Institute of Computing, University of Campinas

Large-Scale Indexing of High Dimensional Data via Nearest Neighbors Graphs

Javier Alvaro Vargas Muñoz

Advisor: Prof. Dr. Ricardo da Silva Torres
Co-Advisor: Prof. Dr. Zanoni Dias
March 12, 2020

Outline

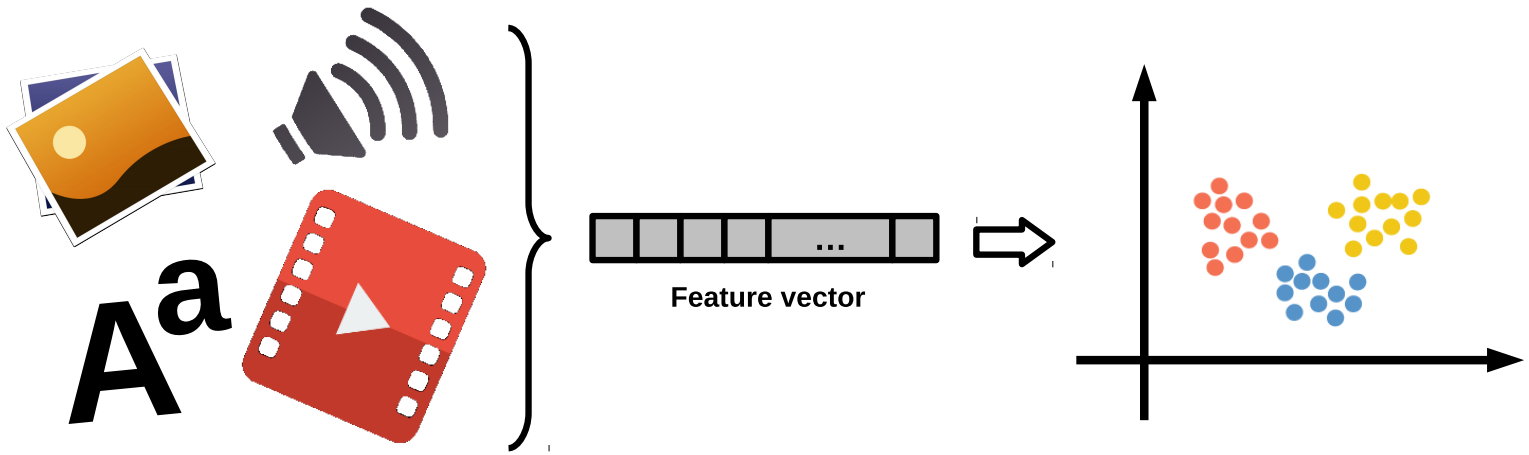
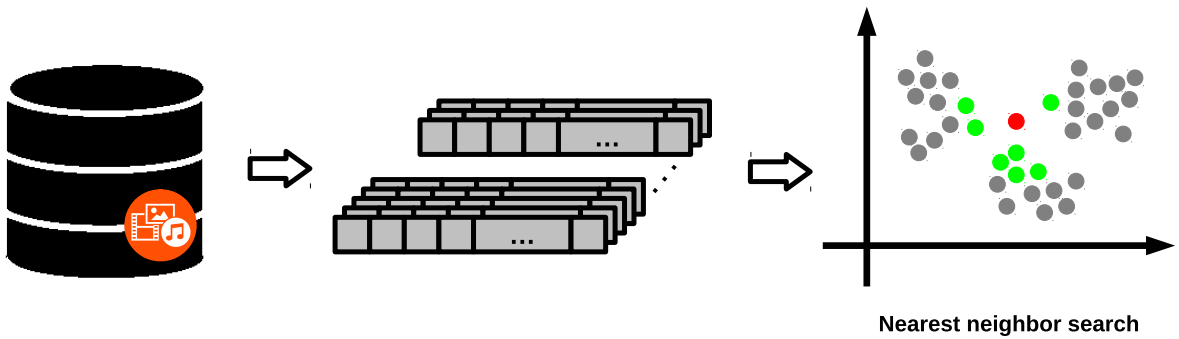


Figure 1: Vector representation for multimedia objects.

Introduction



Given a set $\mathcal{P} = \{x_1, x_2, x_3, .., x_N\}, \mathcal{P} \subset \mathbb{R}^D$, a distance function $d : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$, a query point $q \in \mathbb{R}^D$

K-nearest neighbors search problem

$$KNN(q, K, \mathcal{P}) = A, \text{ where } A \subseteq \mathcal{P} \wedge |A| = K \wedge \forall x \in A, y \in (\mathcal{P} - A), d(q, x) < d(q, y)$$

Nearest Neighbors Search

Computer Vision

- Find the best matches for local image features
- Global image feature matching for scene recognition
- Matching deformable shapes for object recognition
- Near duplicate detection

Machine Learning

- Nearest neighbors classifiers
- Clustering algorithms

Information Retrieval

- Multimedia indexing
- Similarity search

Exact Methods

Naive approach: linear search (brute force)

- **Impractical in large datasets**

Data structures

- KD-Tree, BallTree
- Search in logarithmic time in low dimensional data
- **Quickly converge to linear search as dimensionality increases**

Approximate Methods (ANNS)

Fast search, with small loss in precision

Four group of approaches on the literature:

- Two classic approaches
  - Tree partitioning structures
  - Hashing-based techniques
- **Nearest neighbors graphs**
- Quantization-based schemes

Tree Partitioning Structures

- **Index construction:** at each tree level, objects are split into subsets, based on some criteria
- **Search:** traversing from the root to the leaves, using the same criteria for split
- Examples:
  - FLANN library: Randomized KD-Trees, Hierarchical k-means tree, Hierarchical Clustering Tree
  - PCA-Tree, RP-Tree, Cover-Tree, VP-Tree

Hashing-based Techniques

- **Index construction:** map similar objects to the same positions in the hash tables (buckets)
- **Search:** find the bucket of the query object into, and uses the data objects in that bucket as the candidate set of the results
- Examples: Locality Sensitive Hashing (LSH), Multi-Probe LSH

Nearest Neighbors Graphs (NN graphs)

- **Index construction:** create a graph linking each object to the other most similar ones
- **Search:** start in some (random) vertex and, in a greedy maner, traverse the graph towards the closest points to the query, until some stopping criterion is reached
- Examples: FANNG, HNSW, SW-graph, KGraph
- Recents works and benchmarks have shown considerable gains over other approaches

Classic Search Example

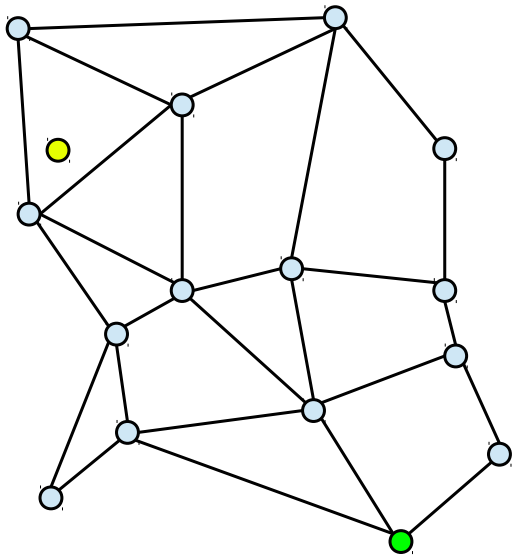


Figure 2: Example of search on a NN graph.

Classic Search Example

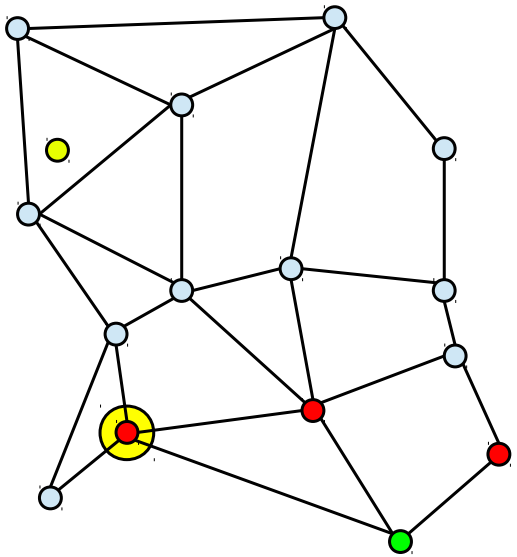


Figure 2: Example of search on a NN graph.

Classic Search Example

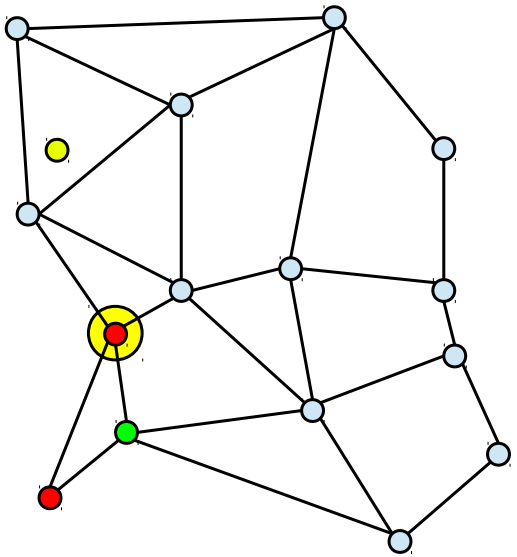


Figure 2: Example of search on a NN graph.

Classic Search Example

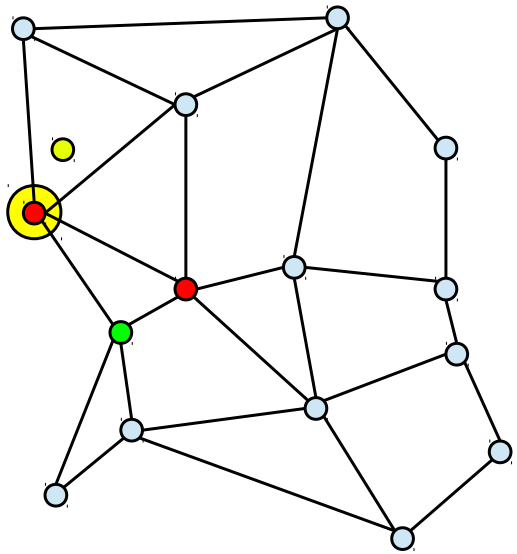


Figure 2: Example of search on a NN graph.

Classic Search Example

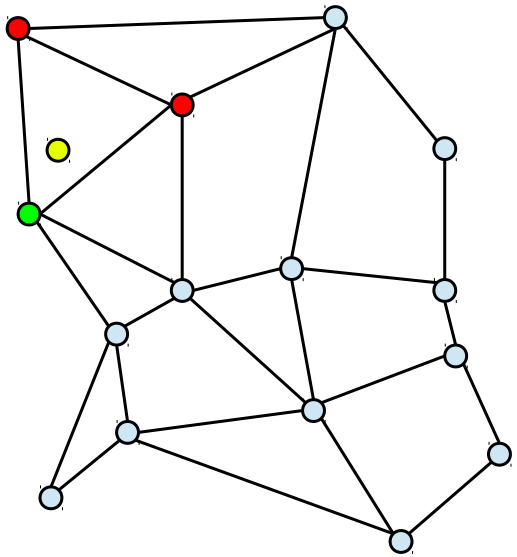


Figure 2: Example of search on a NN graph.

Classic Search Example

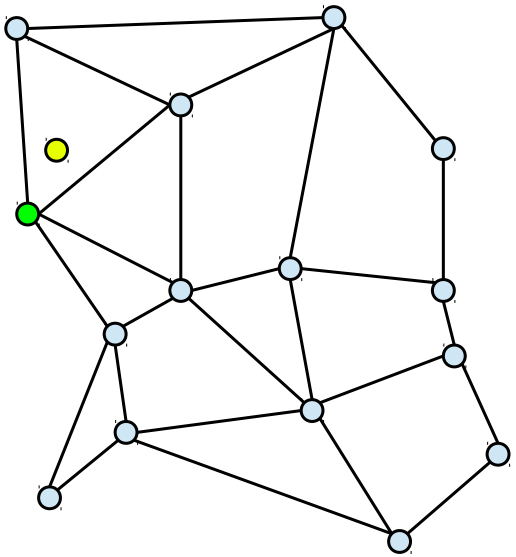


Figure 2: Example of search on a NN graph.

Quantization-based Techniques

- **Vector compression:** original vectors are compressed by employing multiple codebooks
- **Index construction:** generally, based on the idea of inverted indices
- **Search:** scanning the lists starting with those with keys nearest to the query
- Examples:
  - Compression: PQ, OPQ, LOPQ, AQ, CQ, TQ
  - Indexing: IVFADC, IMI, GNOIMI

Table 1: Analysis of principal pros e cons of the state-of-the-art.

	Advantage	**Disadvantage**
Tree	Low index construction time	Scalable to just millions and low search accuracy
Hashing	Scalable to billions	Lower search accuracy than other approaches
Graph	Best reported accuracy	Scalable to just millions
Quantization	Scalable to billions	Search accuracy affected by lossy compression

Research Topics

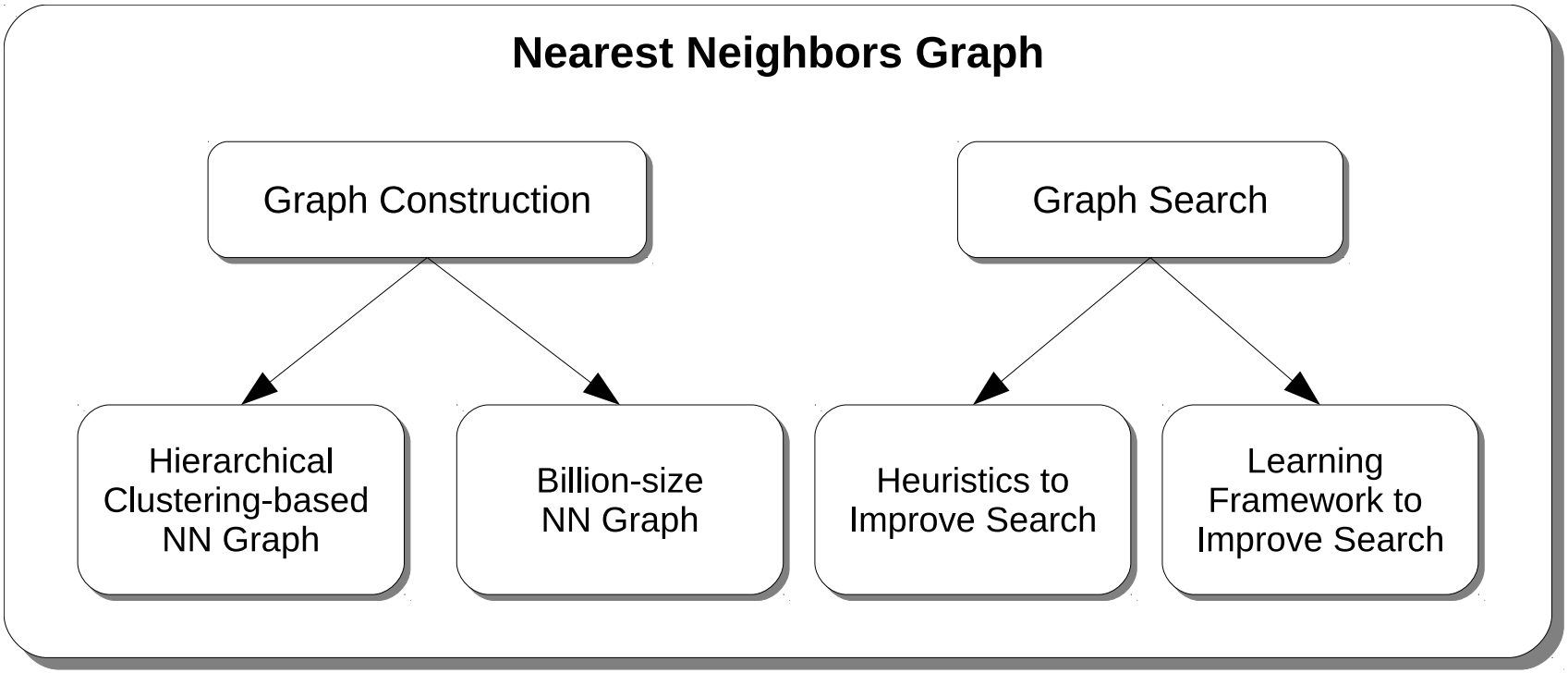


Figure 3: Topics covered in this work.

Hypothesis & Research Questions

Hypothesis 1

The use of multiple randomized clustering leads to the construction of NN graphs, which are faster to traverse at search time.

Q1.1 What is the best way to connect points inside clusters?

Q1.2 How can sub-graphs created by clusters be merged?

Q1.3 Which clustering algorithm should be employed?

Hypothesis & Research Questions

Hypothesis 2

The use of classical tree structures for indexing improves the NN search results on NN graphs, with no significant extra cost.

Q2.1 How the KD-Trees can be employed to improve the starting vertex selection on NN graph searches?

Q2.2 How the KD-Trees-like structures can be employed to avoid visit unnecessary vertices at graph navigation process?

Hypothesis & Research Questions

Hypothesis 3

The use of topological information of vertices (along with the distance to query) through a learning scheme leads to a better selection of next vertex (in each step of NN graph search), which fosters the earlier discovering of the true NN.

Q3.1 How to combine the topological properties with the distance?

Q3.2 Which learning technique can be used to find near optimal combinations of topological properties?

Q3.3 Which topological properties should be considered?

Q3.4 Which function should be optimized in the learning process?

Hypothesis & Research Questions

Hypothesis 4

Compressing vectors via quantization schemes and the adoption of suitable pruning strategies at construction time allow the construction of NN graphs on billion-size datasets with a reasonable memory consumption and time construction.

Q4.1 Does the compression of original vectors affects to the accuracy of search on NN graphs?

Q4.2 Which heuristics can be used for pruning edges at graph construction?

Q4.3 Can we obtain search performance improvements by exploiting the topological properties of vertices for pruning edges at graph construction?

Hypothesis & Research Questions

Hypothesis 4

Compressing vectors via quantization schemes and the adoption of suitable pruning strategies at construction time allow the construction of NN graphs on billion-size datasets with a reasonable memory consumption and time construction.

Q4.4 Does our NN graph technique still maintain its top search performance at NN search when compared to the state-of-the-art schemes for billion-scale ANNS?

Q4.5 How much resources need the proposed NN graph technique in comparison with state-of-the-art?

Contributions

1. A novel approach for efficient construction of NN graphs, with comparable search performance than state-of-the-art techniques for ANNS
2. Two novel heuristics to improve search on NN graphs using classical KD-Trees as auxiliary data structures: one for initial vertex selection, and the other for avoiding exhaustive exploration of vertices' neighbors
3. A Genetic Programming (GP) framework that aims to discover a near-optimal combination of local topological features of vertices along with the classical *distance-to-the-query*, that improves the criterion for selection of the next vertex to be explored in the search algorithm

Contributions

4. The first reported memory-aware technique for creation of sparse NN graphs on billion-size datasets
5. The extension of the GP framework above applied to the selection of vertices' neighbors at NN graph construction stage, in scenarios with restricted degree of vertices (as in the billion-size datasets)

Hierarchical Clustering-based Nearest Neighbors Graphs (HCNNG)

HCNNG Framework

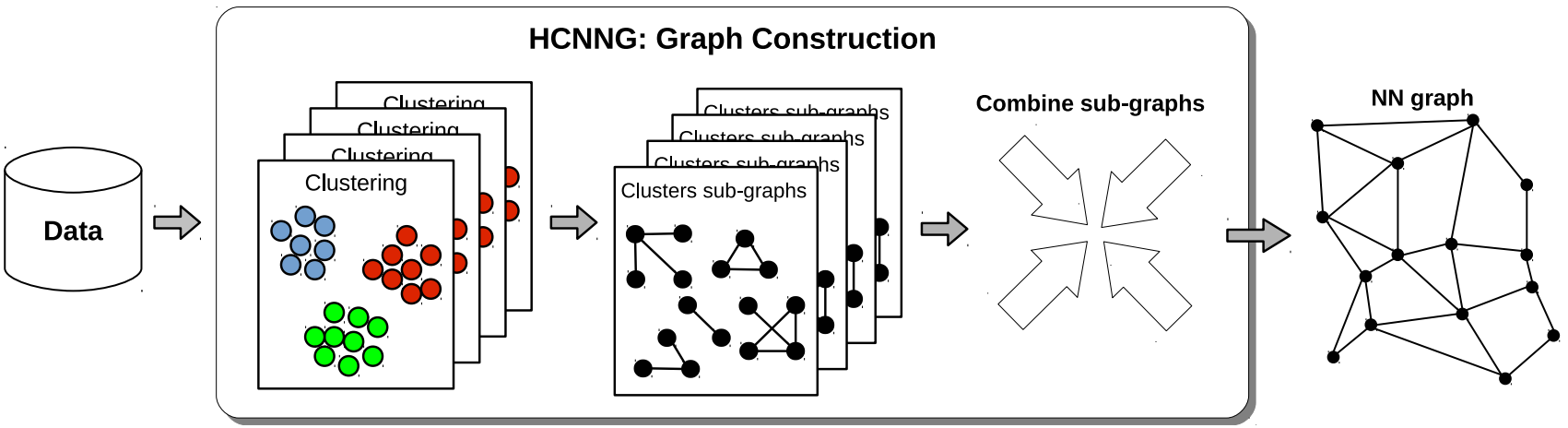


Figure 4: Overview of graph construction algorithm.

HCNNG Framework

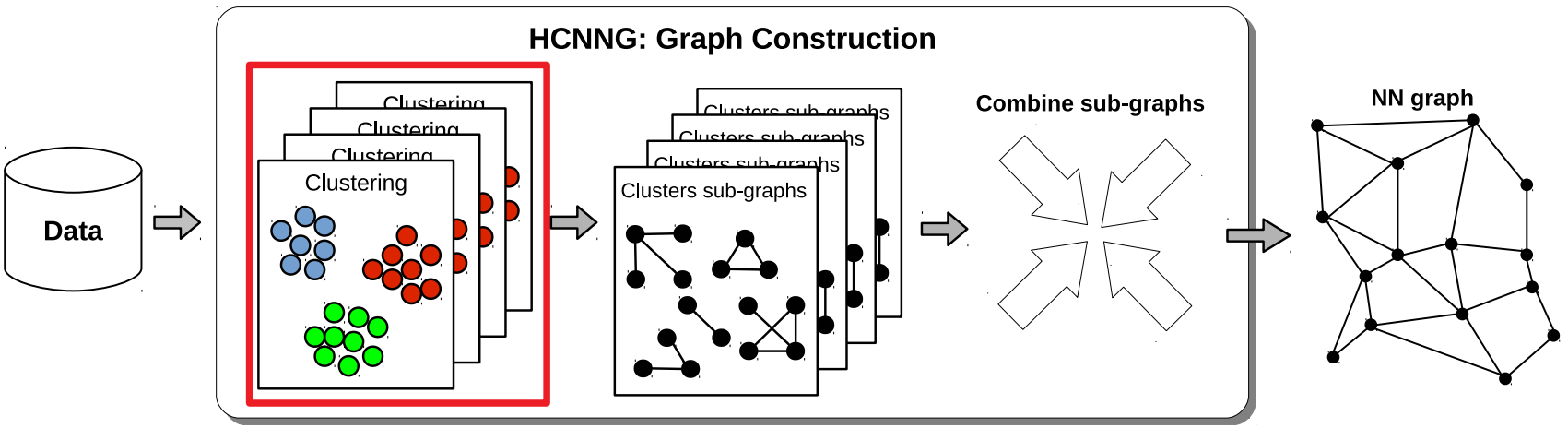


Figure 5: Overview of graph construction algorithm.

- Many techniques in the literature for clustering
- It is extremely important the time execution, since we are working with million or billion of vectors
- Clusters quality is not very important
- **Hierarchical clustering**, in its simplest version, present a low time complexity

Given a set of points $\mathcal{P} = \{x_1, x_2, ..., x_N\}$ and minimum clusters size n

$$HC(\mathcal{P}) = \begin{cases} \mathcal{P}, & |\mathcal{P}| < n \\ HC(\gamma(\mathcal{P}, r, s)) \cup HC(\gamma(\mathcal{P}, s, r)), & else \end{cases} \quad (1)$$

being r and s ($r, s \in \mathcal{P} \wedge r \neq s$) two randomly selected points in each recursive call, and $\gamma(\mathcal{P}, x, y)$ a function that returns the set of points that are closer to x than to y

Expected time complexity: $O(ND \log(N/n))$

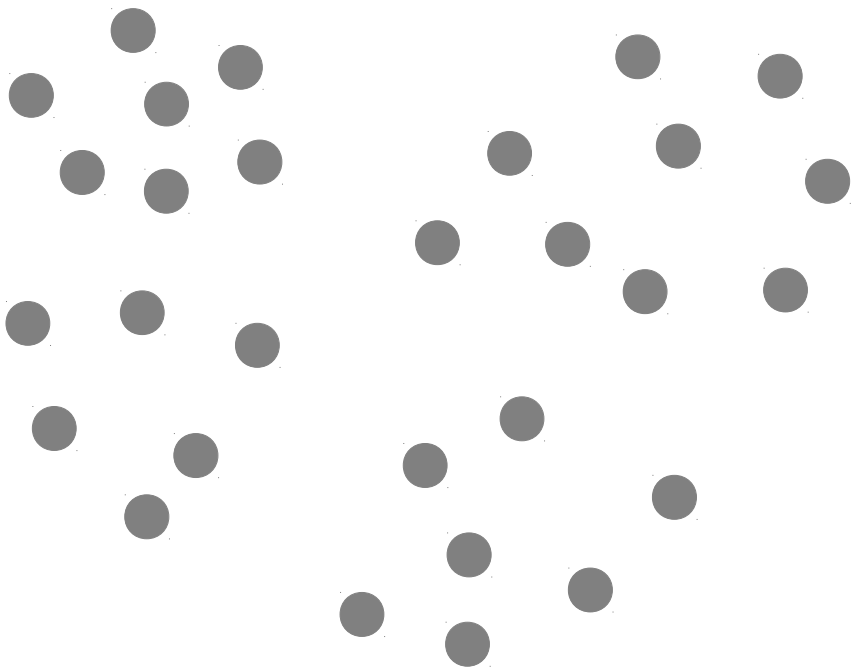


Figure 6: Example of hierarchical clustering.

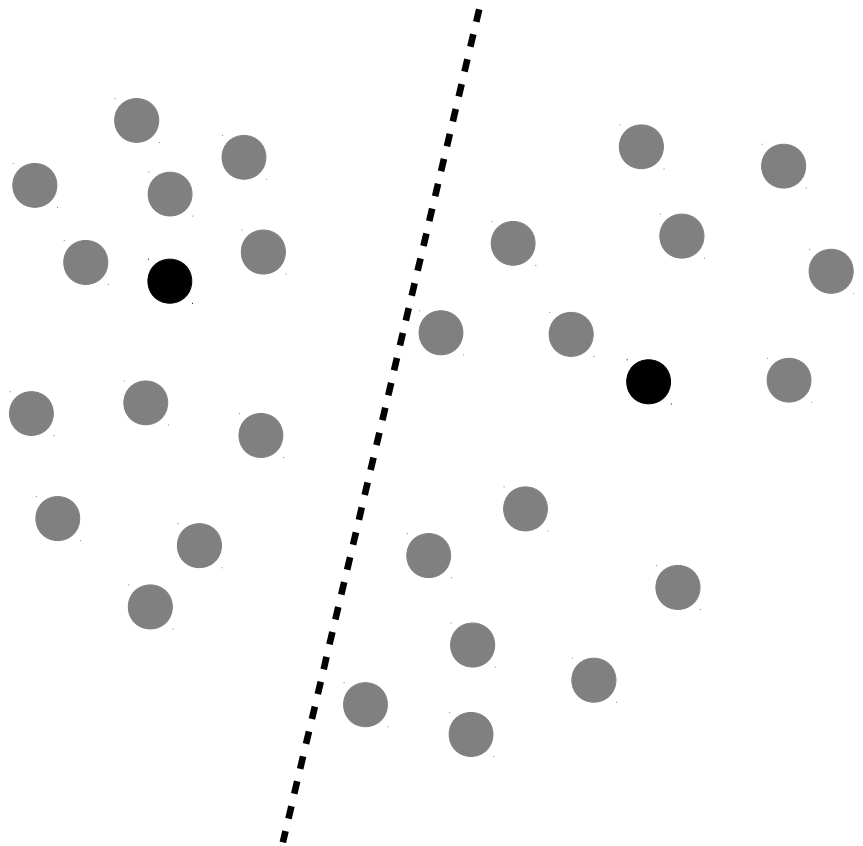


Figure 6: Example of hierarchical clustering.

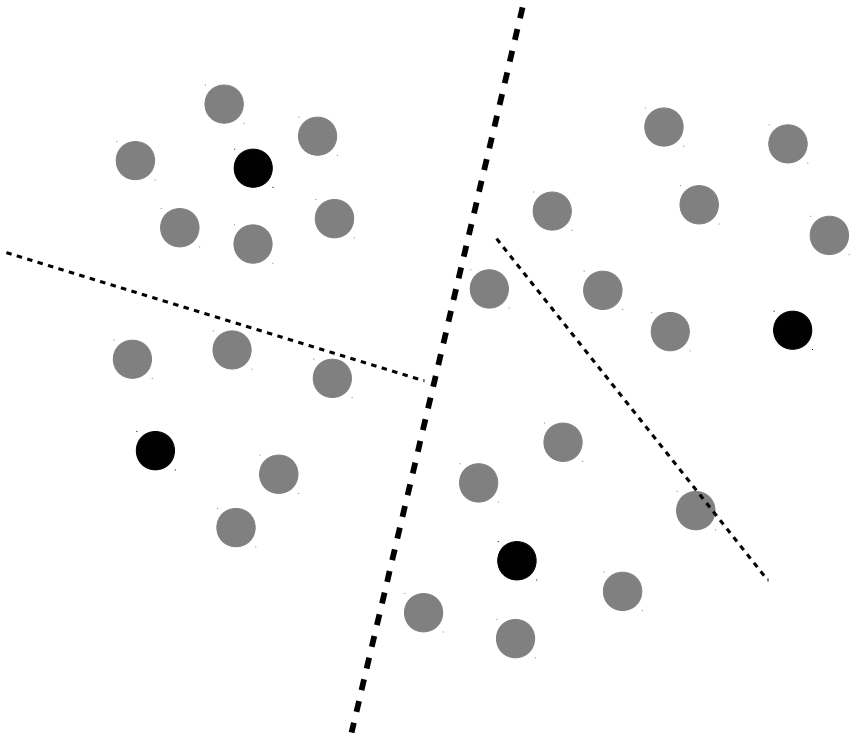


Figure 6: Example of hierarchical clustering.

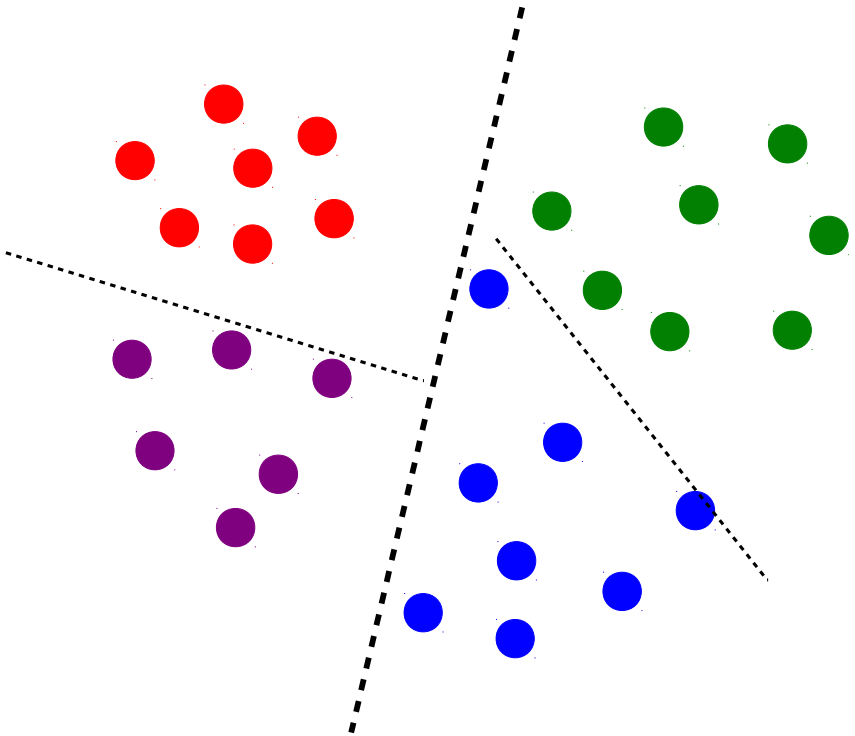


Figure 6: Example of hierarchical clustering.

HCNNG Framework

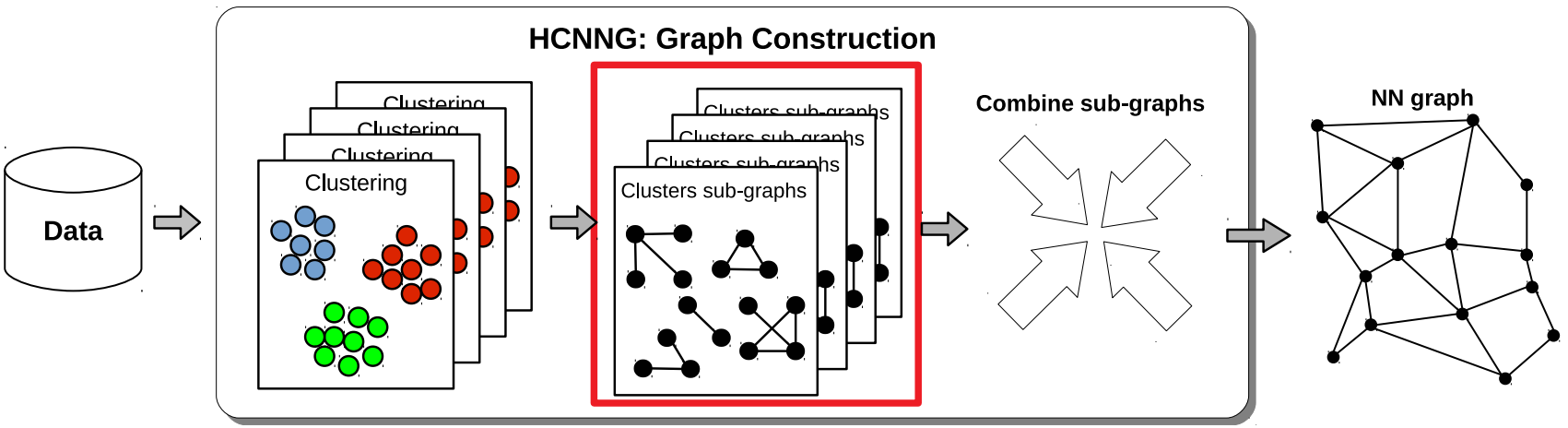
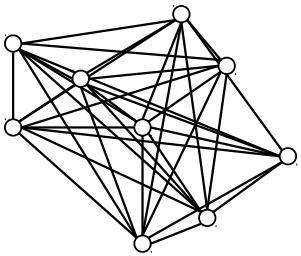


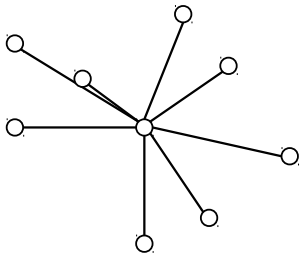
Figure 7: Overview of graph construction algorithm.

Sub-graphs

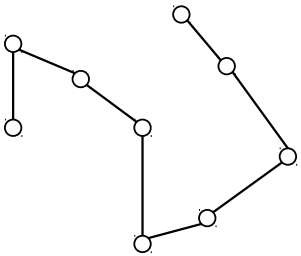
Complete graph



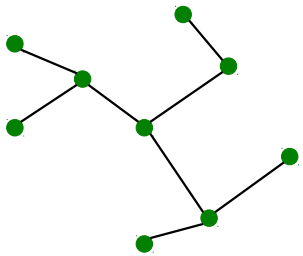
Star



Hamiltonian path



MST3



	Max degree	Max path
Complete	O(n)	O(1)
Star	O(n)	O(1)
Hamiltonian	O(1)	O(n)
MST3	**O(1)**	**log(n)**

Figure 8: Graph structures employed to connect points in clusters.

Sub-graphs

- **Complete graph:** Generates graphs with high degrees for all vertices
- **Stars:** Generate graphs with some vertices presenting high degrees (hubs)
- **Hamiltonian path:** Generate graphs with low degree for all vertices
- **MST3:** Generate graphs with slightly higher degrees than hamiltonian paths

Experimentally, the **MST3** presented best search results and low vertices degrees.

HCNNG Framework

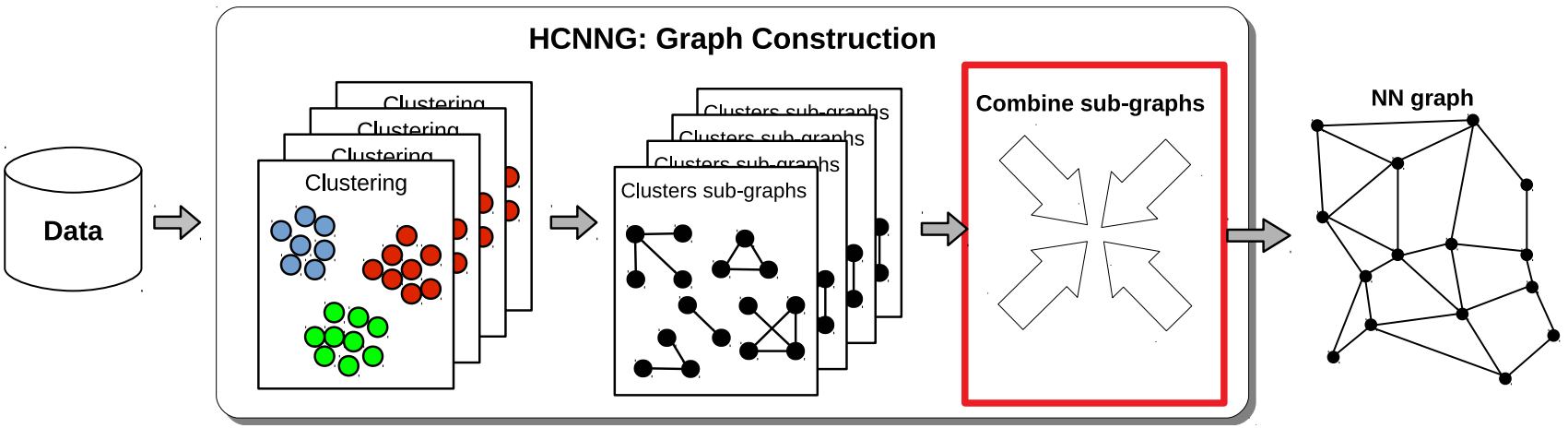


Figure 9: Overview of graph construction algorithm.

Merging Sub-graphs

Given a set of cluster $C = \{c_1, c_2, \ldots, c_S\}$ after multiple hierarchical clustering. Let be $G_i = (V_i, E_i)$ a sub-graph, where $V_i = c_i$ and $E_i = MST3(c_i)$.

Final global graph

The final graph G' is defined as $G' = (V', E')$, such that $V' = \cup_{i=1}^{n} V_i$ and $E' = \cup_{i=1}^{n} E_i$.

Merging Sub-graphs

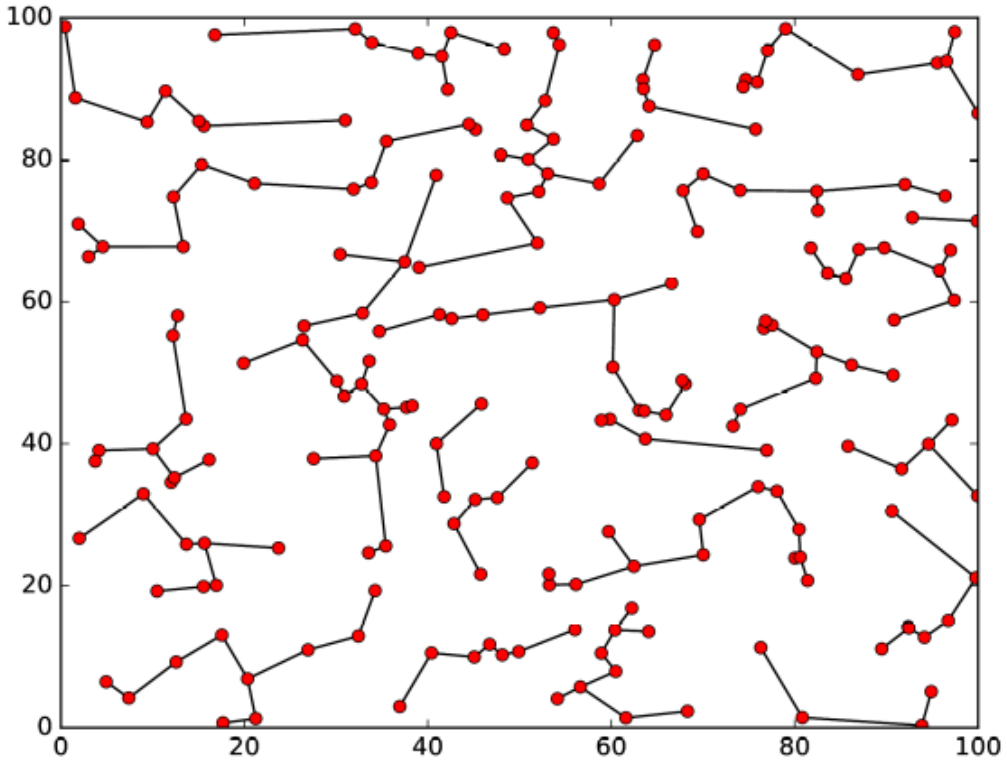


Figure 10: Sub-graphs after an hierarchical clustering execution.

Merging Sub-graphs

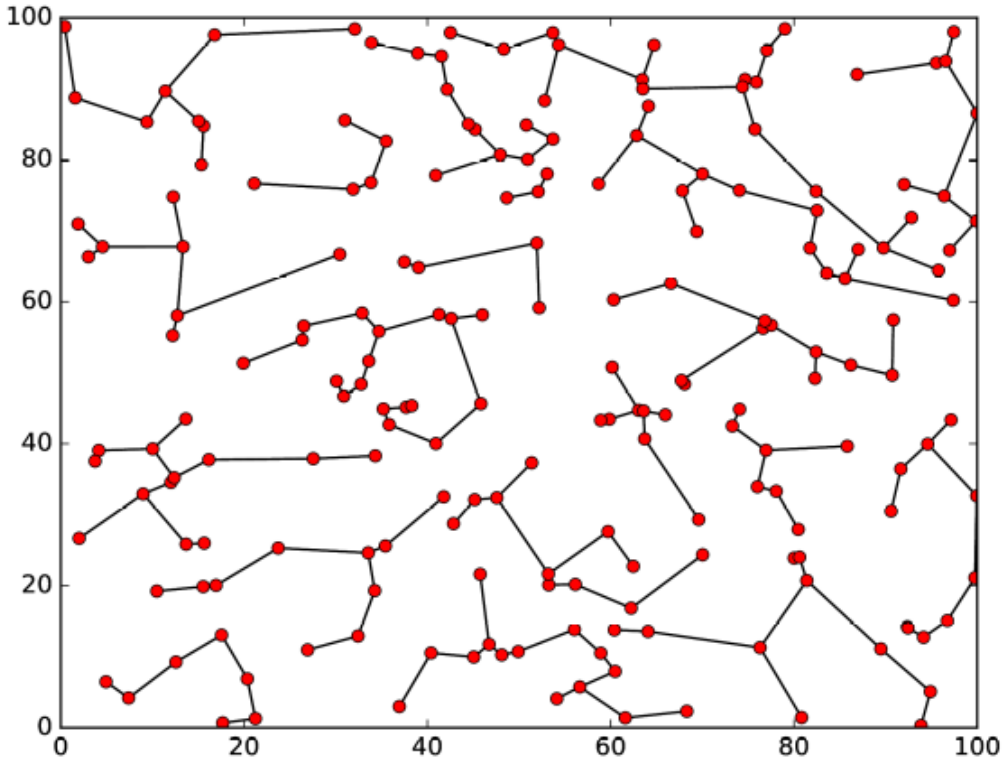


Figure 10: Sub-graphs after an hierarchical clustering execution.

Merging Sub-graphs

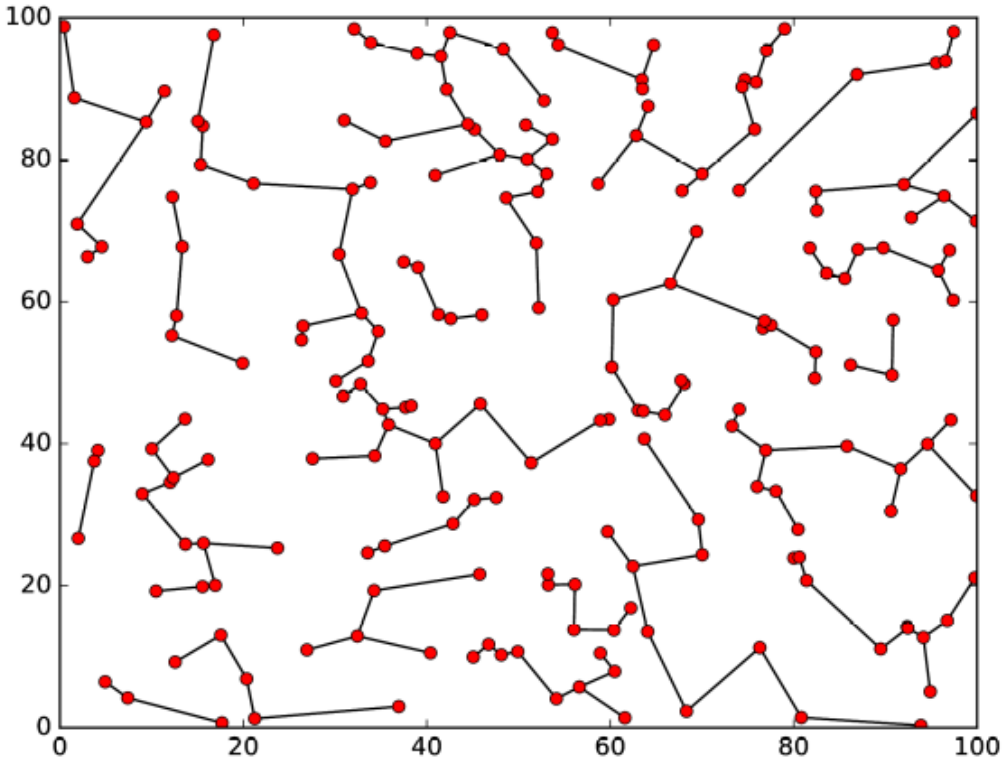


Figure 10: Sub-graphs after an hierarchical clustering execution.

Merging Sub-graphs

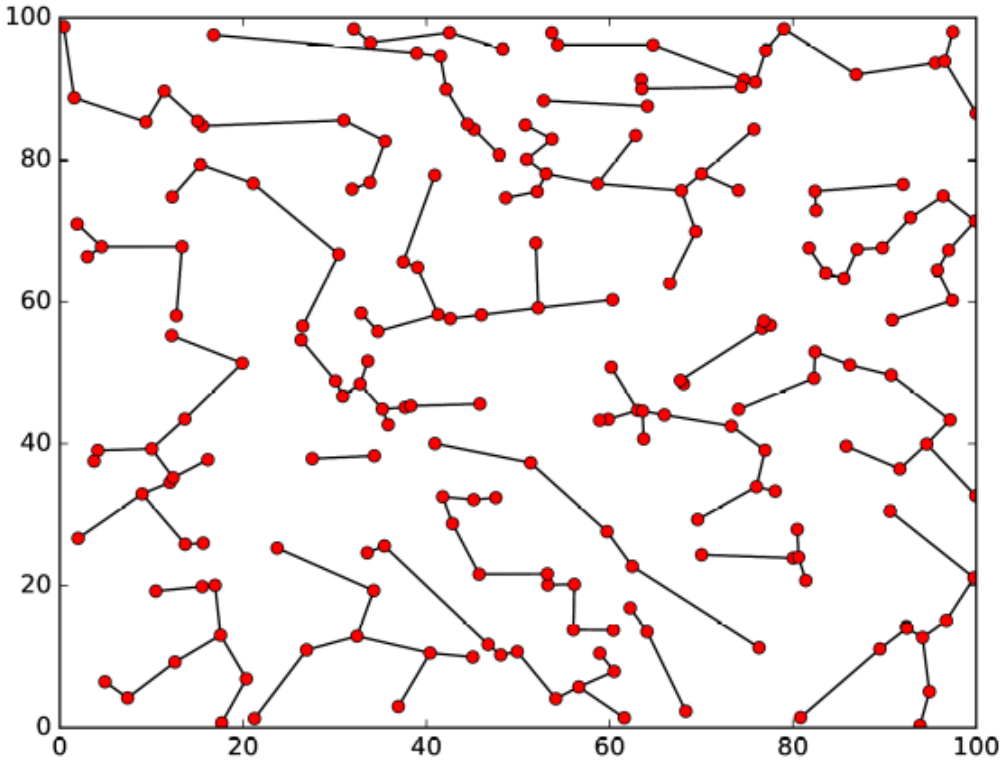


Figure 10: Sub-graphs after an hierarchical clustering execution.

Merging Sub-graphs

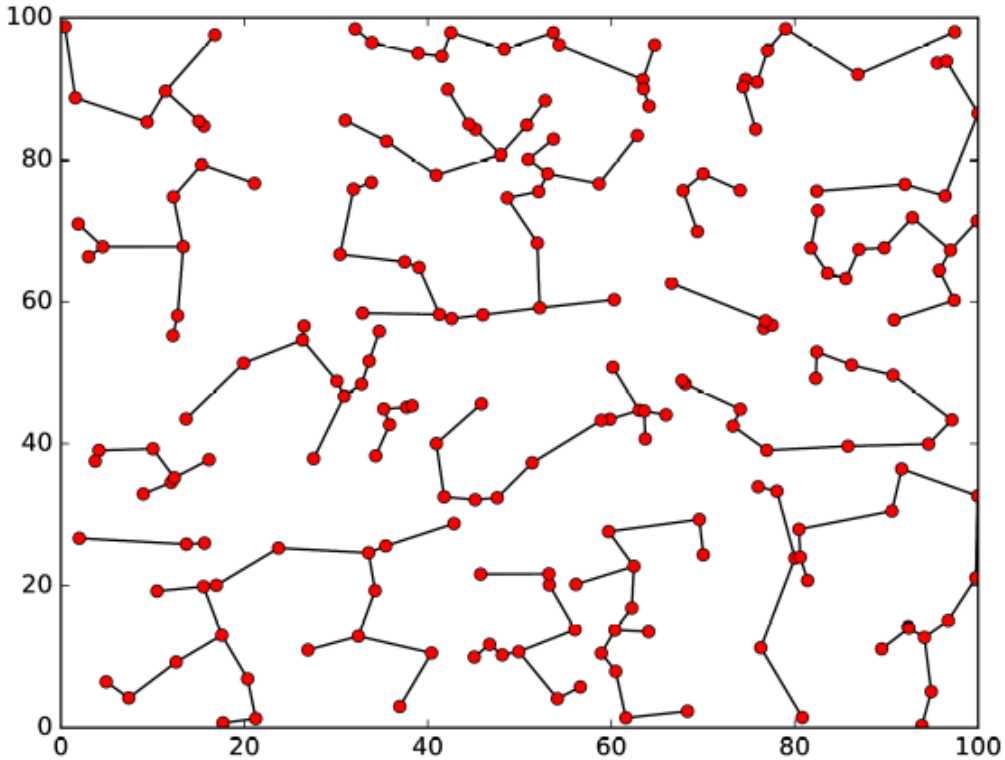


Figure 10: Sub-graphs after an hierarchical clustering execution.

Merging Sub-graphs

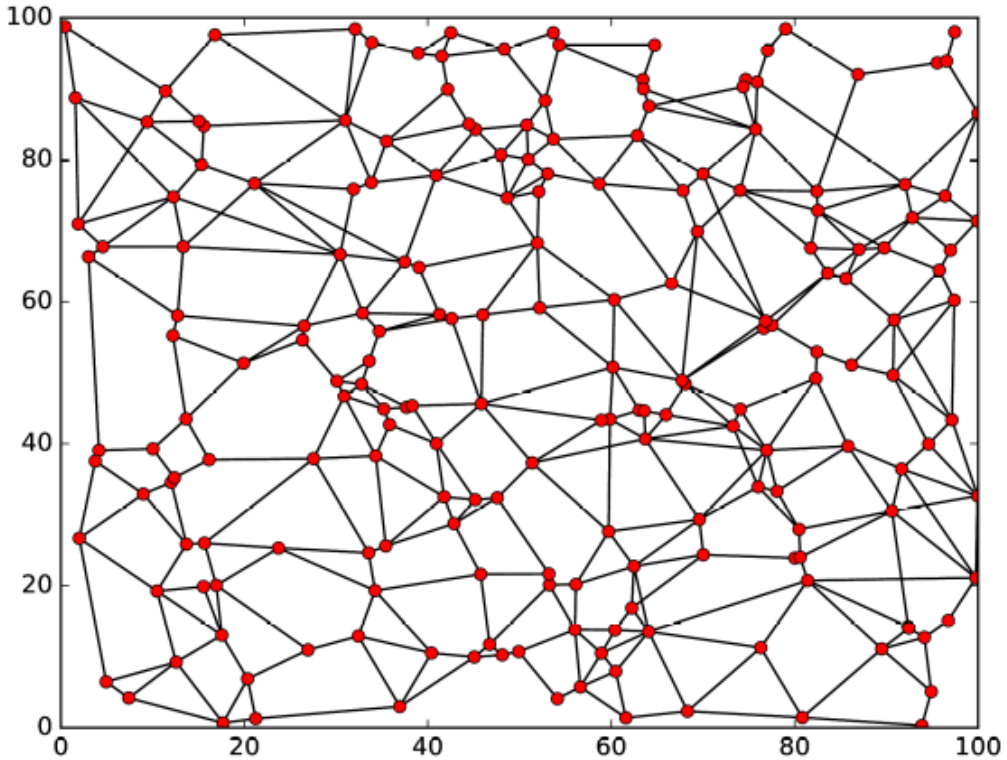


Figure 11: Combination of all sub-graphs after 15 executions.

HCNNG Framework

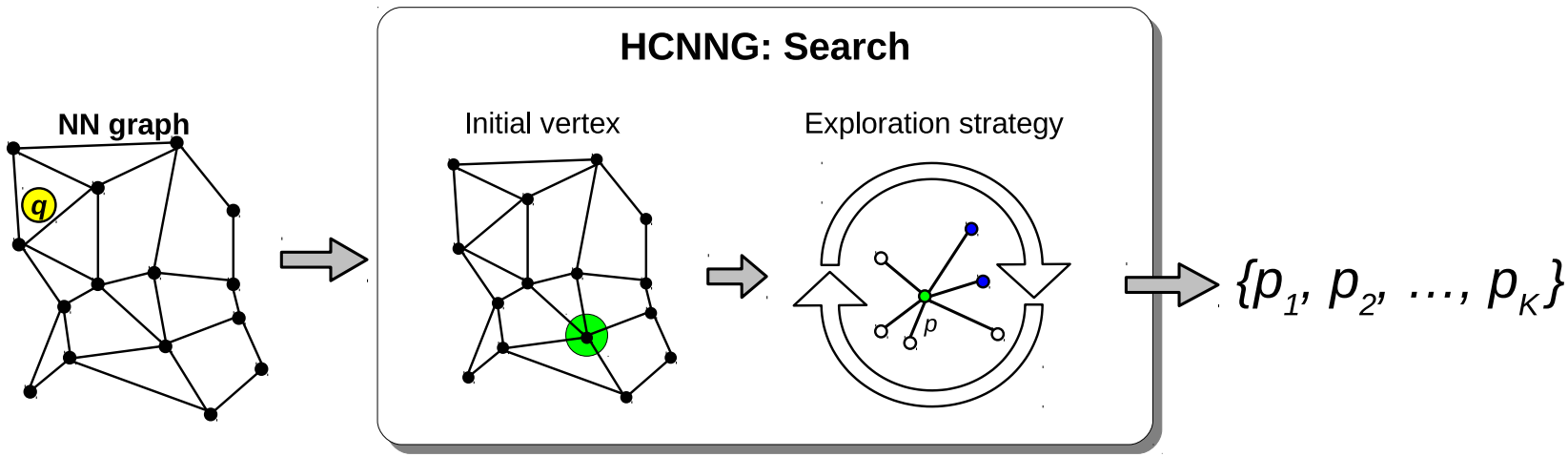


Figure 12: Overview of search algorithm.

Initial Vertex Selection

- We based our initial vertex selection on KD-Trees
- KD-Trees are cheaper in time construction and memory
- We don't perform exhaustive search on KD-Trees
- Multiple KD-Trees improve the selection

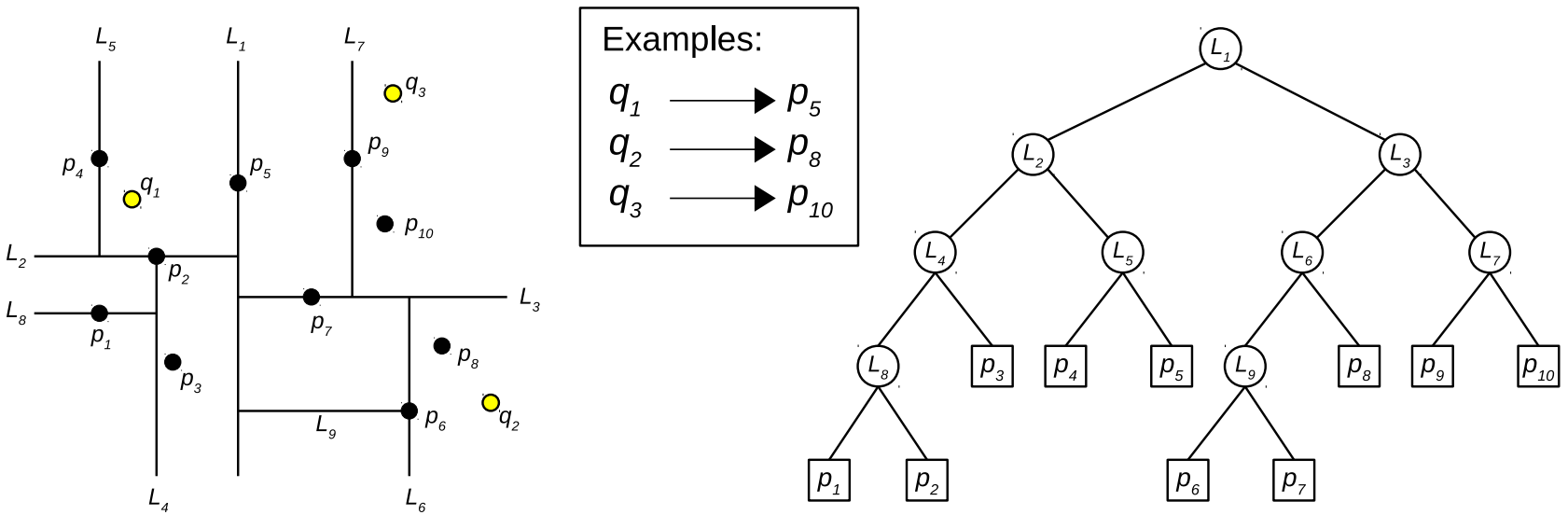


Figure 13: Example of KD-Tree-based initial vertex selection.

Exploration Strategy

- In each step moves towards some vertex in the same quadrant as the query

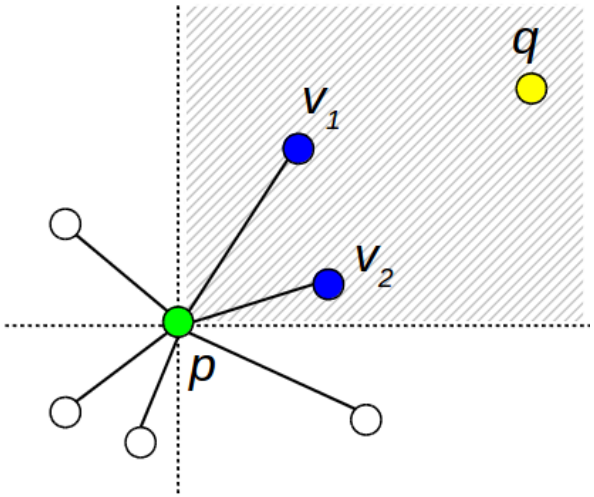


Figure 14: Search *guided* by quandrants.

Guided Search Example

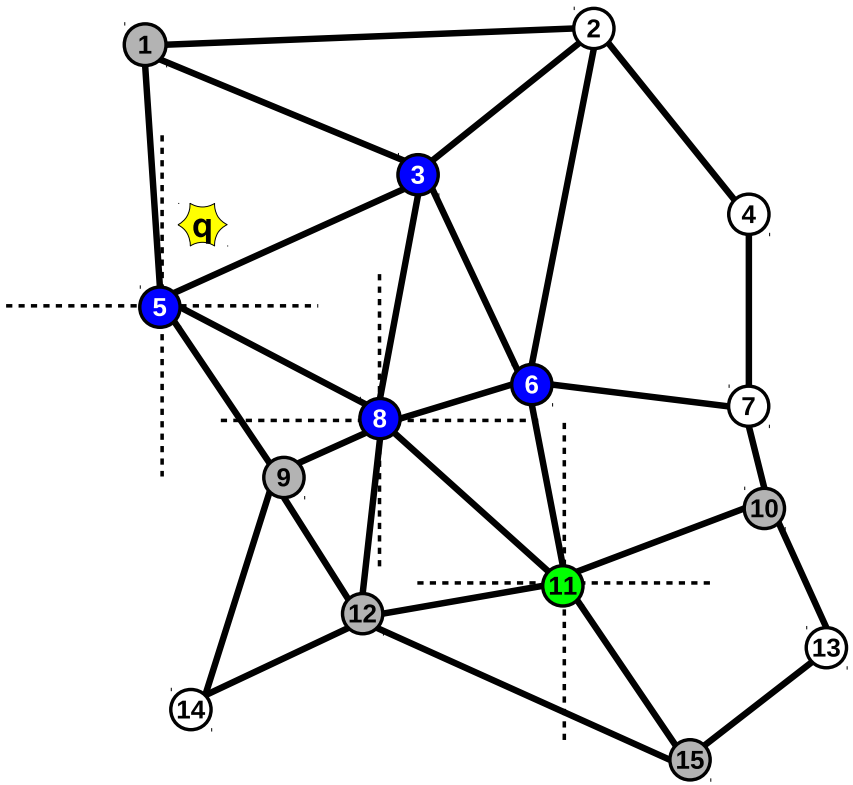


Figure 15: Example of *guided* search vs classical search.

- Computing vertices on quadrants at search is very costly
- Quadrants can be pre-computed and stored in KD-Tree-like structures offline
- They are computed for each vertex and their neighborhood
- These KD-Trees are cheap to traverse at search time

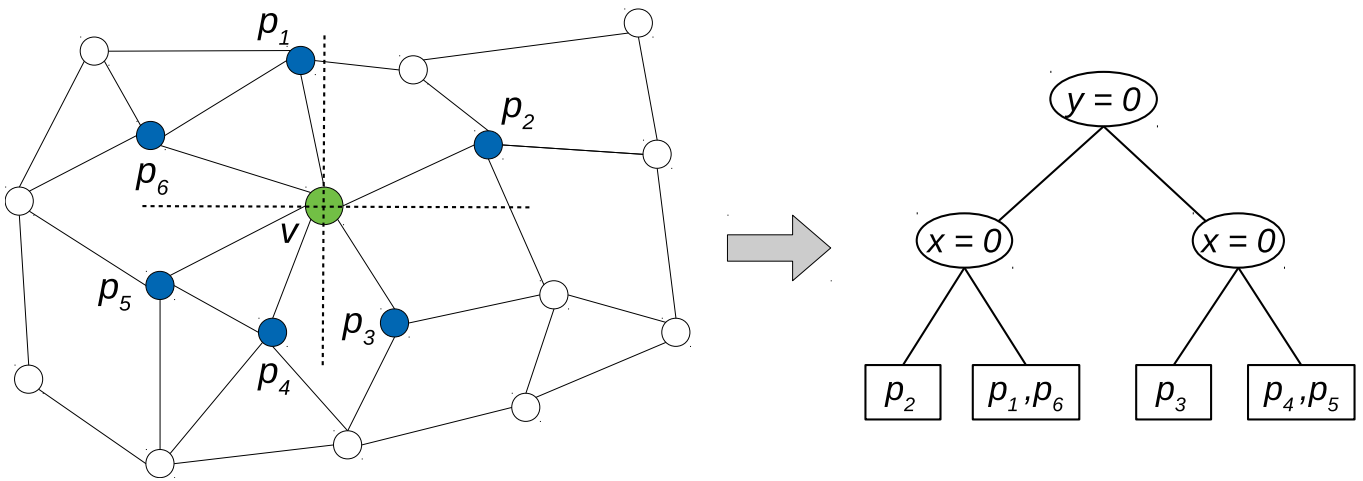


Figure 16: Example of local KD-Tree for neighborhood of vertex *v*.

HCNNG vs State-of-the-art

	HCNNG	HNSW	FANNG	SW-graph	KGraph
	Graph construction				
# graphs	**single**	multiple	single	single	single
strategy	**divide and conquer**	incremental construction	incremental construction	incremental construction	optimization of random graph
generic space	**yes**	yes	no	yes	yes
	Graph search				
initial vertex	**KD-Tree based**	random	centroid	multiple random	random
strategy	**guided + backtracking**	multiple level + backtracking	backtracking	backtracking	backtracking

Validation

- Euclidean distance
- BIGANN datasets for ANNS (visual features):
  - 1 million of SIFT features vectors (128 dimensions) for index construction, and 10K queries to evaluate search performance
  - 1 million of GIST features vectors (960 dimensions) for index construction and 1K queries to evaluate search performance
- GloVe (textual features)
  - 1 million of GloVe features vectors (100 dimensions) for index construction, and 10K queries to evaluate search performance

Validation

- Baselines
  - Fast Library for Approximate Nearest Neighbors (FLANN, Muja and Lowe, 2014), a well-known and widely used
  - Fast Approximate Nearest Neighbour Graphs (FANNG, Harwood and Drummond, 2016): Using our own implementation
  - Small world graphs (SW, Malkov et al., 2013) and Hierarchical Navigable Small World (HNSW, Malkov and Yashunin, 2017): Using the implementation found in Non-Metric Space Library (NMSLIB)
  - KGraph (Dong et al., 2011): Using the implementation provided by authors

Parameter Tuning: Graph Construction

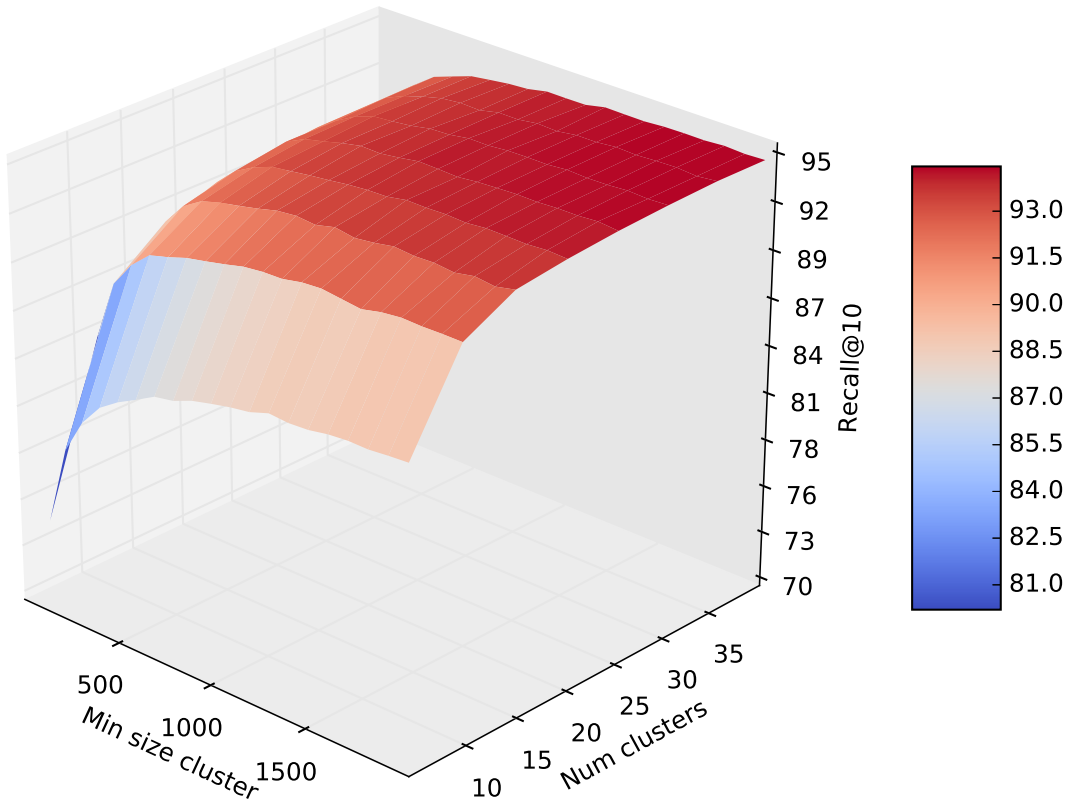


Figure 17: Clusters size × clustering executions (100K 20-D random vectors).

Parameter Tuning: Search Algorithm

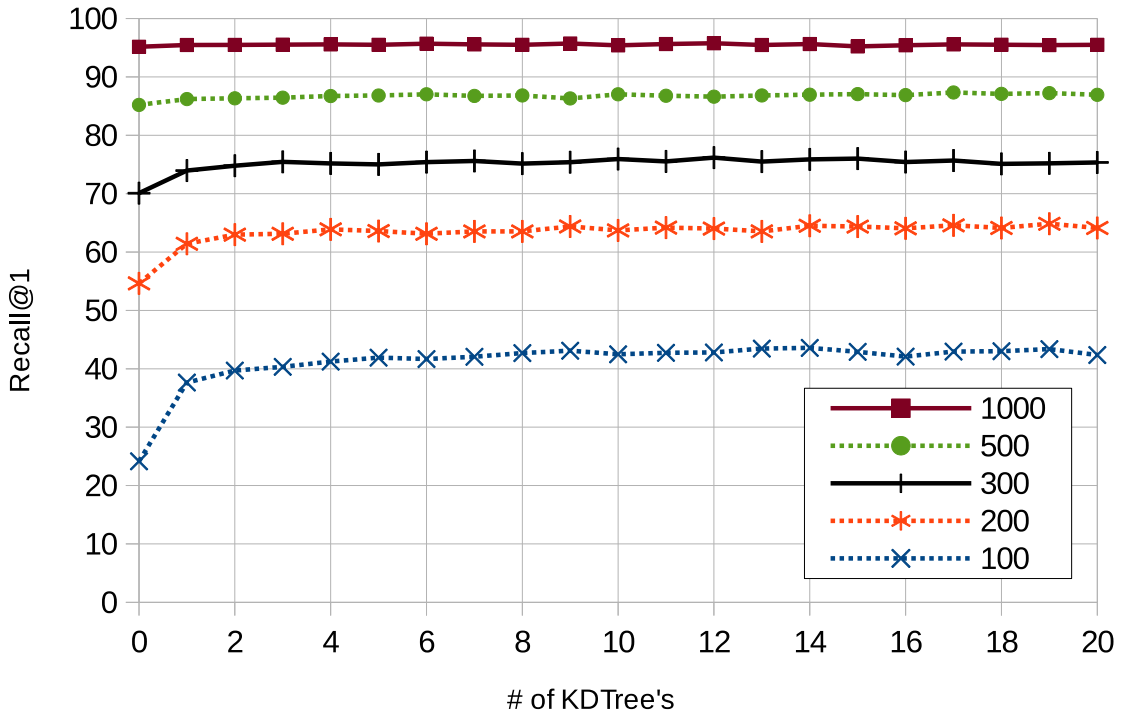


Figure 18: Number of KD-Trees for initial vertex selection (SIFT dataset).

Configurations of Search Algorithm

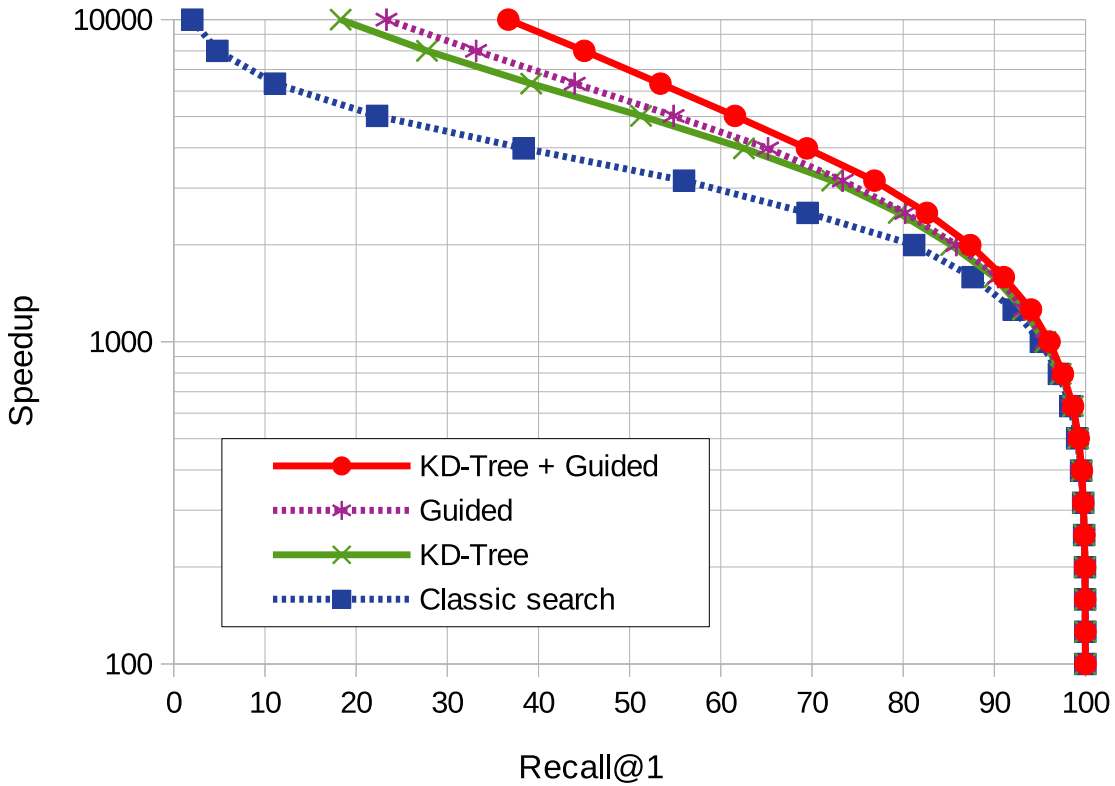


Figure 19: Different configurations of search on NN graph (SIFT dataset).

Results on SIFT

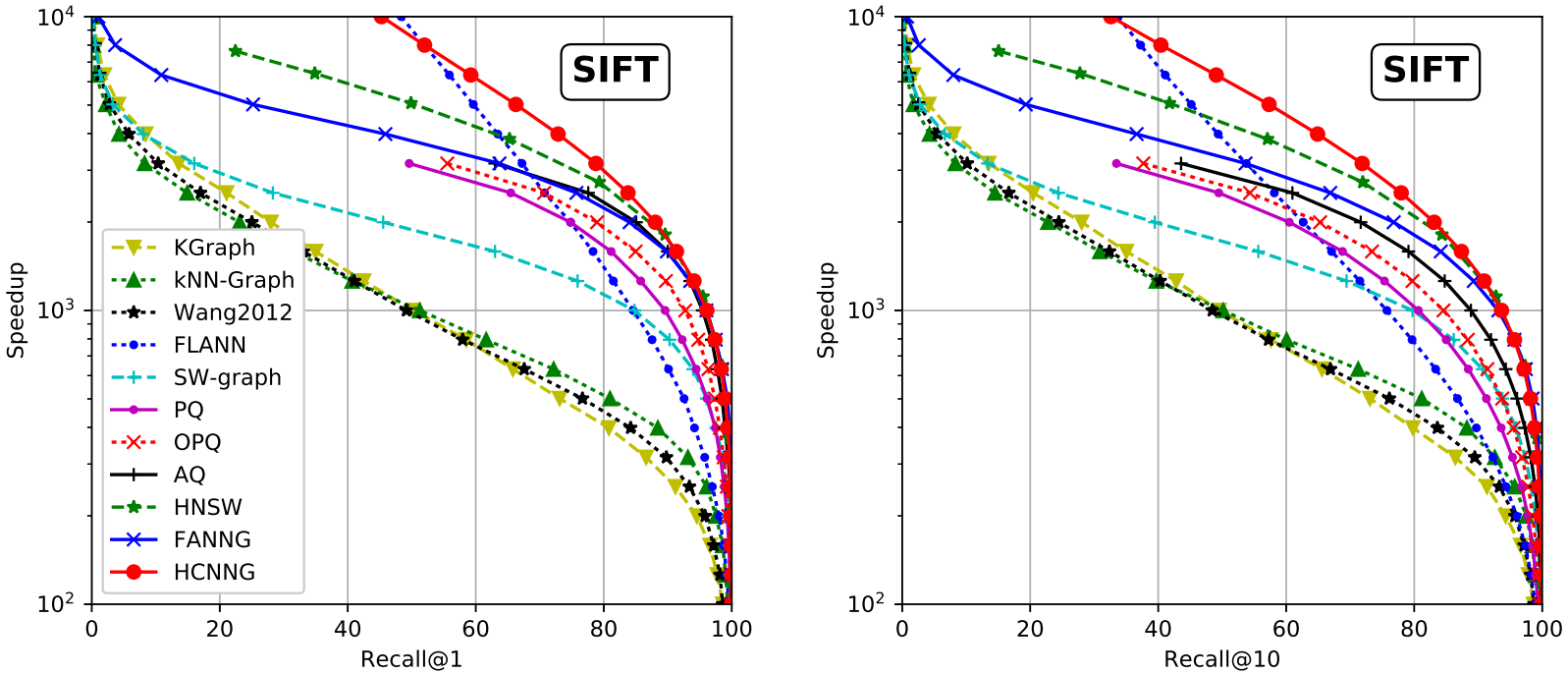


Figure 20: 1-NN and 10-NN search performance on SIFT dataset.

Results on GIST

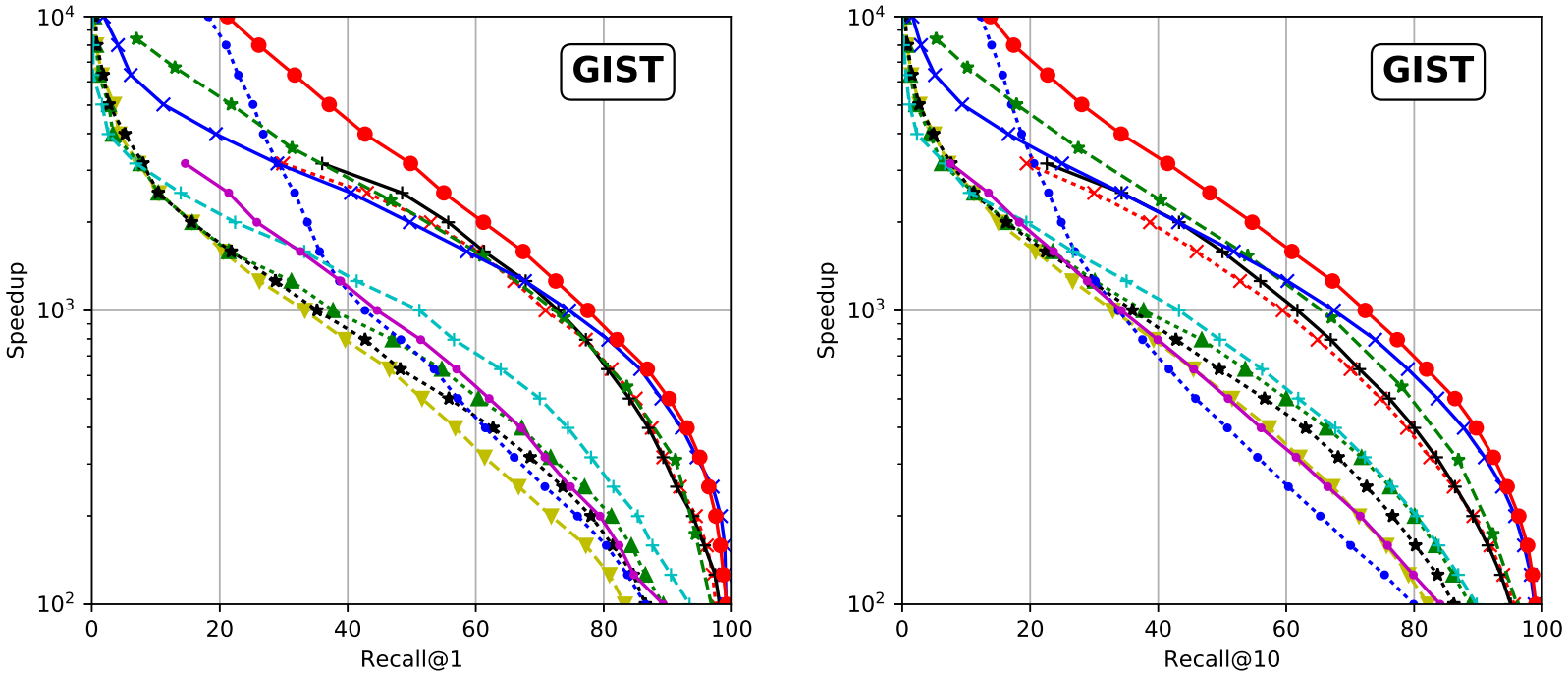


Figure 21: 1-NN and 10-NN search performance on GIST dataset.

Results on GloVe

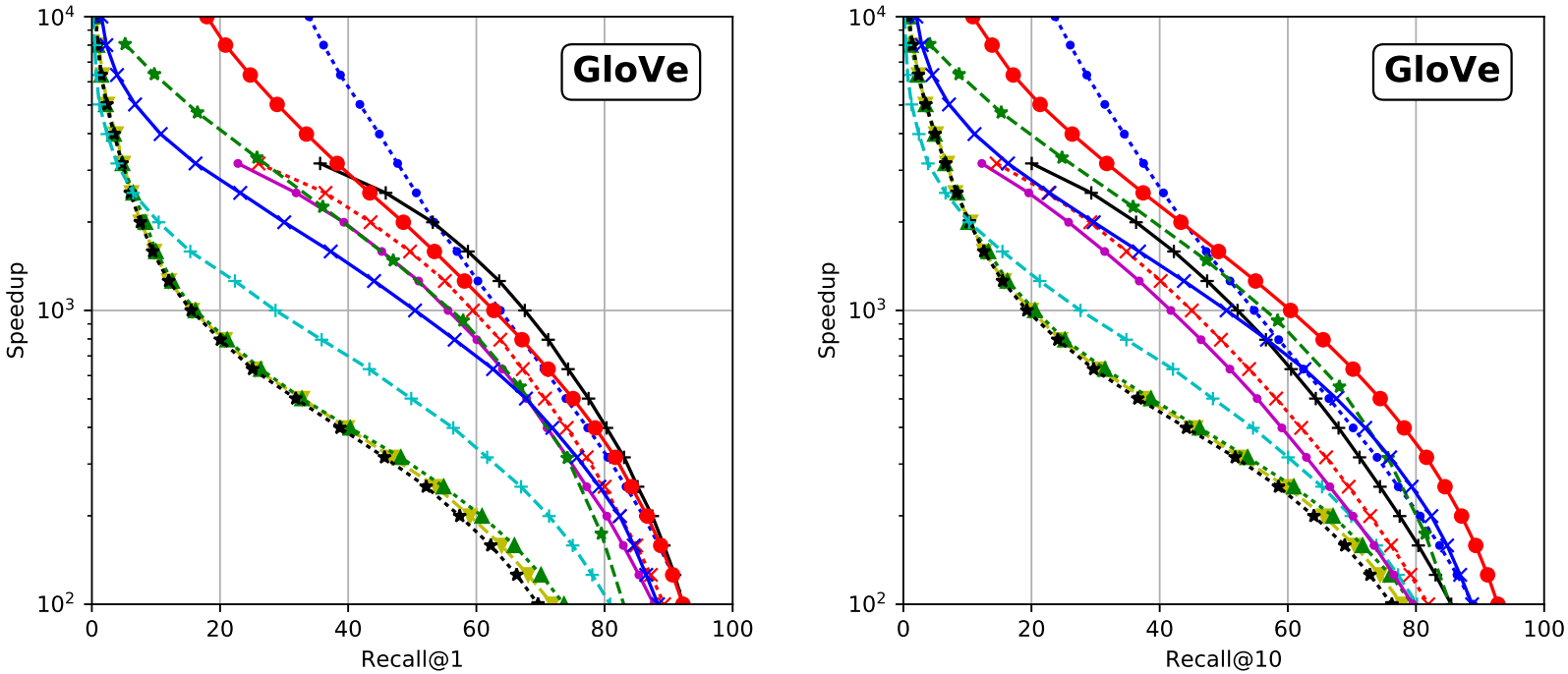


Figure 22: 1-NN and 10-NN search performance on GloVe dataset.

Learning Framework for Search on Nearest Neighbors Graphs

Search Example

- $u \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow v$: 12 vertices explored
- $u \rightarrow y_1 \rightarrow y_2 \rightarrow v$: 8 vertices explored

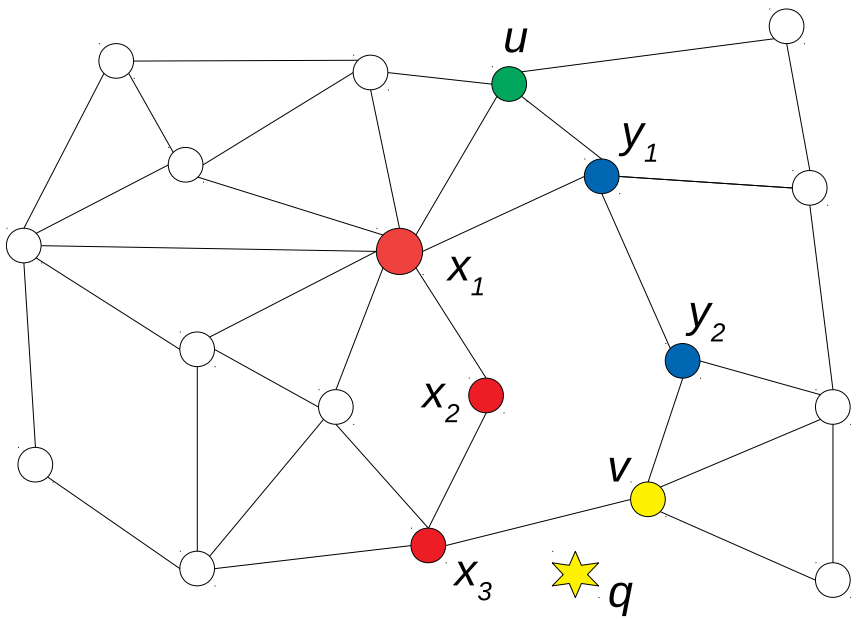


Figure 23: Two different paths to find the nearest neighbor (v) of q.

GP-based Learning Framework

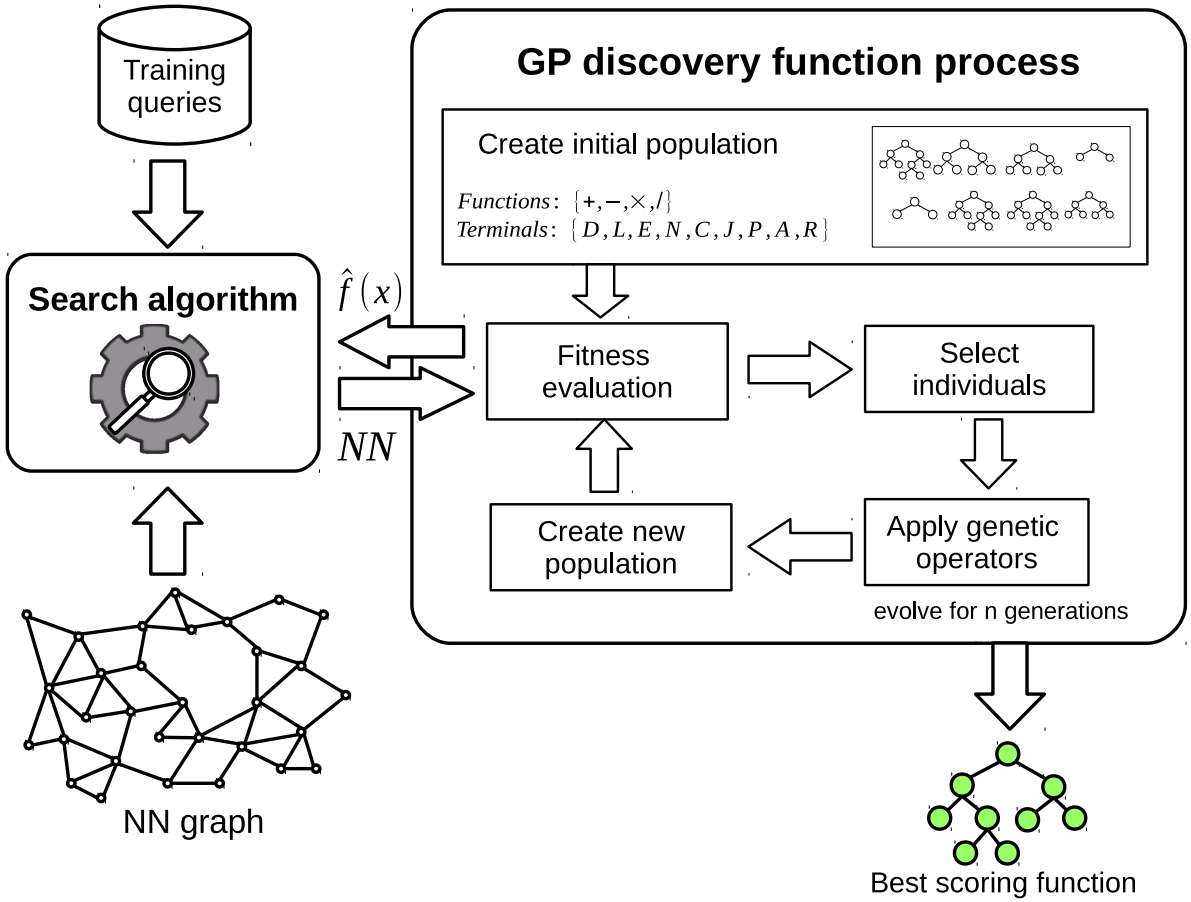


Figure 24: Leaning framework to exploit topological properties.

GP-based Learning Process

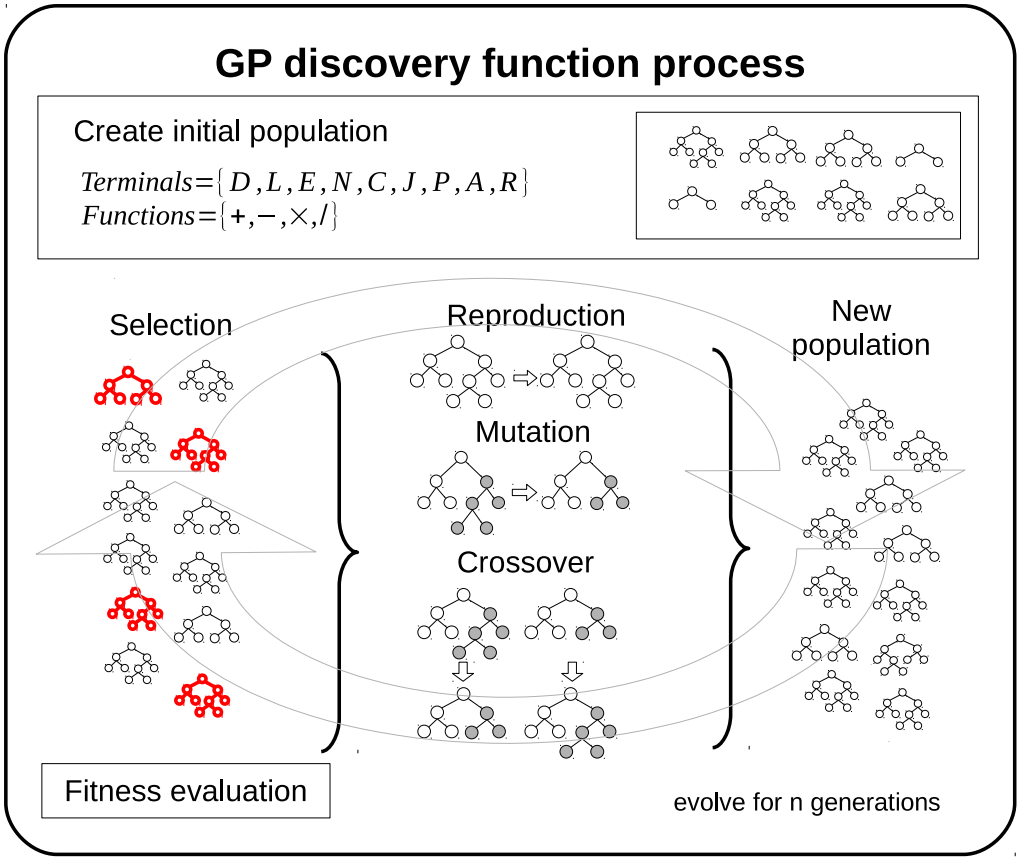


Figure 25: Genetic programming near-optimal function discovery process.

GP Individual

- **Functions:** Internal nodes. The set of functions employed to combine the inputs (commonly mathematical operators)
- **Terminals:** Leaf Nodes. The set of values to be combined

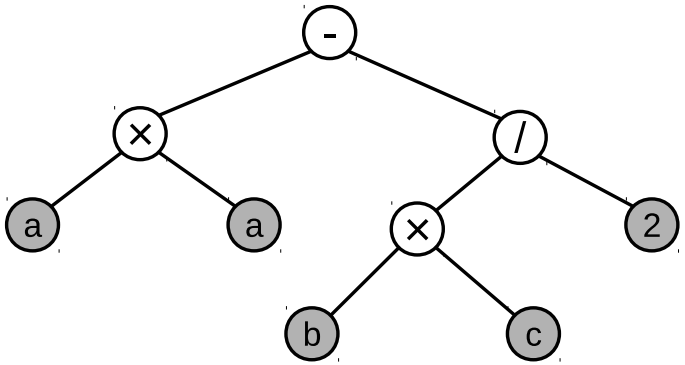


Figure 26: Example of classic tree representation for GP individuals.

$$f(a,b,c) = a \times a - ((b \times c)/2)$$

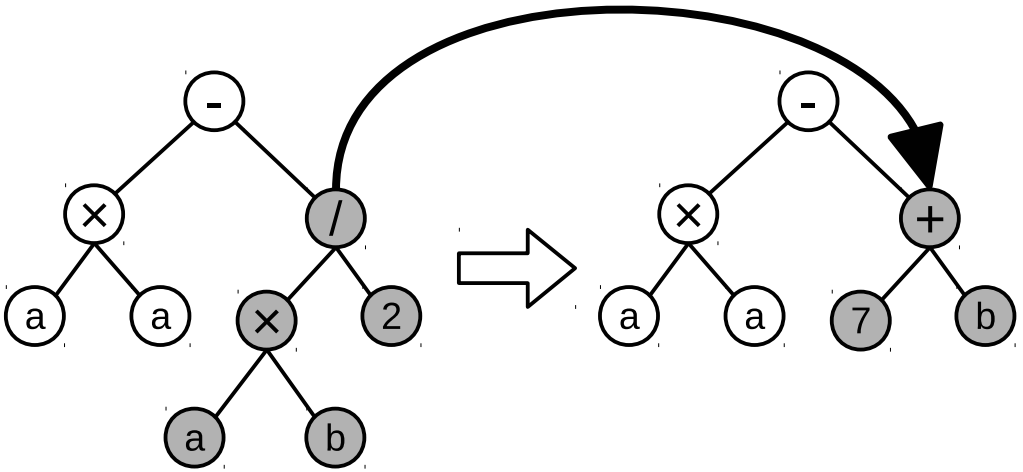


Figure 27: Example of mutation.

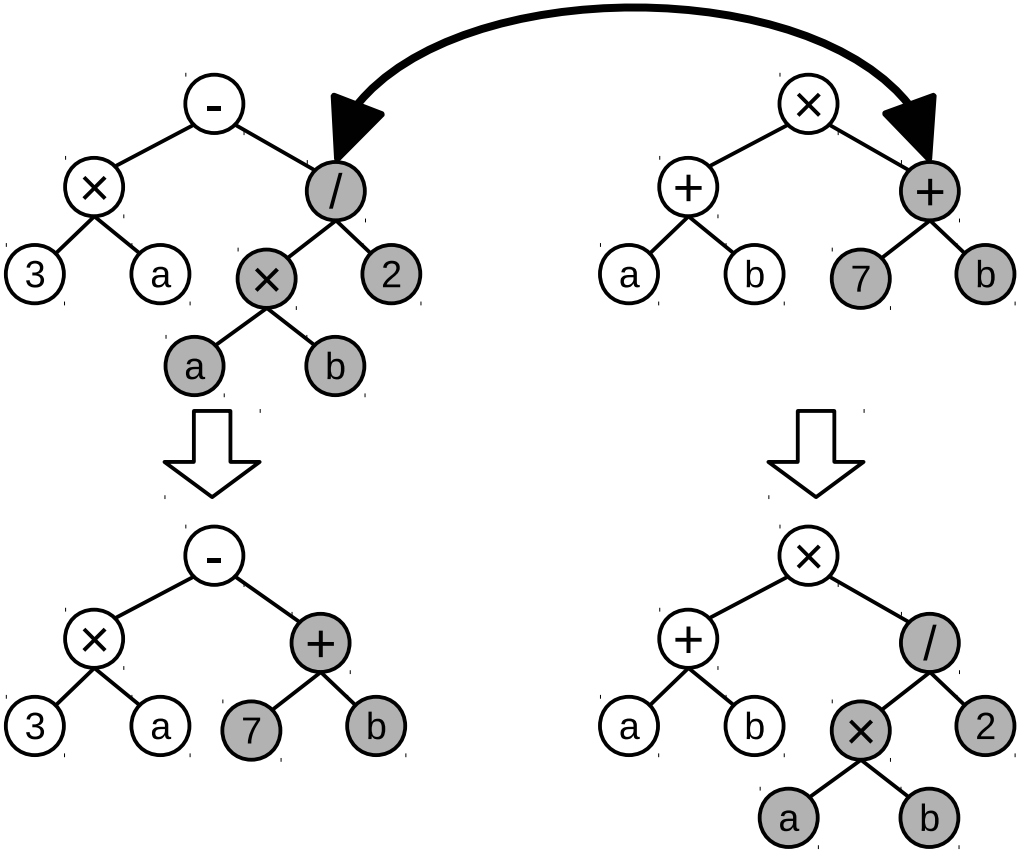


Figure 28: Example of crossover.

Search Dependent Properties:

- **Distance (D)**: the distance (Euclidean) from vertex *u* to the query point
- **Path length (L)**: the length of the path (number of vertices) from the starting vertex to *u*

Search Independent Properties:

- **Edge weight (E)**: the weight of the edge between v and u
- **Vertex degree (N)**: the degree of u
- **Common neighbors (C)**: the number of common neighbors between vertices v and u
- **Jaccard coefficient (J)**: $J(v,u) = \frac{|\tau(v) \cap \tau(u)|}{|\tau(v) \cup \tau(u)|}$, where τ is a function that returns the set of neighbors of a given vertex.
- **Preferential attachment (P)**: $P(v,u) = |\tau(v)| \times |\tau(u)|$
- **Adamic Adar (A)**: $A(v,u) = \sum_{x \in \tau(v) \cap \tau(u)} \frac{1}{\log |\tau(x)|}$
- **Edge redundancy (R)**:

$$R(v,u) = \begin{cases} 0, & \exists w | w \neq v \neq u \wedge w \in \tau(u) \wedge w \in \tau(v) \\ 1, & else \end{cases}$$

Fitness Function

We defined the fitness function of an individual $\hat{f}$ based on the recall, as follows:

$$fitnessNN(\hat{f}) = \frac{1}{|T|} \sum_{t \in T} \left(g(t, \hat{f}) - g(t, L2) \right) \tag{2}$$

where $g(t, s)$ computes the average recall@1 obtained on a training set of queries Q, by using s as scoring function at search, and limiting the number of vertices explored to t

Fitness Function

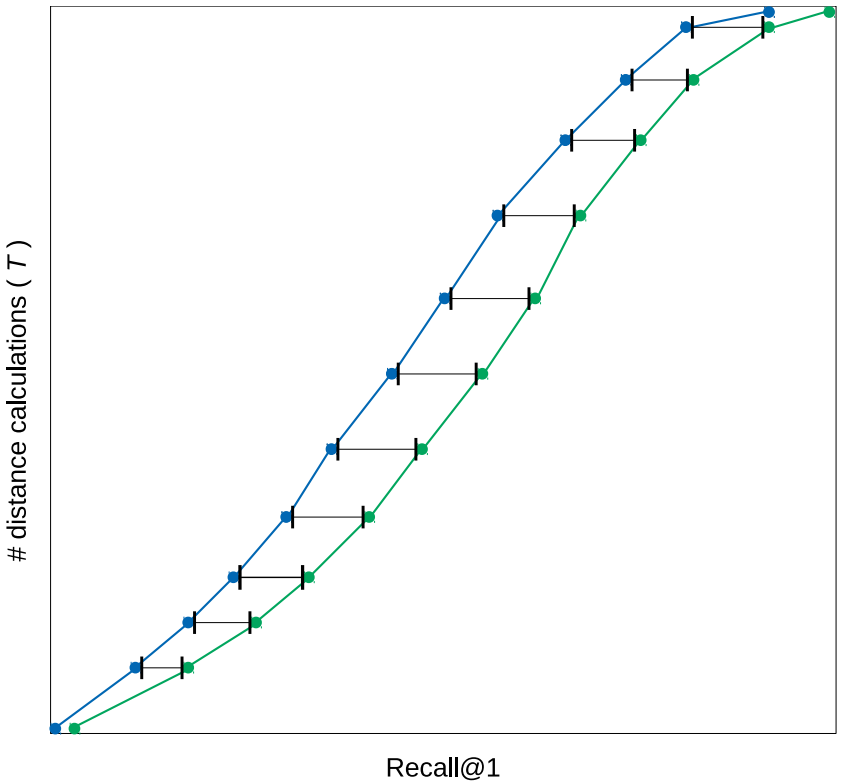


Figure 29: Fitness of GP-based function. Search performance employing GP-based function (green) and L2 (blue).

Scoring Function Example

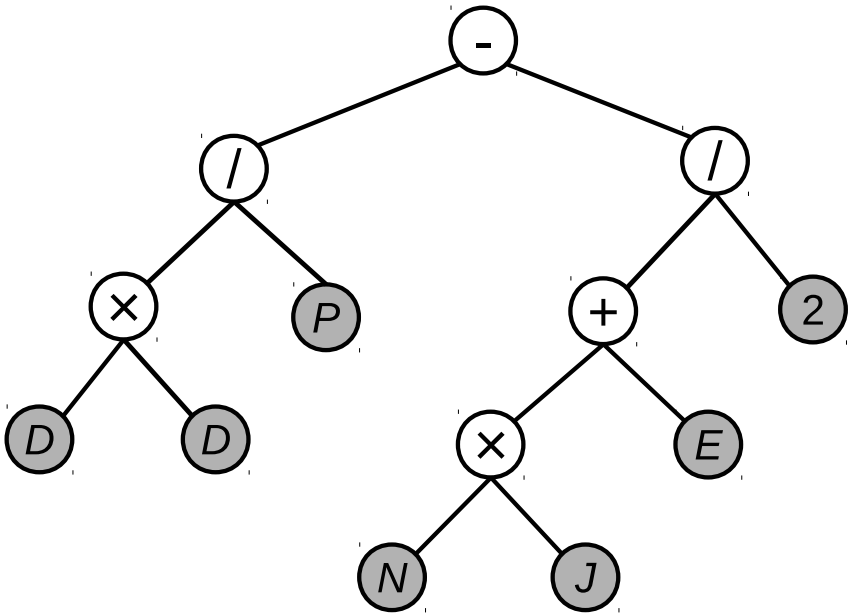


Figure 30: Example of GP-based individual.

$$\hat{f}(x) = \frac{(D \times D)}{P} - \frac{((N \times J) + E)}{2}$$

Validation

Datasets:

- SIFT, GloVe
- YFCC100M: Yahoo-Flickr Creative Commons 100 Million consisting of VLAD vectors, after applying PCA+whitening, we kept the 128 most significant dimensions. We selected randomly a subset of 1 million of images for graph construction and 10 thousand queries

Baselines:

- HCNNG, FANNG, SW-graph, KGraph

Results on GloVe

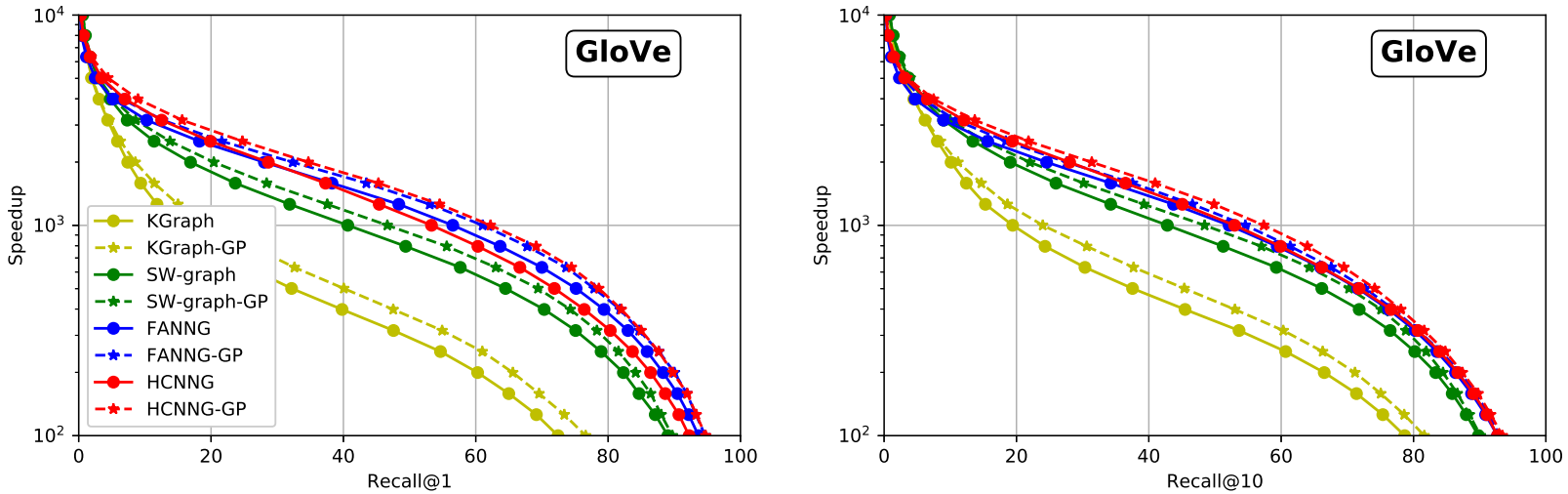


Figure 31: 1-NN and 10-NN search performance on GloVe dataset.

Results on YFCC

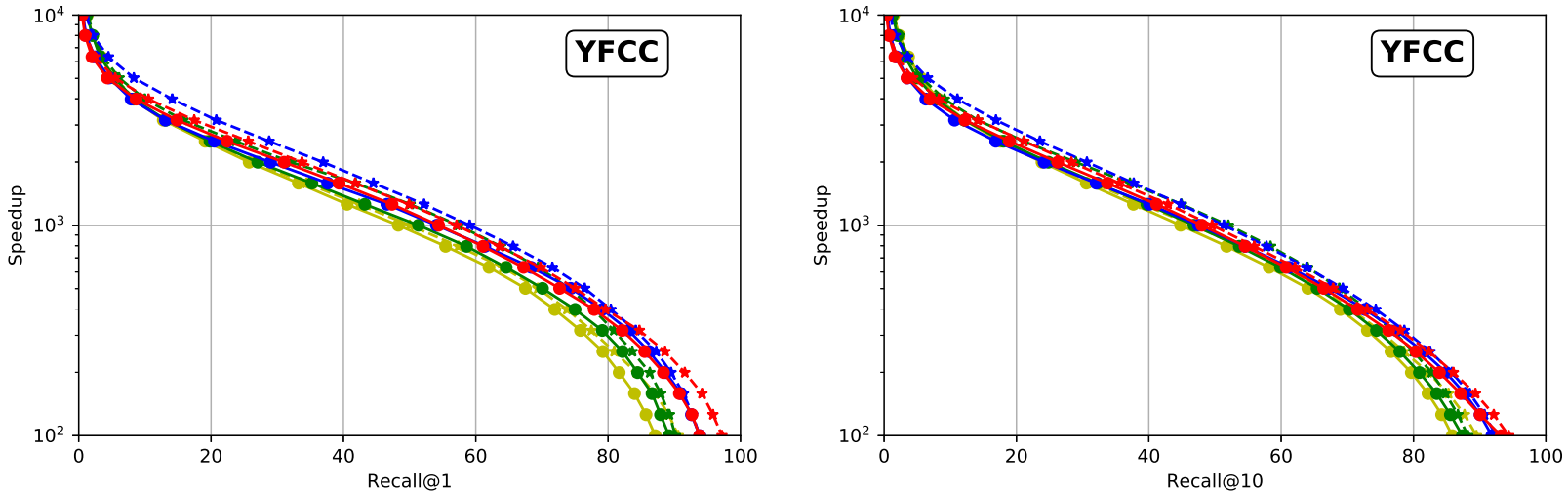


Figure 32: 1-NN and 10-NN search performance on YFCC dataset.

Results on SIFT

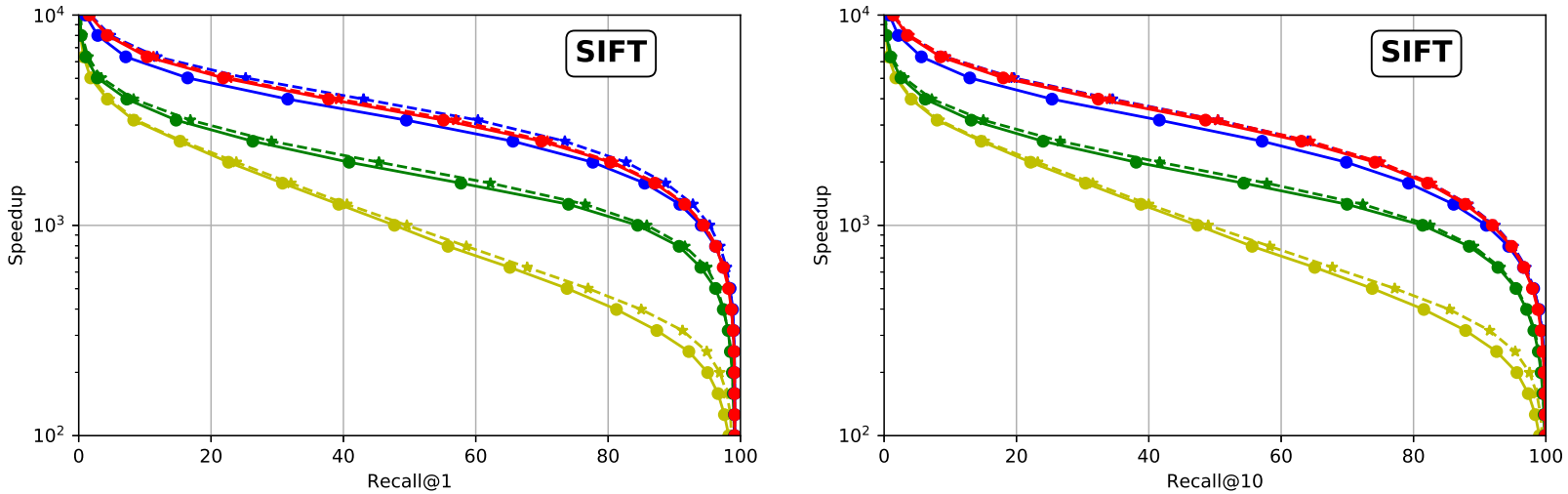


Figure 33: 1-NN and 10-NN search performance on SIFT dataset.

Statistical Tests

Table 2: Statistical paired t-test for 1-NN search ("+": gain, "−": lost, "=": tie, H: HCNNG, F: FANNG, S: SW-graph, K: KGraph).

Speedup	GloVe				SIFT				YFCC			
	H	F	S	K	H	F	S	K	H	F	S	K
$10^{2.0}$	+	+	+	+	=	=	=	+	+	=	+	+
$10^{2.2}$	+	+	+	+	=	−	=	+	+	+	+	+
$10^{2.4}$	+	+	+	+	=	=	+	+	+	+	+	+
$10^{2.6}$	+	+	+	+	=	=	=	+	+	+	+	+
$10^{2.8}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.0}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.2}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.4}$	+	+	+	+	+	+	+	+	+	+	+	+
$10^{3.6}$	+	+	+	=	+	+	+	+	+	+	+	+
$10^{3.8}$	+	+	+	−	+	+	+	+	+	+	+	=
$10^{4.0}$	+	+	=	=	=	+	+	+	+	+	+	=
General	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**

Billion-scale Searches based on Nearest Neighbors Graphs

Problems at Billion Scale

Existing NN graphs approaches:

- Require to load all feature vectors to principal memory
- For million-size datasets they require at most a couple of GB for high dimensional data
- For billion-size they would require up-to terabytes of RAM, which is unpractial

Our Solution

Compress original vectors via quantization methods!

Encoding Vectors via OPQ

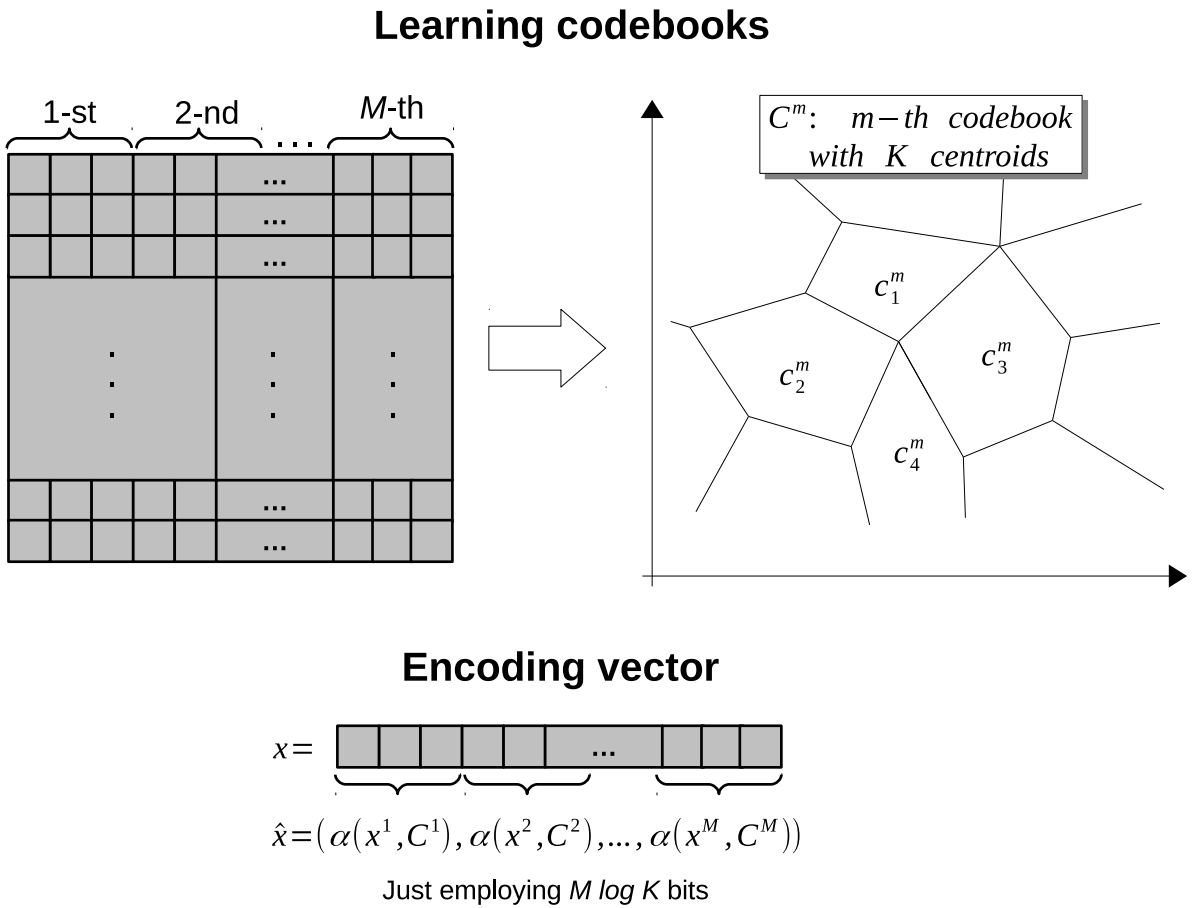


Figure 34: Quantization illustration, where $\alpha(x^m, C^m) = \arg\max_i d(x^m, c_i^m)$.

Problems at Billion Scale

Existing NN graphs approaches:

- Commonly they generate graphs with vertices' degrees in the scale of hundreds
- Cost for loading the data structure itself could be equal to load original vectors in memory

Our Solution

Prune edges at graph construction phase!

- We extended the HCNNG construction algorithm to work with compressed data and add an extra step for pruning edges

Base case:

- When the expected size of clusters is reached, the $MST3$ of current set of points is computed
- The edges of the $MST3$ are ranked, based on a **scoring function**, along with the current set of edges
- Just the top k edges are kept for each vertex

Recursion:

- Two points $\hat{x}_1$ and $\hat{x}_2$ are selected randomly from current set of points
- Two recursive calls for clustering the points closer to $\hat{x}_1$ and $\hat{x}_2$
- Distances between compressed vectors are computed employing a lookup table (to reduce the complexity)

Pruning Edges

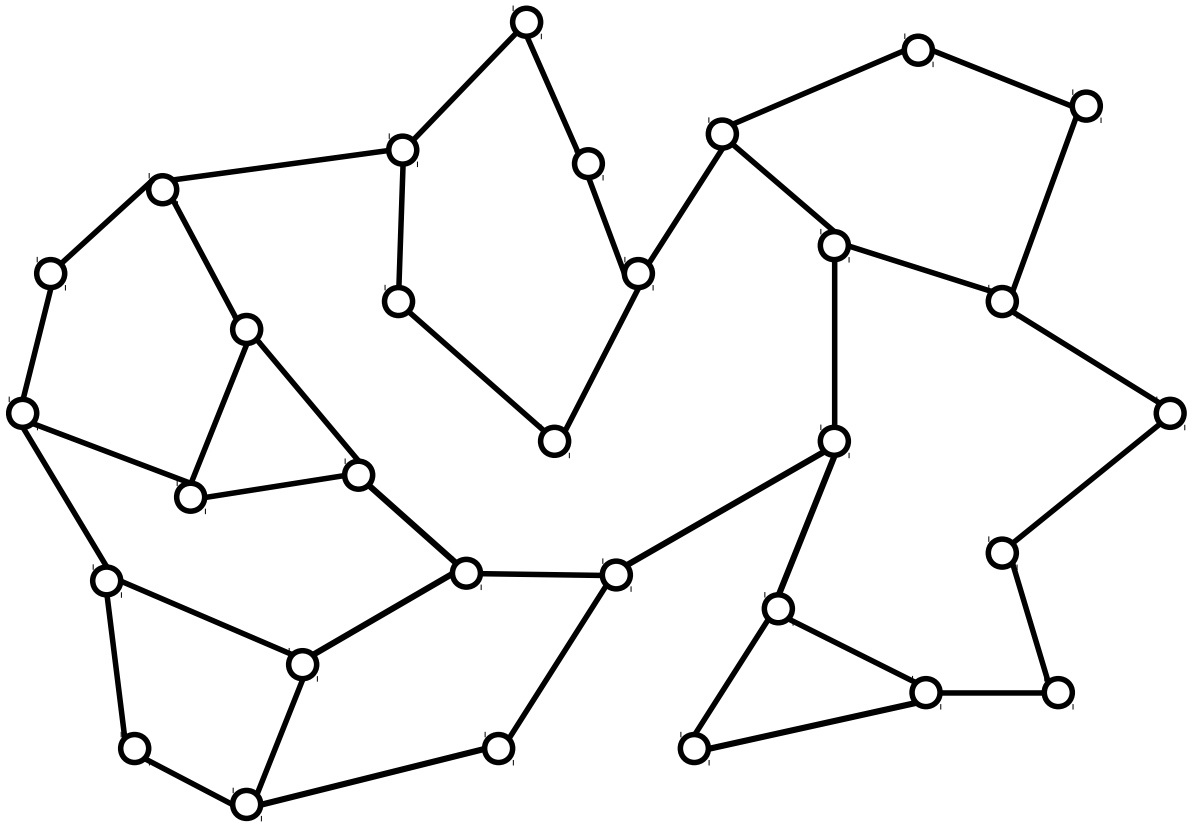


Figure 35: Case base of hierarchical clustering (maximum degree of 3).

Pruning Edges

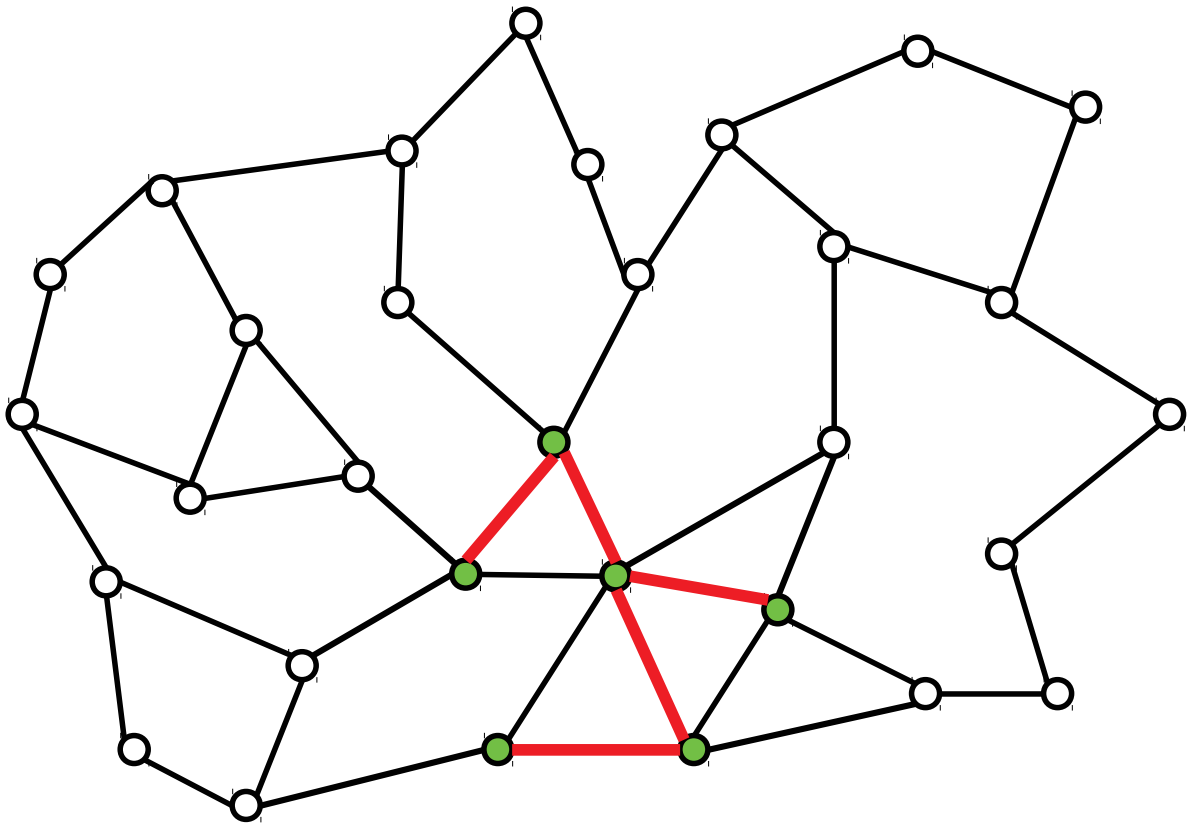


Figure 35: Case base of hierarchical clustering (maximum degree of 3).

Scoring Function

Naïve approach:

- Use the edge weight (distance between vertices) to rank edges for a given vertex

Supervised approach:

- Exploit the topological information of vertices to improve the ranking
- Employ an analog GP framework to learn a near optimal scoring function that combines topological information
- Scoring function will be optimized to create graphs instead of searching on them

Leaning Framework for Pruning Edges

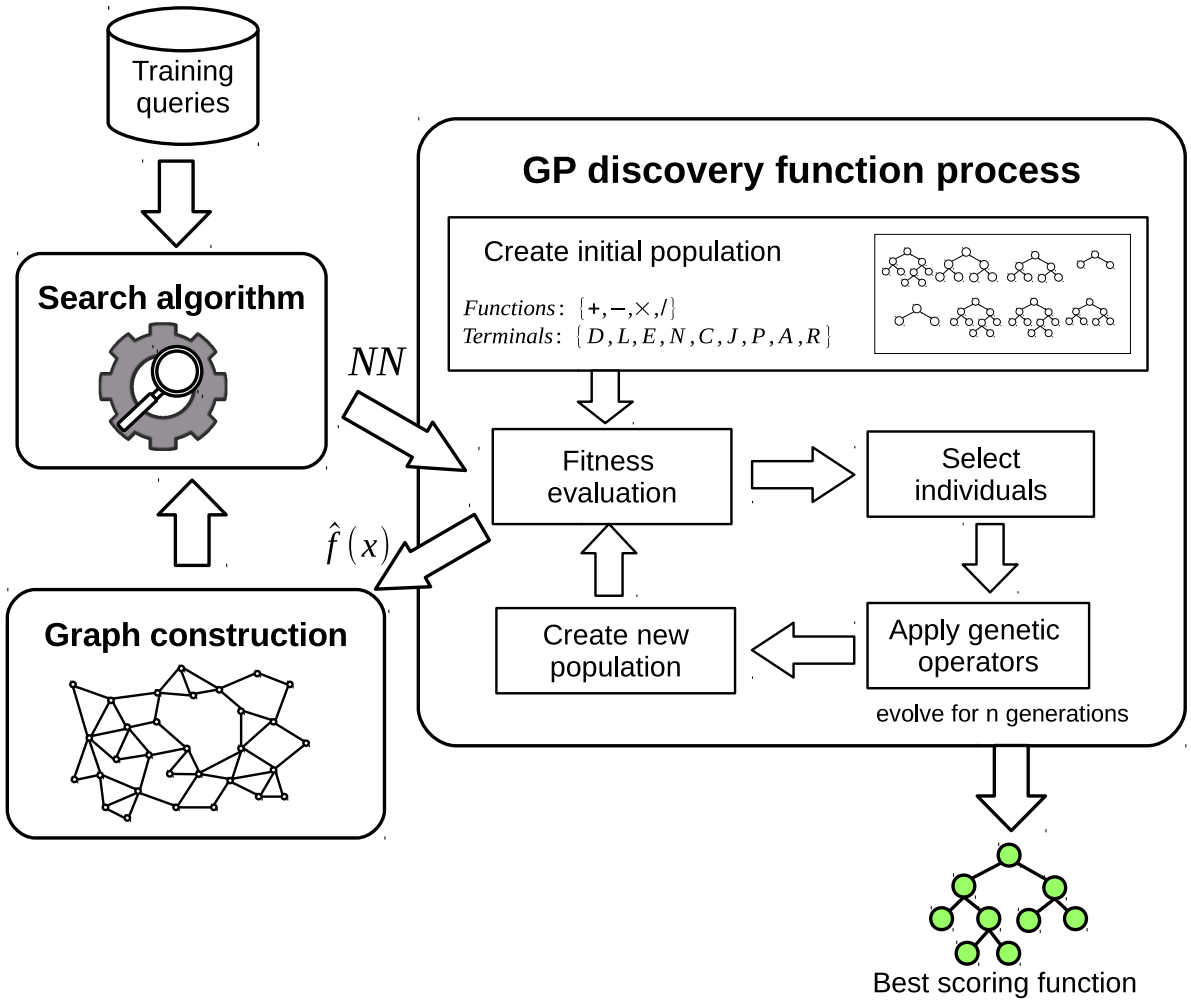


Figure 36: GP-based framework for learning scoring functions for pruning.

Validation

Datasets:

- **SIFT1B:** composed of one billion SIFT vectors and 10 thousand queries for search evaluation
- **DEEP1B:** composed of one billion deep learning-based feature vectors with 96 dimensions, and 10 thousand queries for search evaluation

Parameter Tuning

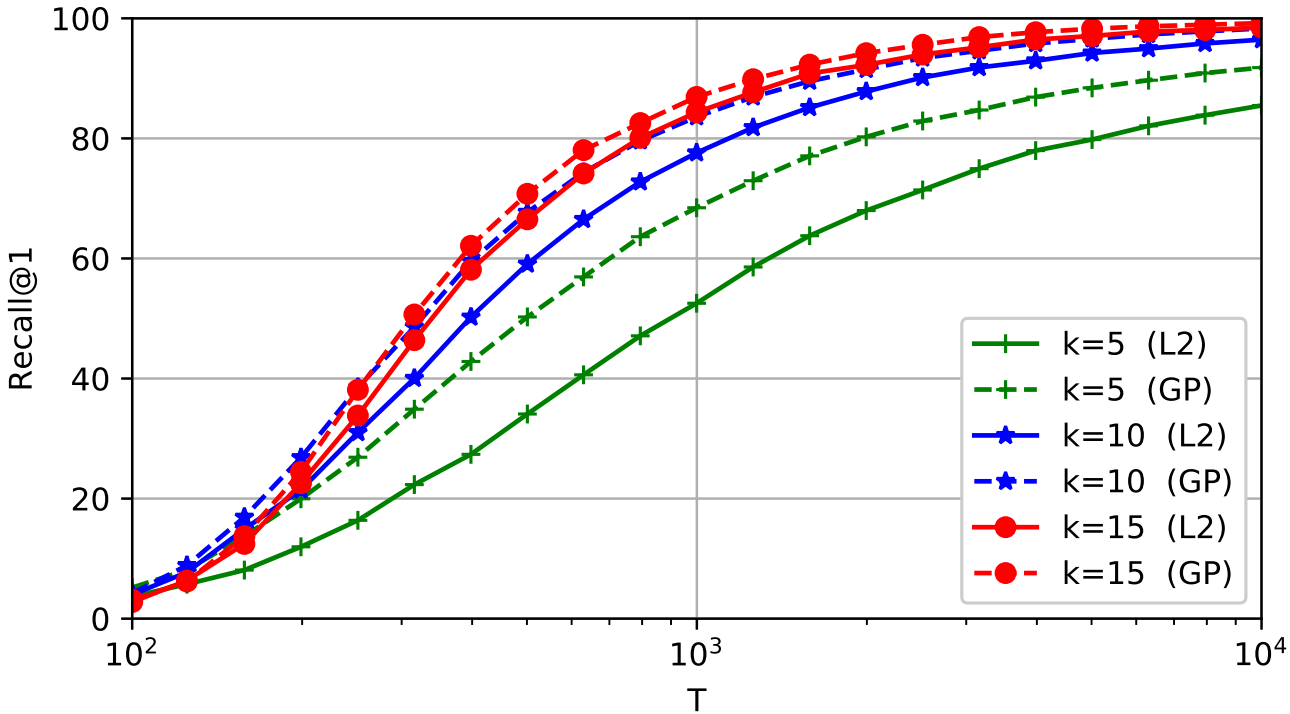
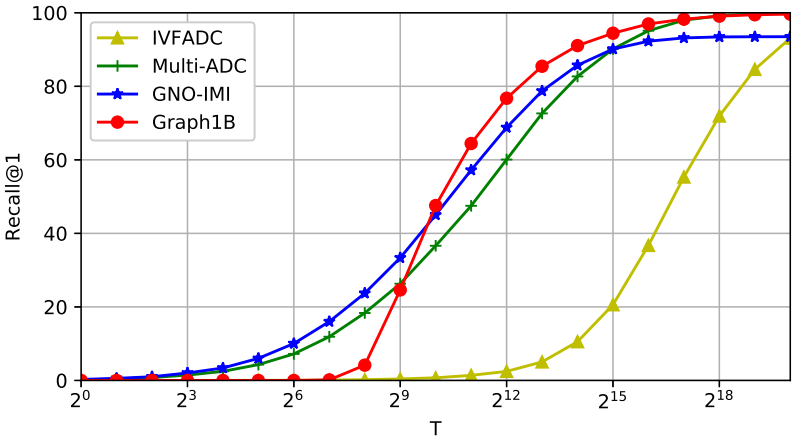


Figure 37: Search performance on DEEP1M using graphs with different degrees and scoring functions for neighbors selection.

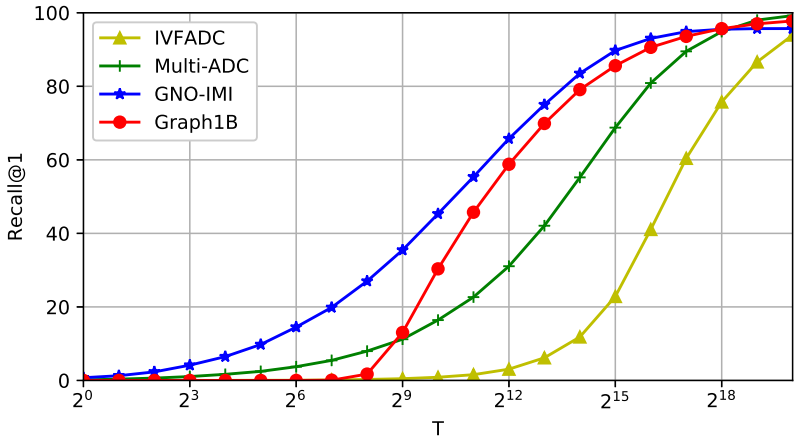
Baselines

- **IVFADC**: based on simple inverted indices. We used our own implementation. For index construction, we used a coarse codebook with $K = 2^{14}$ centroids. Coding via PQ with 12 bits per sub-quantizer
- **Multi-ADC**: based on inverted multi indices. We used our own implementation. For index construction, we used coarse quantizers with $K = 2^{14}$. Coding via OPQ using 12 bits per sub-quantizer
- **GNO-IMI**: based on a generalization of inverted multi indices. We implemented partially this baseline. For index construction, we used coarse quantizers with $K = 2^{14}$. Coding via local OPQ with 8 bits per sub-quantizer

Results on Billion Datasets

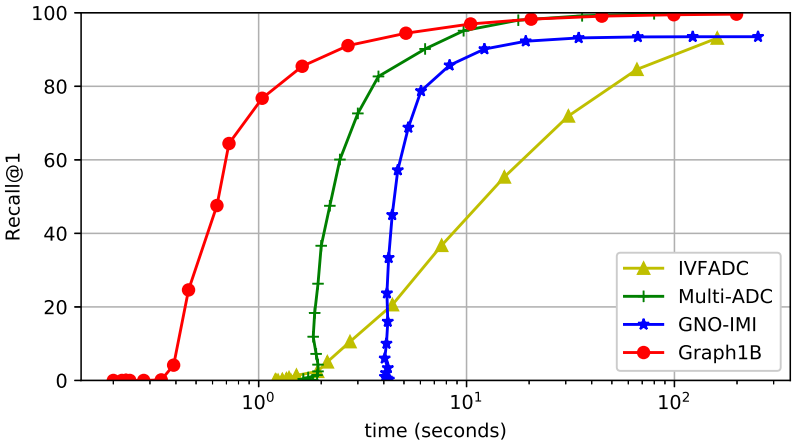


(a) SIFT1B

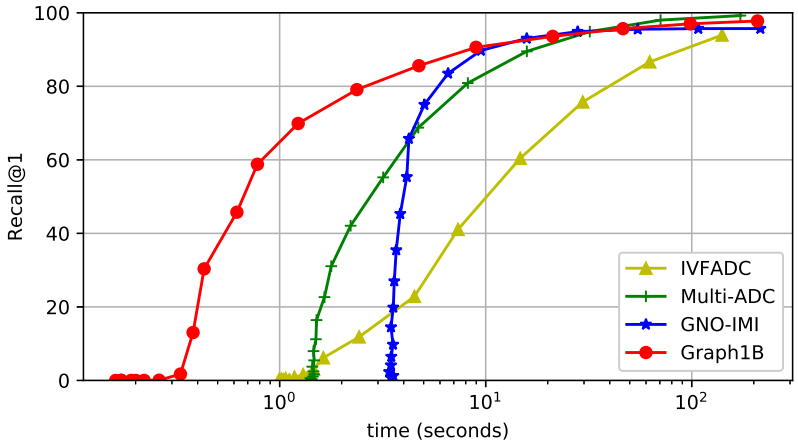


(b) DEEP1B

Time Execution



(a) SIFT1B



(b) DEEP1B

Resources Consumption

Table 3: Resources consumption for indexing techniques.

	Search memory peak (GB)		Construction time (hours)	
Method	**SIFT1B**	**DEEP1B**	**SIFT1B**	**DEEP1B**
Graph1B	72.2	72.2	41.3	37.8
GNO-IMI	31.5	30.9	65.1	55.9
Multi-ADC	28.6	27.5	23.2	18.4
IVFADC	20.0	19.7	18.9	14.2

Conclusions & Future Work

Conclusions

- We research in this thesis about nearest neighbors graphs-based approaches for ANNS
- We proposed a technique for construction of NN graphs with competitive results when compared to the state-of-the-art
- We propose heuristics and a learning framework for improving navigation on graphs searches
- We propose the first NN graph-based technique that scales up-to billion size datasets, with very competitive results when compared to the state-of-the-art for billion scale NN search

Summary

Q1.1 *What is the best way to connect points inside clusters?*
Connecting the points by minimum spanning trees with maximum degree 3 produced the best results among all structures considered.

Q1.2 *How can sub-graphs created by clusters be merged?*
To create the final global graph, we just performed the union of all sets of edges and vertices of sub-graphs.

Q1.3 *Which clustering algorithm should be employed?*
We employed the hierarchical clustering due to their low time complexity.

Summary

Hypothesis 1

The use of multiple randomized clustering leads to the construction of NN graphs, which are faster to traverse at search time

Summary

Q2.1 *How the KD-Trees can be employed to improve the starting vertex selection on NN graph searches?*
Creating multiple global KD-Trees and performing soft searches over them, to finally select the best vertex among those contained in the same leaf as the query.

Q2.2 *How the KD-Trees-like structures can be employed to avoid visit unnecessary vertices at graph navigation process?*
We employed local KD-Trees-like structures to partition the space with respect to each vertex, and, at search time, just explore the neighbors of vertices that are contained in the same sub-space than the query.

Summary

Hypothesis 2

The use of classical tree structures for indexing improves the NN search results on NN graphs, with no significant extra cost

Summary

Q3.1 *How to combine the topological properties with the distance?*
By means of mathematical operators to compose new scoring functions for the priority queue.

Q3.2 *Which learning technique can be used to find near optimal combinations of topological properties?*
We employed genetic programming, since it was employed with success combining different evidences in similar scenarios.

Q3.3 *Which topological properties should be considered?*
The list of topological properties considered was detailed above. We selected those based on their low time complexity to be computed.

Q3.4 *Which function should be optimized in the learning process?*
Since we evaluated the search results using the *recall* metric, we also defined the fitness function based on this metric

Summary

Hypothesis 3

The use of topological information of vertices (along with the distance to query) through a learning scheme leads to a better selection of next vertex (in each step of NN graph search), which fosters the earlier discovering of the true NN

Summary

Q4.1 *Does the compression of original vectors affects to the accuracy of search on NN graphs?*

Yes. In the parameter setup presented above, as it was observed, when the compression error is reduced (by using more bits in the encoding), the search results improved.

Q4.2 *Which heuristics can be used for pruning edges at graph construction?*

We experimented with the distance as the unique criteria to select vertices' neighbors, saving for each vertex just the k-nearest vertices found in the whole graph construction stage.

Summary

Q4.3 *Can we get search performance improvements by exploiting the topological properties of vertices for pruning edges at graph construction?*

Yes. Experiments results demonstrated the improvements in search results (compared to naïve approach) by selecting vertices' neighbors based on the topological information of vertices at graph construction phase.

Q4.4 *Does our NN graph technique still maintain its top search performance at NN search when compared to the state-of-the-art schemes for billion-scale ANNS?*

Yes. Our proposed technique showed best query time/recall trade-off than state-of-the-art techniques considered.

Summary

Q4.5 *How much resources need the proposed NN graph technique in comparison with state-of-the-art?*

The proposed technique required approximately the double of memory and index construction time than best baseline, but it is still far lower than the resources that another NN graph-based technique would require to scale up to billion size dataset, which it would be approximately 8 times the memory required for our proposed approach (considering the same datasets).

Summary

Hypothesis 4

Compressing vectors via quantization schemes and the adoption of suitable pruning strategies at construction time allow the construction of NN graphs on billion-size datasets with a reasonable memory consumption and time construction

Publications

Related to this thesis:

1. **Javier Vargas Muñoz**, Marcos André Gonçalves, Zanoni Dias, Ricardo da Silva Torres: **Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search**. *Pattern Recognition*, 2019 (Chapter 3)
2. **Javier Vargas Muñoz**, Zanoni Dias, Ricardo da Silva Torres: **A Genetic Programming Approach for Searching on Nearest Neighbors Graphs**. *International Conference on Multimedia Retrieval*, 2019 (Chapter 4)
3. **Javier Vargas Muñoz**, Zanoni Dias, Ricardo da Silva Torres: Extension of **A Genetic Programming Approach for Searching on Nearest Neighbors Graphs** (Extension of 3). Preparing for submission to an international journal in 03/2020 (Chapter 4)
4. **Javier Vargas Muñoz**, Zanoni Dias, Ricardo da Silva Torres: **Billion Scale Nearest Neighbors Graph**. Preparing for submission to an international journal in 03/2020 (Chapter 5)

Publications

Other collaborations:

5. **Javier Vargas Muñoz**, Marcos André Gonçalves, Zanoni Dias, Ricardo da Silva Torres: **Learning to Aggregate Rankings**. Submitted to an international journal in 02/2020.
6. Keiller Nogueira, Samuel G. Fadel, Ícaro C. Dourado, Rafael de O. Werneck, **Javier Vargas Muñoz**, Otávio A. B. Penatti, Rodrigo Tripodi Calumby, Lin Tzy Li, Jefersson A. dos Santos, Ricardo da Silva Torres: **Exploiting ConvNet Diversity for Flooding Identification**. *IEEE Geoscience and Remote Sensing Letters* Nogueira et al. 2018.
7. **Javier Vargas Muñoz**, Lin Tzy Li, Ícaro C. Dourado, Keiller Nogueira, Samuel G. Fadel, Otávio Augusto Bizetto Penatti, Jurandy Almeida, Luis A. M. Pereira, Rodrigo Tripodi Calumby, Jefersson A. dos Santos, Ricardo da Silva Torres. RECOD@ Placing Task of MediaEval 2016: **A Ranking Fusion Approach for Geographic-Location Prediction of Multimedia Objects**. *In MediaEval 2016*.

Future Research Directions

- Validate the proposed techniques in other metric and non-metric spaces
- Include more complex topological properties in the learning framework for NN graph search
- Development of GPU-based implementation for accelerate index construction and searches
- Study the use of distributed architectures
- Evaluate the proposed graph construction algorithms in other tasks, e.g, image annotation

Acknowledgements

- CNPq
- CAPES
- FAPESP
- Institute of Computing - UNICAMP

Bibliography I

Artem Babenko and Victor Lempitsky (June 2015). “The Inverted Multi-Index”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.6, pp. 1247–1260. ISSN: 0162-8828.

– (June 2016). “Efficient Indexing of Billion-Scale Datasets of Deep Descriptors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2055–2063.

Tiezheng Ge et al. (Apr. 2014). “Optimized Product Quantization”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.4, pp. 744–755.

Ben Harwood and Tom Drummond (2016). “FANNG: Fast Approximate Nearest Neighbour Graphs”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5713–5722.

Bibliography II

Hervé Jégou, Matthijs Douze, and Cordelia Schmid (2011). "Product quantization for nearest neighbor search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1, pp. 117–128.

Yury A. Malkov and Dmitry A. Yashunin (2016). "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs". In: *arXiv preprint arXiv:1603.09320*.

Yury Malkov et al. (2014). "Approximate nearest neighbor algorithm based on navigable small world graphs". In: *Information Systems* 45, pp. 61–68.

Marius Muja and David G Lowe (2014). "Scalable nearest neighbor algorithms for high dimensional data". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11, pp. 2227–2240.

Keiller Nogueira et al. (Sept. 2018). "Exploiting ConvNet Diversity for Flooding Identification". In: *IEEE Geoscience and Remote Sensing Letters* 15.9, pp. 1446–1450. ISSN: 1558-0571. DOI: 10.1109/LGRS.2018.2845549.

Bibliography III

Javier A. Vargas, Zanoni Dias, and Ricardo da S. Torres (2019). “A Genetic Programming Approach for Searching on Nearest Neighbors Graphs”. In: *Proceedings of the 2019 on International Conference on Multimedia Retrieval*. ICMR’2019. Ottawa ON, Canada: ACM, pp. 43–47. ISBN: 978-1-4503-6765-3.

Javier A. Vargas, Marcos A. Gonçalves, et al. (2019). “Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search”. In: *Pattern Recognition* 96, p. 106970. ISSN: 0031-3203.

Time Execution HCNNG

Table 4: Total time for 100K queries (in seconds) using classical search approach vs the guided proposed (without compiler optimizations).

	SIFT			GIST			GloVe		
Max. Distance Calculations	Query time (seconds)		Gain (recall)	Query time (seconds)		Gain (recall)	Query time (seconds)		Gain (recall)
	Classic	Guided		Classic	Guided		Classic	Guided	
100	30.412	37.141	21.37%	87.118	91.605	5.60%	26.438	29.346	6.14%
250	78.675	85.869	27.39%	236.031	227.334	20.92%	68.176	72.196	14.27%
500	161.466	171.058	5.16%	449.054	445.305	11.20%	141.542	145.689	10.56%
750	247.852	260.133	1.77%	679.243	687.949	5.22%	218.889	226.076	6.44%
1000	338.050	348.214	0.76%	910.847	921.767	2.70%	294.816	305.367	3.00%

Time Execution HCNNG

Table 5: Total time for 100K queries (in seconds) using classical search approach vs the guided proposed (with -O3 option for compiler optimization).

	SIFT			GIST			GloVe		
Max. Distance Calculations	Query time (seconds)		Gain (recall)	Query time (seconds)		Gain (recall)	Query time (seconds)		Gain (recall)
	Classic	Guided		Classic	Guided		Classic	Guided	
100	10.969	10.797	21.37%	43.558	25.029	5.60%	9.167	9.659	6.14%
250	28.243	25.617	27.39%	110.971	59.117	20.92%	24.036	22.552	14.27%
500	58.703	50.911	5.16%	226.253	115.686	11.20%	50.594	44.631	10.56%
750	91.684	78.329	1.77%	345.229	176.746	5.22%	78.676	68.131	6.44%
1000	125.634	104.463	0.76%	464.278	234.372	2.70%	110.736	90.916	3.00%

HCNNG: Results for 100-NN

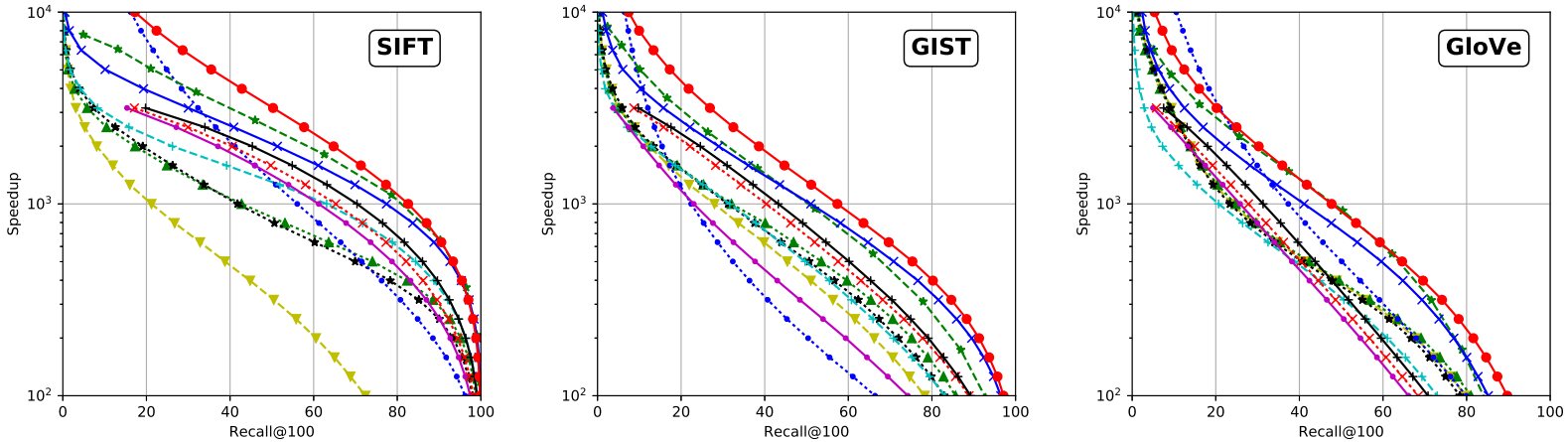


Figure 40: Results for 100-NN.

Table 6: GP parameter values for learning process to improve search.

Parameter	Value
Size population	400
Depth individuals	5
Functions	$\{+, -, /, *, min, max\}$
Terminals	$\{E, N, C, J, P, A, R\} \cup random[-1, 1]$
Num generations	100
Genetic operators	reproduction (5%), mutation (10%), crossover (85%)
Fitness function	$T = \{10^2, 10^{2.1}, 10^{2.2}, \ldots, 10^{3.9}, 10^4\}$

Time Execution GP-based Search

Table 7: Measured time for the test set in seconds for both classic and our GP-based search, on the GloVe dataset.

	HCNNG			FANNG			SW-graph			KG
	time (seconds)		gain	time (seconds)		gain	time (seconds)		gain	time (second
T	classic	GP	(recall)	classic	GP	(recall)	classic	GP	(recall)	classic	G
$10^{3.0}$	9.85	12.22	+8.89%	9.03	10.59	+4.46%	10.81	12.34	+5.97%	10.44	1
$10^{3.2}$	15.95	20.43	+7.80%	13.99	16.35	+3.66%	17.74	20.18	+5.35%	18.39	18
$10^{3.4}$	26.57	29.11	+5.60%	22.42	24.83	+2.43%	27.68	31.34	+3.95%	28.36	28
$10^{3.6}$	45.98	50.08	+3.96%	36.03	39.71	+1.78%	46.77	53.38	+2.62%	45.88	46
$10^{3.8}$	76.44	83.30	+3.33%	56.63	62.55	+1.41%	77.78	89.14	+1.76%	72.14	75
$10^{4.0}$	131.29	142.72	+2.50%	91.51	101.31	+1.01%	133.97	153.43	+0.69%	119.15	120

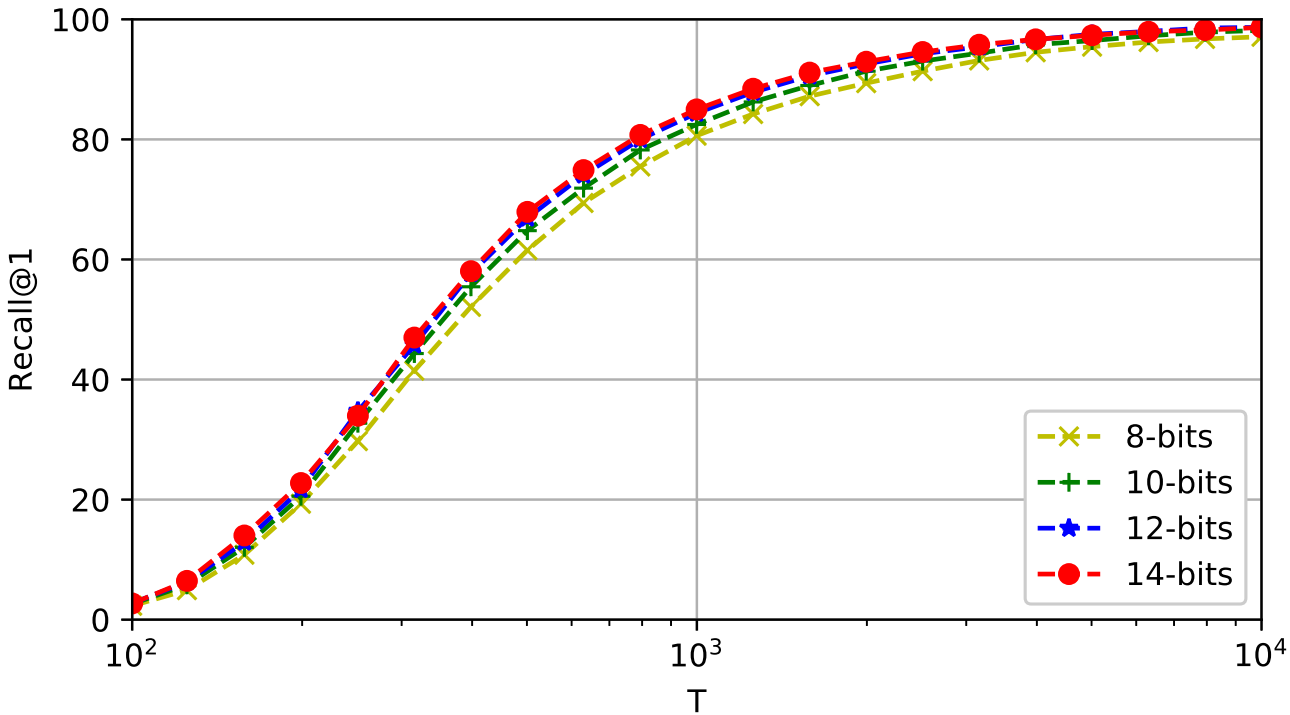


Figure 41: Search performance on DEEP1M encoding with different numbers of bits per sub-quantizer.

Table 8: GP parameter values for learning process to prune edges.

Parameter	Value
Size population	200
Depth individuals	4
Functions	$\{+, -, /, *, min, max\}$
Terminals	$\{E, N, C, J, P, A, R\} \cup random[-1, 1]$
Num generations	20
Genetic operators	reproduction (5%), mutation (10%), crossover (85%)
Fitness function	$T = \{10^2, 10^{2.1}, 10^{2.2}, \ldots, 10^{3.9}, 10^4\}$

Time Execution on SIFT1B

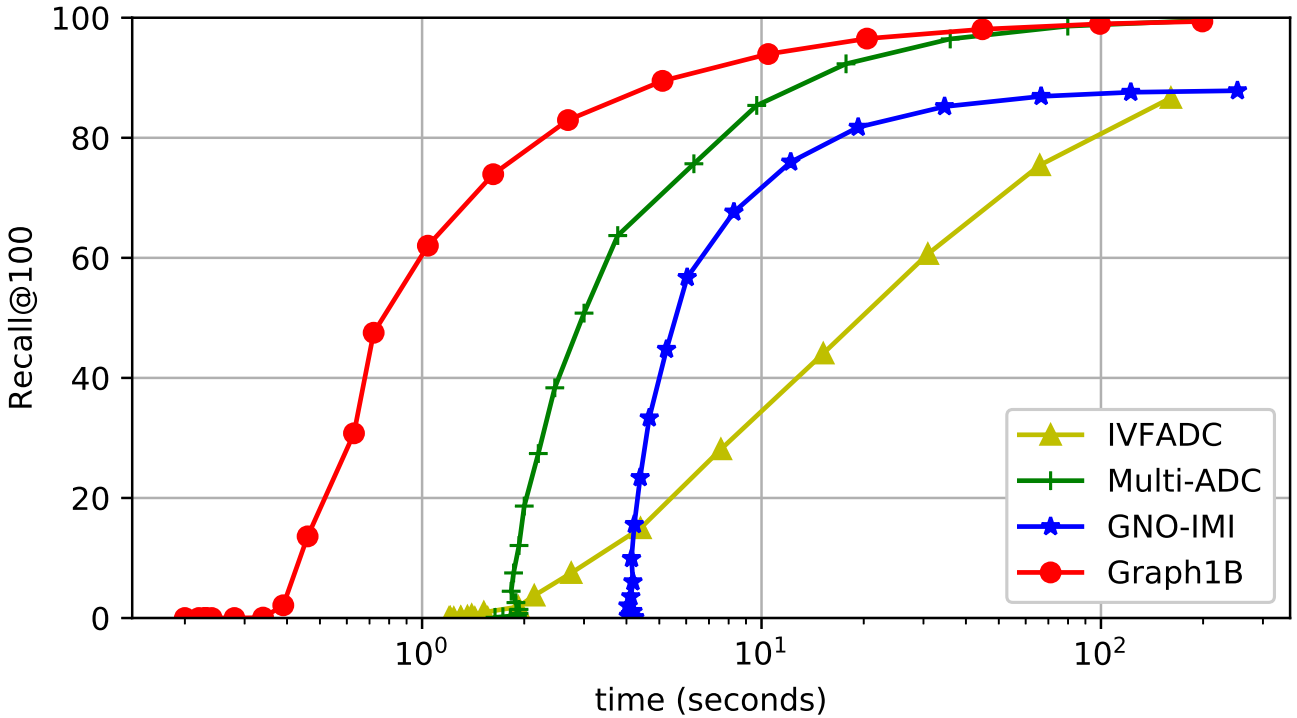


Figure 42: Time execution for 100-NN search