^o \leq \alpha \leq 60^o$ is a hyper-parameter.





NSSG - SSG

- Properties of SSGs:
  1. (Theorem) An SSG is an MSNET (like the NSG).
  2. (Theorem) For $\mathcal{S}$ (randomly distributed in $\mathbb{R}^d$), any query $q \notin \mathcal{S}$ and nearest neighbor $r \in \mathcal{S}$, the probability that each step on the search path is monotonic to both r and q is $0.5 + \varepsilon, 0 < \varepsilon \leq 0.5$, under the condition that for any neighbor s of the node p in the current search step, $\delta(p, s) < \delta(p, q)$. Further, $\varepsilon = 0.5$ when we set $alpha \leq 30^o$ for the SSG. (that's a strong argument to guarantee monotonicity of an SSG)
  3. Larger α leads to a sparser graph.

NSSG – Navigating SSG

- NSSGs are build as pruned SSGs.
- NSSGs do not have the theoretical “good” properties of SSGs, but under some empirical studies, authors claim that NSSGs’ performance is close to theoretical analysis.

Vamana (DiskANN) - 2019 [Microsoft]

- Original paper: https://papers.nips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf [12]
- Modified Greedy Search algorithm (Beam Search)

Vamana – Pruning Algorithm

- $RobustPrune(G = (V, E), p: a\ point\ in\ \mathcal{S}, V: candidate\ set,$
  $a \geq 1: distance\ threshold, R > \log(\mathcal{S}): degree\ bound)$

1. $V \leftarrow \left(V \cup N_{out}(p)\right) \setminus \{p\}$
2. $N_{out}(p) \leftarrow \emptyset$
3. $while\ V \neq \emptyset:$
4. $\quad p^* \leftarrow armin_{p' \in V} \delta(p, p')$
5. $\quad N_{out}(p) \leftarrow N_{out}(p) \cup \{p^*\}$
6. $\quad if\ |N_{out}(p)| = R:$
7. $\quad\quad break;$
8. $\quad if\ p' \in V:$
9. $\quad\quad if\ a \cdot \delta(p^*, p') \leq \delta(p, p'):$
10. $\quad\quad\quad remove\ p'\ from\ V$
11. $return\ G$

Vamana – Indexing Algorithm

- $VamanaIndexingAlgorithm(\mathcal{S}, \mathrm{a}, \mathrm{L}, \mathrm{R})$:
  1. $initialize\ G\ to\ a\ random\ R - rectangular\ directed\ graph$
  2. $s \leftarrow the\ medoid\ of\ \mathcal{S}$
  3. $\sigma \leftarrow random\ permutation\ of\ 1, \dots, |\mathcal{S}|$
  4. $for\ i\ in\ range(1, |\mathcal{S}|)$:
  5. $\quad [L; V] \leftarrow GreedySearch(s, \sigma(i), 1, L)$
  6. $\quad run\ RobustPrune(\sigma(i), V, a, R)\ to\ update\ out - neighbors\ os\ \sigma(i)$
  7. $\quad for\ all\ j \in N_{out}(\sigma(i))$:
  8. $\quad\quad if\ |N_{out}(j) \cup \{\sigma(i)\}| > R$:
  9. $\quad\quad\quad run\ RobustPrune(j, N_{out}(j) \cup \{\sigma(i)\}, a, R) to\ update\ out - neighbors\ of\ j$
  10. $\quad\quad else$:
  11. $\quad\quad\quad N_{out}(j) \leftarrow N_{out}(j) \cup \{\sigma(i)\}$

Vamana – Vamana Indexing Algorithm (example)

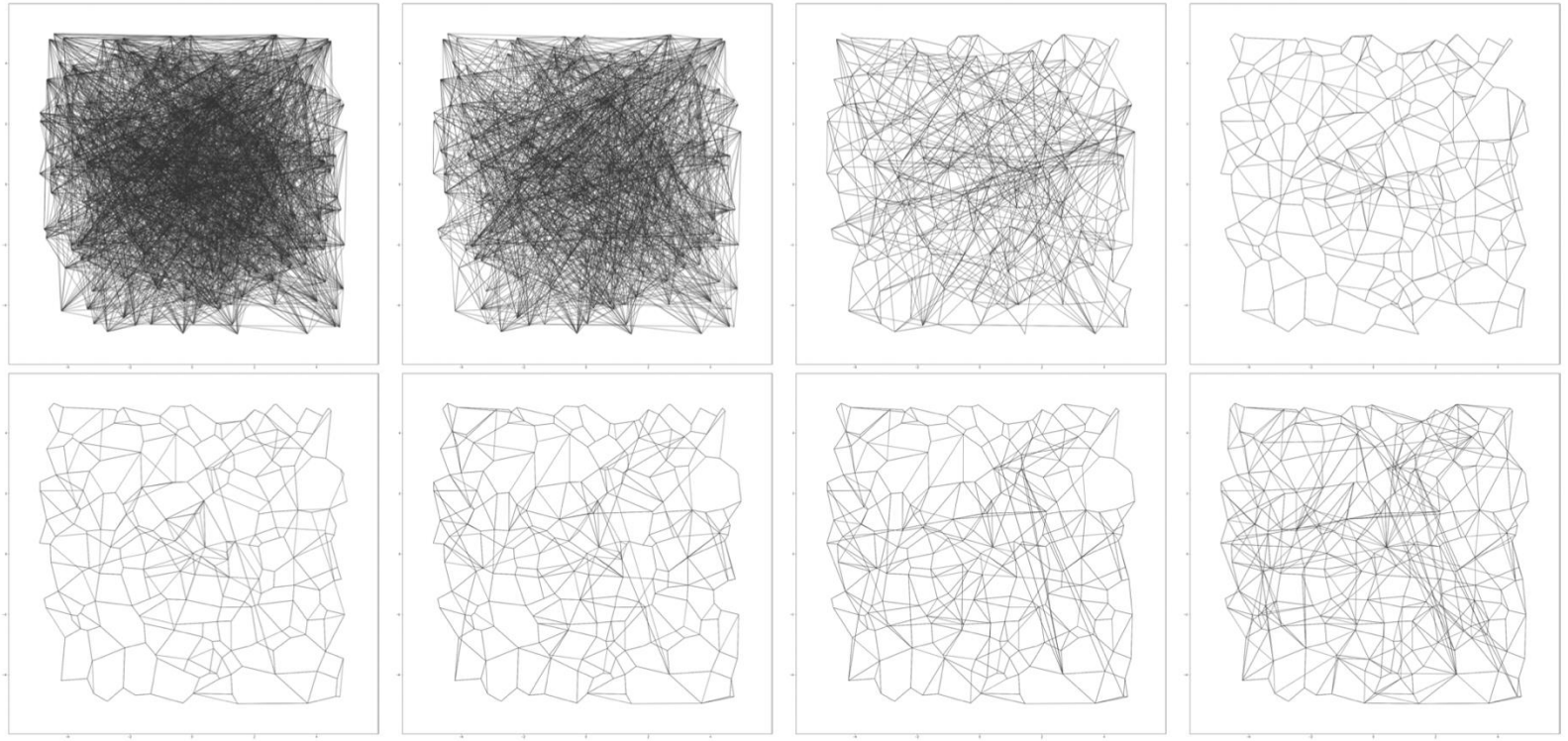
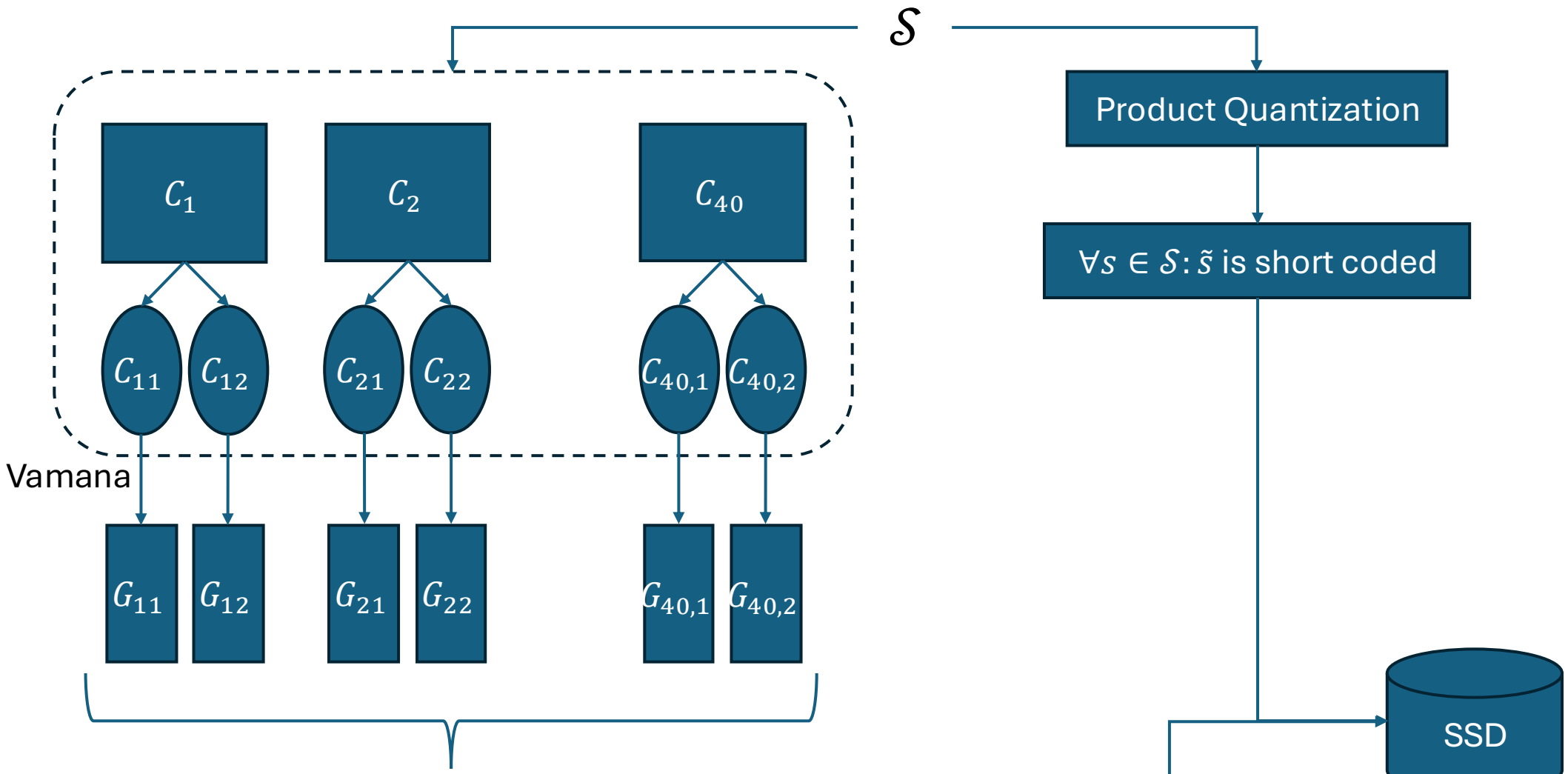


Figure 1: Progression of the graph generated by the Vamana indexing algorithm described in Algorithm 3 on a database with 200 points in 2 dimensions. Notice that the algorithm goes through the first pass with $\alpha = 1$, followed by the second pass where it introduces long range edges.

Vamana – DiskANN (Index)



HCNNG - 2019

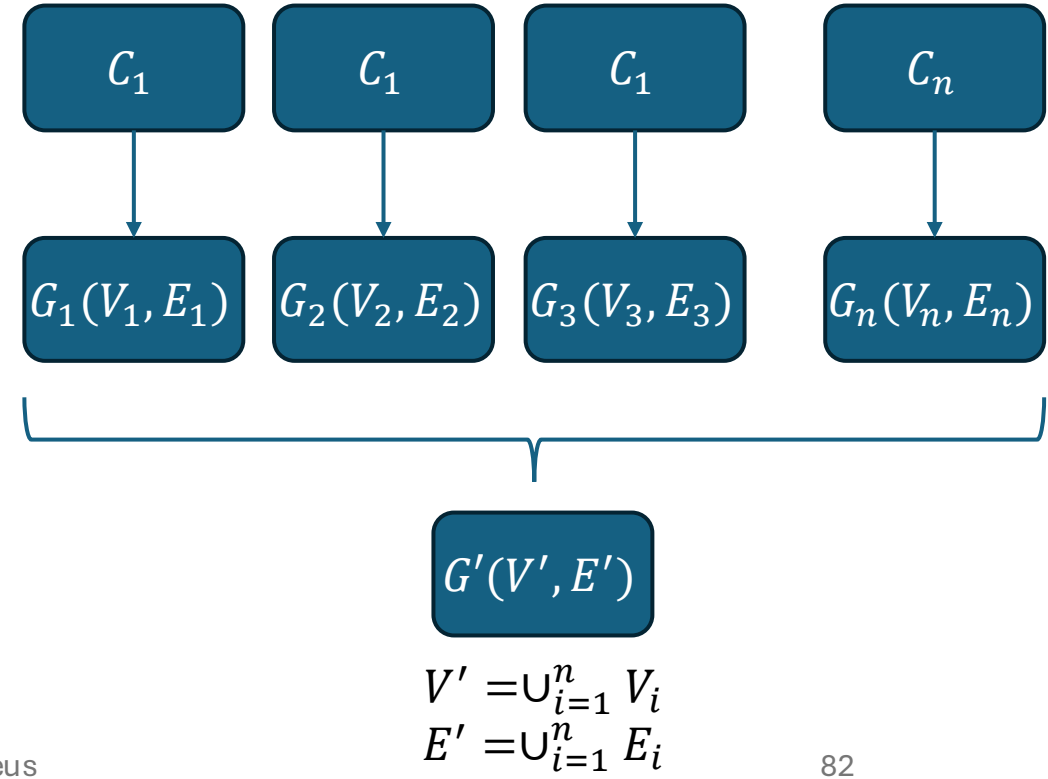
- Original paper: https://www.sciencedirect.com/science/article/pii/S0031320319302730 [13]
- Graph construction: MST-based
- Initialization of entry points is done using KD-trees
- Multiple hierarchical clustering procedures
  - Objective: generate graph with good connectivity, without unnecessary multiple paths

HCNNG – Hierarchical Clustering

- $HierarchicalClustering(\mathcal{S}$: data points, n: min size of clusters)
  1. $N \leftarrow \mathcal{S}$
  2. $E \leftarrow \emptyset$
  3. $if\ N < n$:
  4. $\quad E \leftarrow MST3(\mathcal{S})$
  5. $else$:
  6. $\quad p_1, p_2 \leftarrow select\ randomly\ points\ from\ \mathcal{S}$
  7. $\quad P_1 \leftarrow \{p \in \mathcal{S} | \delta(p, p_1) < \delta(p, p_2)\}$
  8. $\quad P_2 \leftarrow \{p \in \mathcal{S} | \delta(p, p_1) \geq \delta(p, p_2)\}$
  9. $\quad E_1 \leftarrow HierarchicalClustering(P_1, n)$
  10. $\quad E_2 \leftarrow HierarchicalClustering(P_2, n)$
  11. $\quad E \leftarrow E_1 \cup E_2$
  12. $return\ E$

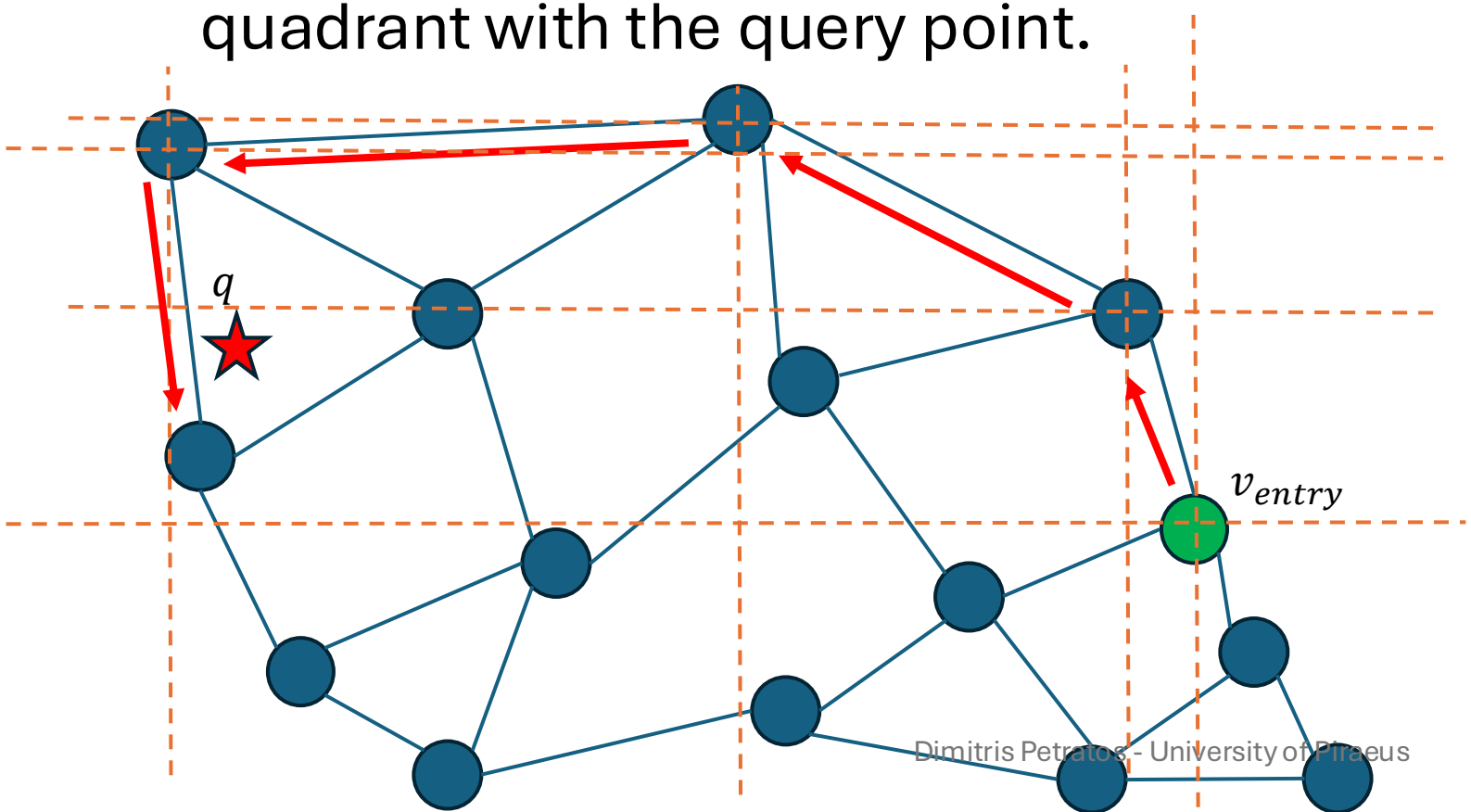
HCNNG – Graph Fusion

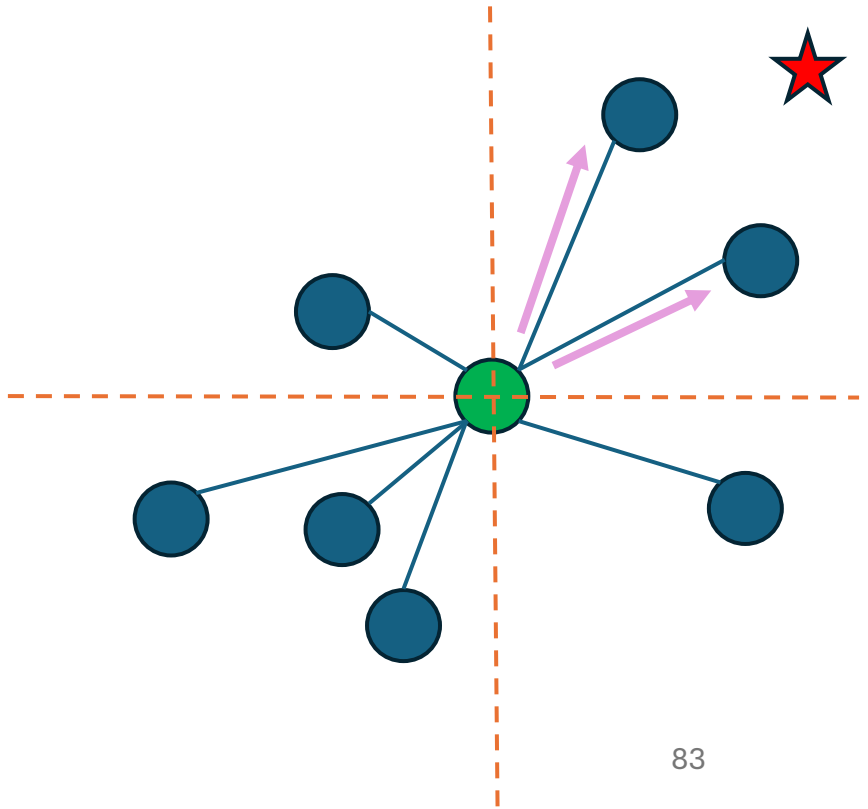
- $CreateHCNNG(\mathcal{S}$: data points, m: number of random clustering, n: minimum size of clusters)
  1. $E \leftarrow \emptyset$
  2. $for\ i\ in\ range(m)$:
  3. $\quad E_i \leftarrow HierarchicalClustering(\mathcal{S}, n)$
  4. $\quad E \leftarrow E \cup E_i$
  5. $return\ E$



HCNNG – Search Algorithm

- The main innovation regarding the search algorithm, is the fact that authors propose a search of points belonging to the same quadrant with the query point.





Metrics

Name	Formula	Paper
Recall	$\frac{\lvert Actual\ knns \cap retrieved\ knns \rvert}{k}$	[2, 4, 5, 6, 7, 8, 9, 12, 13, 14]
Accuracy		[7, 10]
Speedup	$\frac{Collection\ size}{Number\ of\ distance\ calculations}$	[9, 12, 13]
Queries per Second	How much queries an algorithm can process per second at a given precision	[10, 11]
Precision	$\frac{\lvert R \cap G \rvert}{\lvert G \rvert}$	[6, 10, 11]
Speedup over linear search/error rate		[5]
Number of computations		[6]
Query time		[4, 5, 6]
Scan rate	$\frac{\#\ similarity\ calculations}{N(N-1)/2}$	[14]

Competitors

Paper	Competitors
NSW [2]	NAPP [15]
HNSW [4]	NSW [2], Annoy [16], VP-tree, FLANN [17]
FANNG[5]	Linear search, FLANN [17], NSW [2]
NGT [6]	FLANN [17], HNSW [4]
EFANNA [7]	Linear search, FLANN [17], GNNS [19], KGraph [14], IEH [8]
IEH [8]	LSH, KLSH, SH, AGH, SPH (all hashing algorithms)
DPG [9]	QALSH, SRS, SGH, AGH, NSH, SH, OPQ, NAPP, FLANN, Annoy, VP-tree, SW, HNSW, RCT, KGraph (all kind of types)
NSG [10]	FANNG [5], KGraph [14], EFANNA [7], DPG[9], HNSW [4]
NSSG [11]	NSG [10], HNSW [4], KGraph [14], DPG [9]
Vamana [12]	HNSW [4], NSG [10]
HCNNG [13]	KGraph [14], FLANN [17], FANNG [5], HNSW [4], NSW [2], Wang2012 [22], OPQ, AQ
KGraph [14]	RLB [18]

Datasets used

Dataset name	Dimensionality/Paper	Description
GloVe	100 [4, 6, 9, 11, 13], 300 [6]	Text descriptors
GIST1M	960 [5, 6, 7, 9, 10, 11, 12, 13], 384 [8]	Histopathological descriptors
SIFT1M	128 [4, 6, 7, 8, 9, 10, 11, 12, 13]	Image descriptors
DEEP1M	96 [12]	Deep learning descriptors
RAND4M1	128 [10]	Synthetic dataset U(0,1)
RAND4M2	128 [10]	Synthetic dataset U(0,1)
GAUSS5M	128 [10]	Synthetic dataset N(0,3)
Crawl	300 [11]	Text descriptors
Deep100M	96 [4, 11]	Media descriptors
CoPHiR	208 [2], 272 [4]	Image descriptors
Synthetic Dataset	50 [2]	Synthetic dataset uniformly sampled
CIFAR 10	3072 [8]	Image descriptors

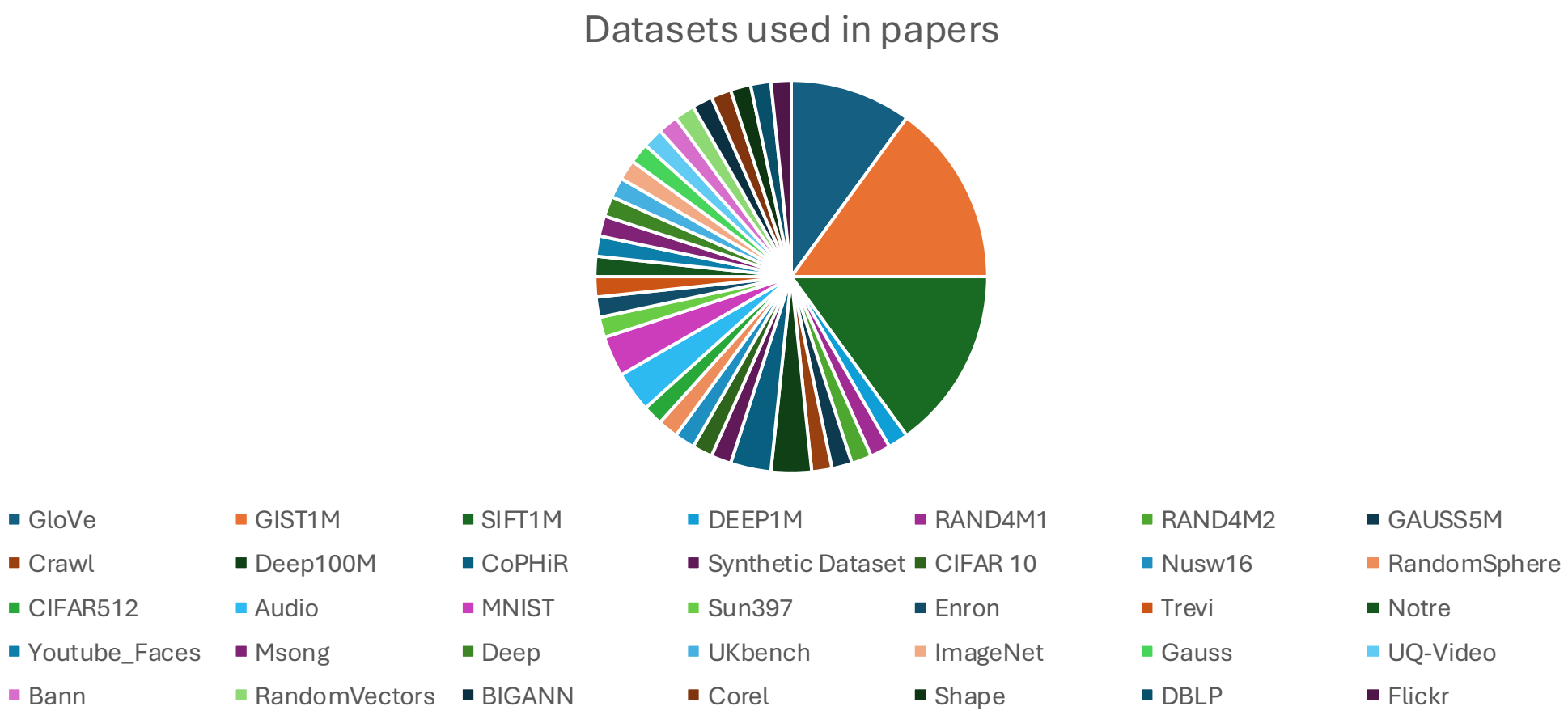
Datasets used

Dataset name	Dimensionality/Paper	Description
Nusw16	500 [9]	Text descriptors
RandomSphere	100 [9]	Points randomly sampled from sphere
CIFAR512	512 [9]	Image descriptors
Audio	192 [9, 14]	Audio descriptors
MNIST	784 [4, 9]	Image descriptors
Sun397	512 [9]	Image descriptors
Enron	1396 [9]	Text descriptors
Trevi	4096 [9]	Image descriptors
Notre	128 [9]	Image descriptors
Youtube_Faces	1770 [9]	Video descriptors
Msong	420 [9]	Audio descriptors
Deep	256 [9]	Image descriptors
UKbench	128 [9]	Image descriptors
ImageNet	150 [9]	Image descriptors

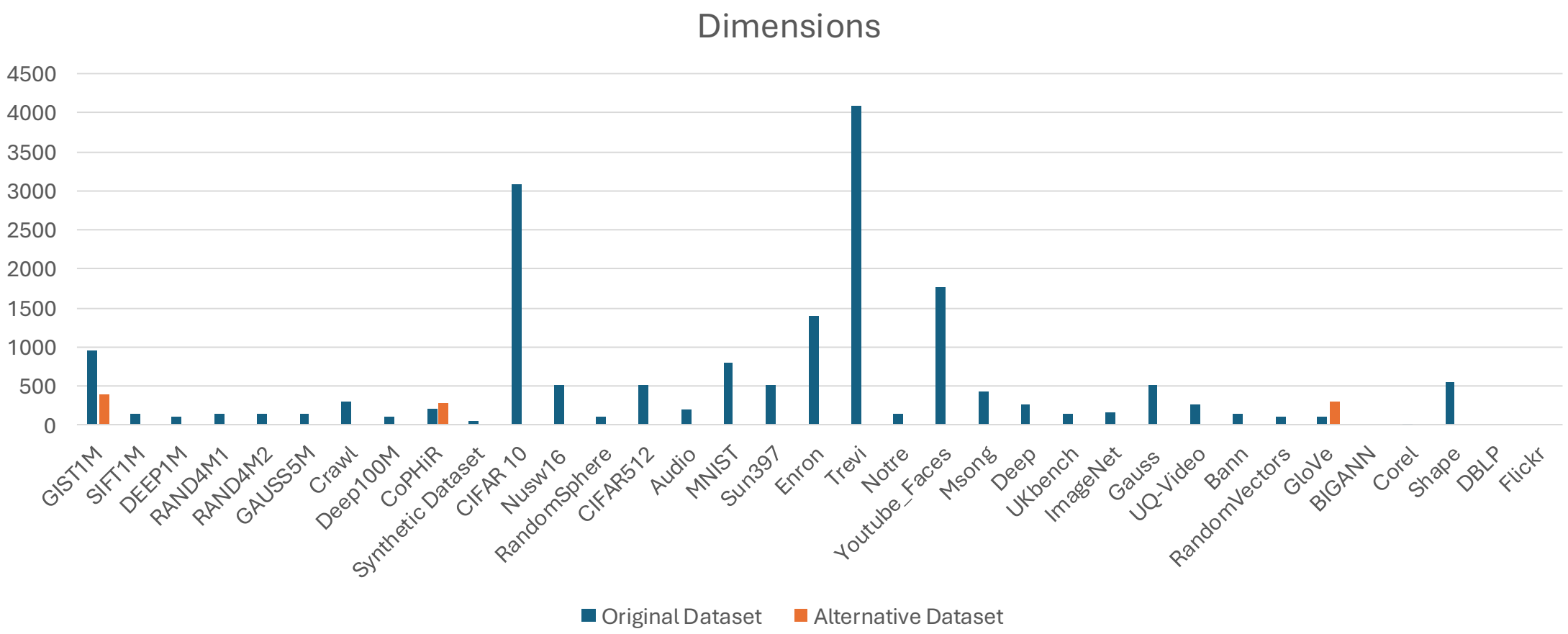
Datasets used

Dataset name	Dimensionality/Paper	Description
Gauss	512 [9]	Synthetic dataset N(0,1)
UQ-Video	256 [9]	Video descriptors
Bann	128 [9]	Image descriptors
RandomVectors	96 [4]	Random vectors in hypercube
BIGANN	? [5]	(probably) Image descriptors
Corel	14 [14]	Image descriptors
Shape	544 [14]	Shape descriptors
DBLP	N/A [14]	Text descriptors
Flickr	N/A [14]	Image descriptors

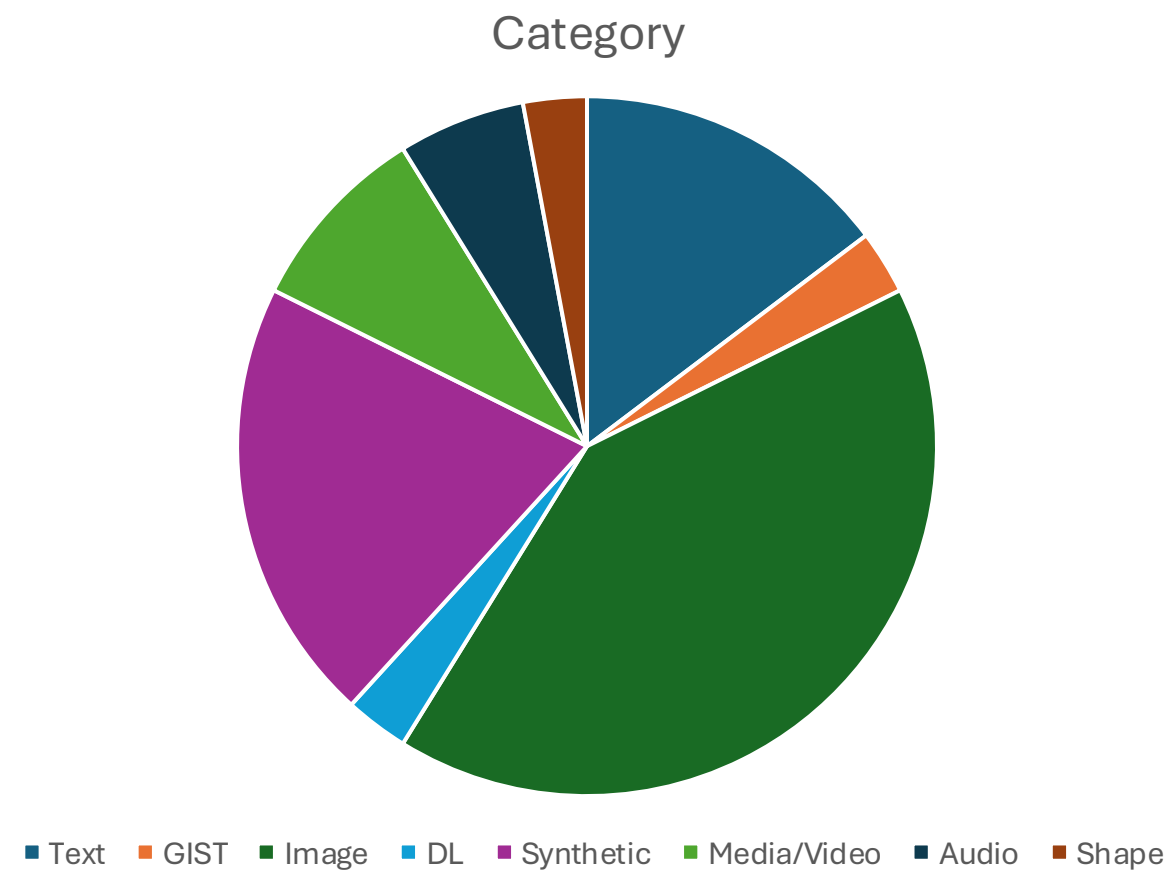
Datasets used



Datasets used



Datasets used



Other approaches

- Tree-based ANNS
- Hashing-based ANNS
- Product quantization

References

1. Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. Proc. VLDB Endow. 14, 11 (July 2021), 1964–1978. https://doi.org/10.14778/3476249.3476255
2. Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. Information Systems 45 (2014), 61–68.
3. https://www.pinecone.io/learn/series/faiss/hnsw/
4. Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE [89] transactions on pattern analysis and machine intelligence (2018).
5. Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 5713–5722.

References

6. Masajiro Iwasaki. 2016. Pruned bi-directed k-nearest neighbor graph for prox- imity search. In International Conference on Similarity Search and Applications. Springer, 20–33.
7. Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. arXiv preprint arXiv:1609.07228 (2016).
8. Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. IEEE transactions on cybernetics 44, 11 (2014), 2167–2177.
9. Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. IEEE Transactions on Knowledge [85] and Data Engineering (2019).
10. Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. Proc. VLDBEndow.12,5(2019),461–474. https://doi.org/10.14778/3303753.3303754

References

11. Cong Fu, Changxu Wang, and Deng Cai. 2021. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. IEEE Transactions on Pattern Analysis and Machine Intelligence (2021).
12. Suhas Jayaram Subramanya, F Devvrit, HV Simhadri, R Krishnawamy, and R Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In NeurIPS 2019. 1771–13781.
13. Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. Pattern Recognition 96 (2019), 106970.
14. Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In Proceedings of the 20th international conference on World wide web (WWW '11). Association for Computing Machinery, New York, NY, USA, 577–586. https://doi.org/10.1145/1963405.1963487

References

15. E.S. Tellez, E. Chávez, G. Navarro, Succinct nearest neighbor search, in: Proceedings of SISAP, 2011.
16. https://github.com/spotify/annoy
17. M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227-2240, 1 Nov. 2014, doi: 10.1109/TPAMI.2014.2321376
18. J. Chen, H. ren Fang, and Y. Saad. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. Journal of Machine Learning Research, 10:1989–2012, 2009.
19. K.Hajebi,Y.Abbasi-Yadkori,H.Shahbazi,andH.Zhang.Fastap- proximate nearest-neighbor search with k-nearest neighbor graph. In *IJCAI 2011, Proceedings of the International Joint Conference on Artificial Intelligence,* volume 22, pages 1312–1317, 2011.
20. R. Gan, J. Wang, J. Wang, G. Zeng, Z. Tu, and S. Li. Scalable k- nn graph construction for visual descriptors. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113, 2012.

References

21. Y.-m. Zhang, K. Huang, G. Geng, and C.-l. Liu. Fast knn graph construction with locality sensitive hashing. In *Joint European Con- ference on Machine Learning and Knowledge Discovery in Databases*, pages 660–674. Springer, 2013.
22. J. Wang , J. Wang , G. Zeng , Z. Tu , R. Gan , S. Li , Scalable K-NN graph construc- tion for visual descriptors, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, in: CVPR’2012, 2012, pp. 1106–1113 .

Acknowledgments

The research project is implemented in the framework of H.F.R.I call “3rd Call for H.F.R.I.” Research Projects to Support Faculty Members & Researchers” (H.F.R.I. Project Number: 25146).