Exploring Efficient Similarity Search Algorithms with K-Nearest Neighbor Graph

Final Year Project – Term 2

HU Bin (Evan)

Agenda

- Review of Hashing/Graph-Based Algorithms
- GNNS on KNN Graph
- GNNS on PCA Hashing-Based AKNN Graph
- Future Work
- Q&A

1. Review of Hashing/Graph-Based Algorithms

Hashing Based Methods

Common in LSH, PCAH, ITQ:
Use hashing matrix to hash the original data to buckets

Locality Sensitive Hashing (LSH)

Map each vector in $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$ to a c-bit binary code $\boldsymbol{H}(\boldsymbol{x})$

$$\boldsymbol{H}(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \dots, h_c(\boldsymbol{x})]$$

where

$$h_l(\boldsymbol{x}) = sgn(x\boldsymbol{w}_l^T) \in \{0, 1\}$$

is the l-th hash function and with $\boldsymbol{w}_l$ being a randomly generated weight vector

Locality Sensitive Hashing (LSH)

Use Hamming distance $d_H(\boldsymbol{a}, \boldsymbol{b})$ as a proxy and scan through items that fall in hash buckets that fall within a radius r_H to $\boldsymbol{H}(\boldsymbol{q})$, i.e., return

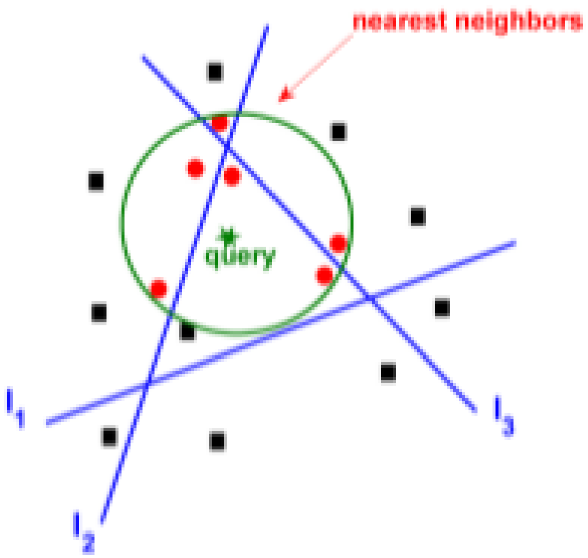
$$argmin_{\boldsymbol{x} \in \{\boldsymbol{x}: d_H(\boldsymbol{x}, \boldsymbol{q}) \leq r_H\}} d(\boldsymbol{x}, \boldsymbol{q})$$

where d_H is the Hamming distance function and $d_H(\boldsymbol{a}, \boldsymbol{b})$ is the Hamming distance between the two binary codes

Note that d_H can be efficiently computed with low-level hardware operation XOR

Locality Sensitive Hashing (LSH)

Example LSH in 2-D space



Green: the query point $\boldsymbol{q}$ green star
Red: its NN neighbors
Black: other data points

Hyperplanes l1, l2, and l3 to separate the 2-D space into seven parts

To achieve a 100% recall, one must specify the parameter to be $r_H \geq 2$

K-Nearest Neighbor (kNN) Graph

Let $\mathcal{N}_k(\boldsymbol{x})$ denote the set of k nearest nodes of data point $\boldsymbol{x}$ in the reference set $\boldsymbol{X} = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$

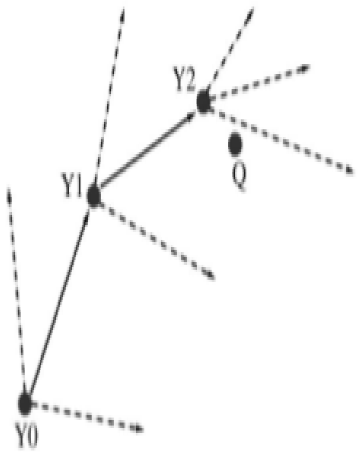
A KNN graph is a directed graph $\mathcal{G} = (\boldsymbol{X}, \boldsymbol{E})$, where $\boldsymbol{E}$ is the directed edge set where

$$\text{vertex } \boldsymbol{x}_i \text{ is connected to vertex } \boldsymbol{x}_j \Leftrightarrow \boldsymbol{x}_j \in \mathcal{N}_k(\boldsymbol{x}_i).$$

starting from a randomly selected item, iterative hill-climbing takes place and lead the search to reach near the query target Q

Graph Nearest Neighbor Search (GNNS)

Starting from a randomly selected item, iterative hill-climbing takes place and lead the search to reach near the query target Q



$\mathcal{S} \leftarrow \{\}$ is the set of visited nodes
$\mathcal{U} \leftarrow \{\}$ is the set of ρ distance measures of the visited nodes against $\boldsymbol{q}$
for $r = 1, \ldots, R$ **do**
  Randomly select an item $\boldsymbol{y}_0$ from $\boldsymbol{X}$
  for $t = 1, \ldots, T$ **do**
    Update $\boldsymbol{y}_t = argmin_{\boldsymbol{y} \in \mathcal{N}_E(\boldsymbol{y}_{t-1})} \rho(\boldsymbol{y}, \boldsymbol{q})$
    Update $\mathcal{S} = \mathcal{S} \cup \mathcal{N}_E(\boldsymbol{y}_{t-1})$
    Update $\mathcal{U} = \mathcal{U} \cup \{\rho(\boldsymbol{y}, \boldsymbol{q}) \mid \boldsymbol{y} \in \mathcal{N}_E(\boldsymbol{y}_{t-1})\}$
Return the k items in $\mathcal{S}$ with minimum corresponding ρ in $\mathcal{U}$

2. GNNS on KNN Graph

Graph Construction

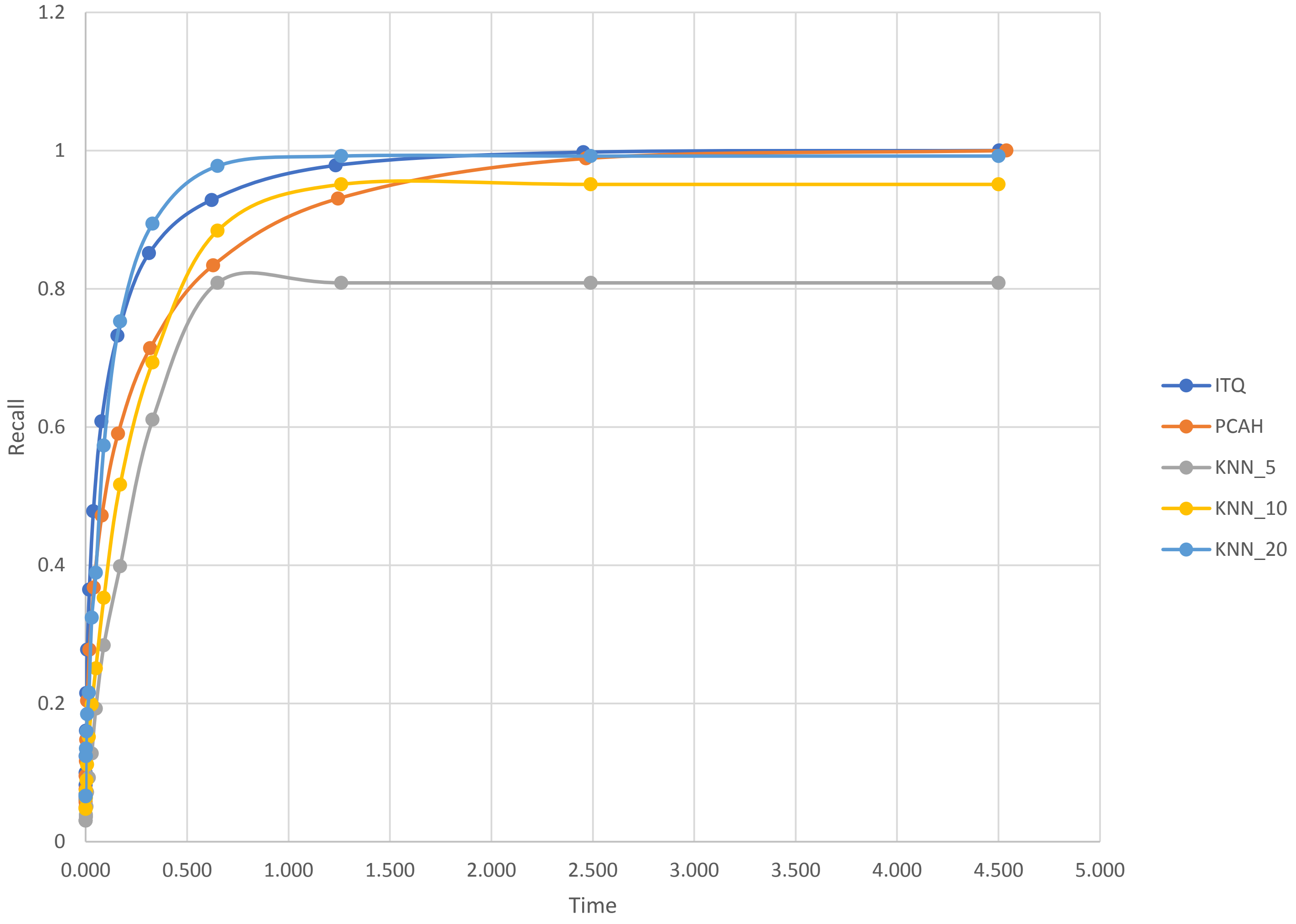
Dataset	Cardinality	Dimension	Construction Method	Time	Graph Quality
SIFT1m_10k	10,000	128	naïve_knn	**4m46.990s**	100%
SIFT1m_10k	10,000	128	fast_knn	**2m51.005s**	100%
SIFT1m_10k	10,000	128	pcah_aknn	**0m25.055s**	84.17%

naïve_knn: n^2 distance computations, O($nlogn$) finding top k

fast_knn: $\frac{n^2}{2}$ distance computations, O(n) finding top k

pcah_knn: even better

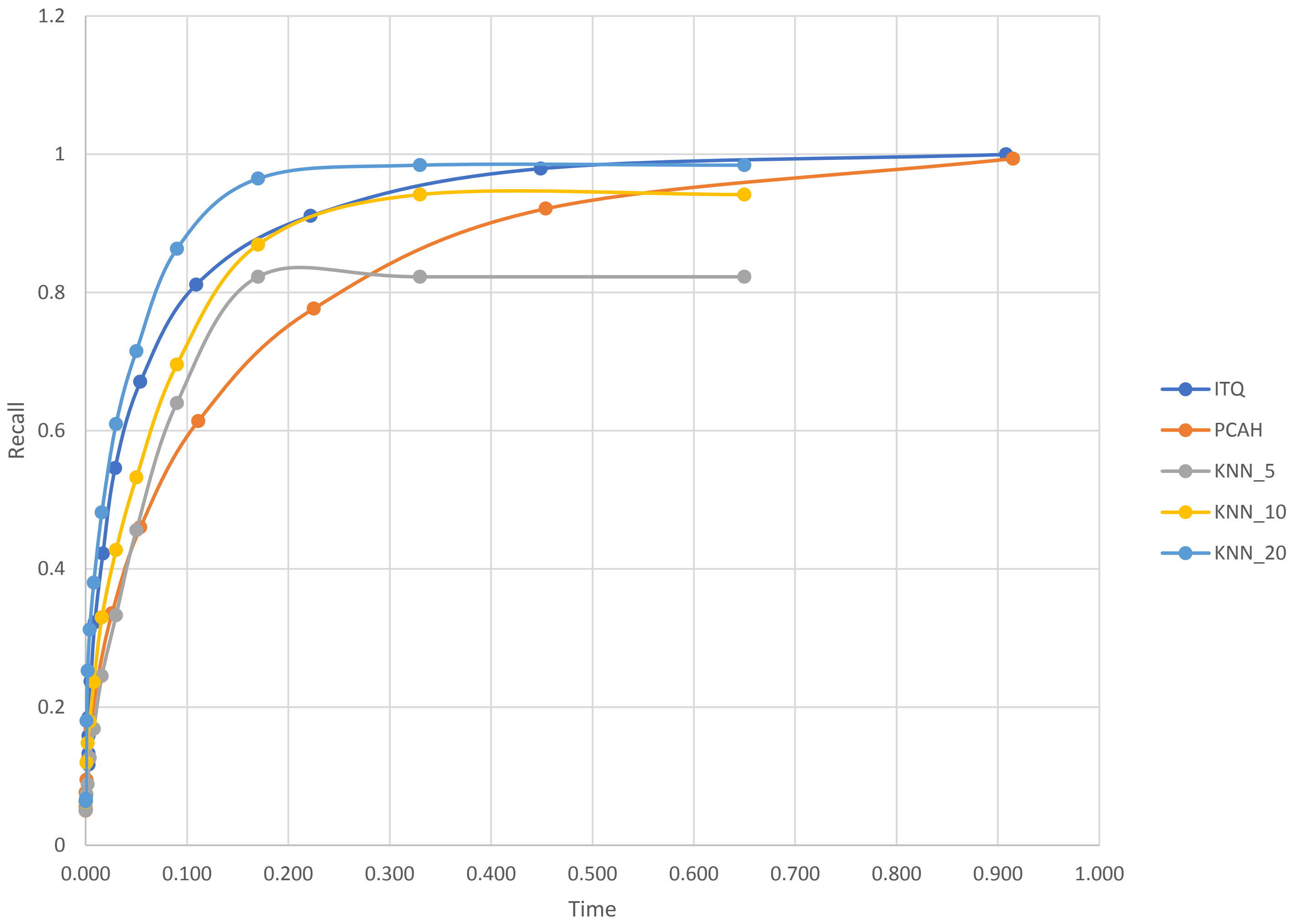
GNNS Search Results – CIFAR60k



Beating Hashing Techniques

K ↑, Performance ↑

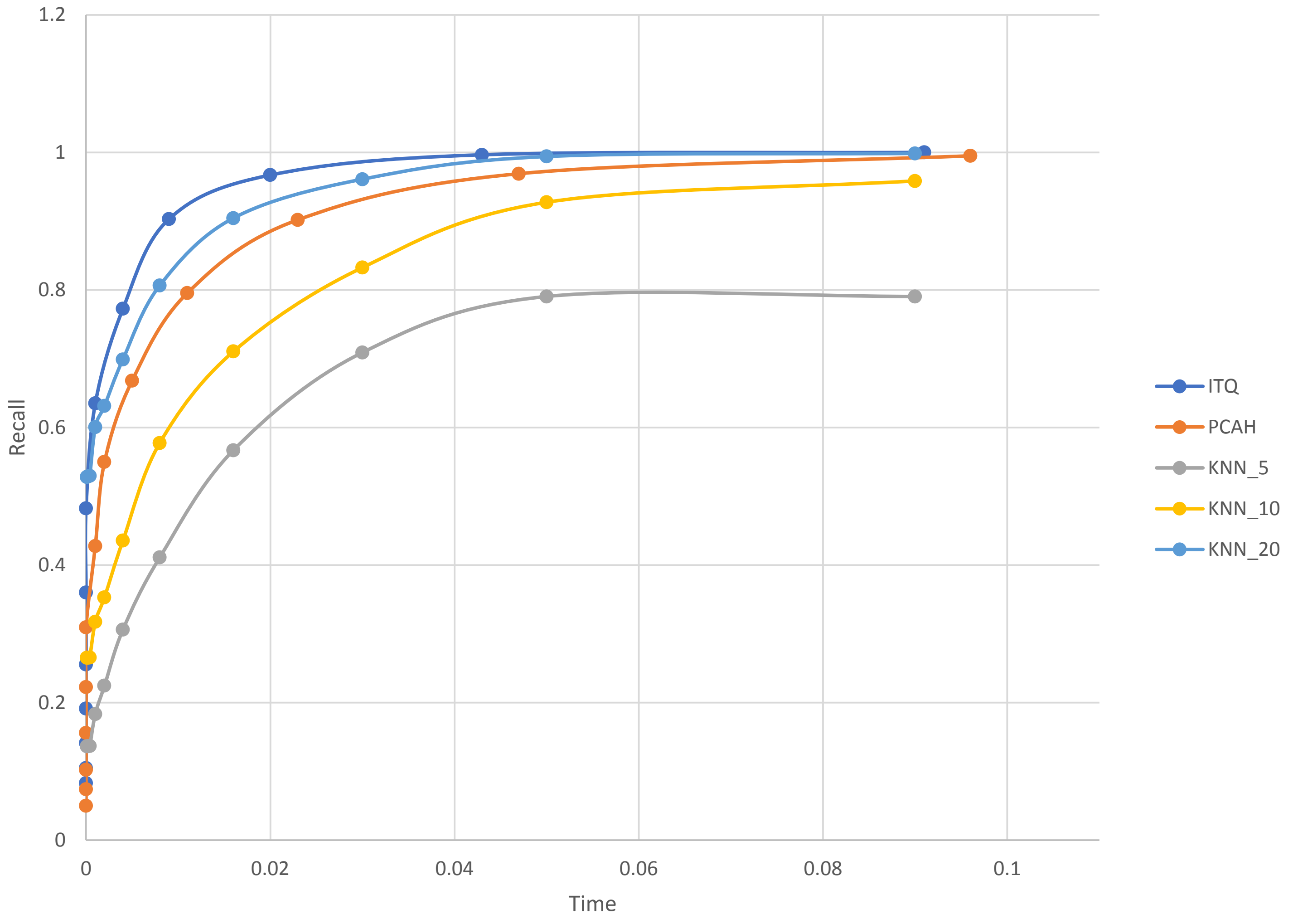
GNNS Search Results – GIST_10k



Beating Hashing Techniques

K ↑, Performance ↑

GNNS Search Results – GIST_10k



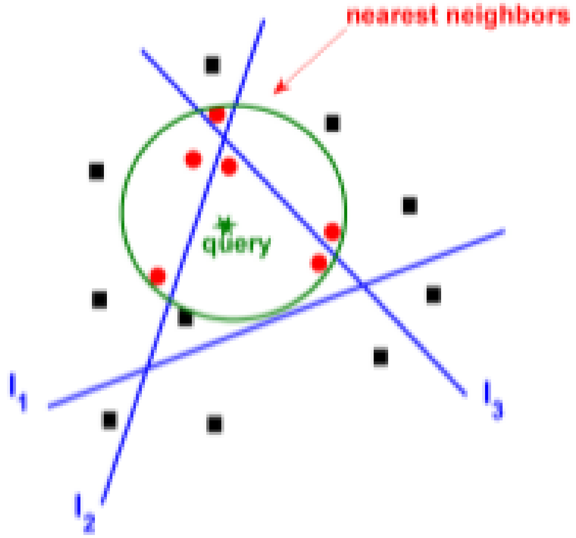
Beating Hashing Techniques

K ↑, Performance ↑

3. GNNS on PCA Hashing-Based AKNN Graph

Constructing Approximate KNN (AKNN) with PCAH

Idea



Limit the search range

Results

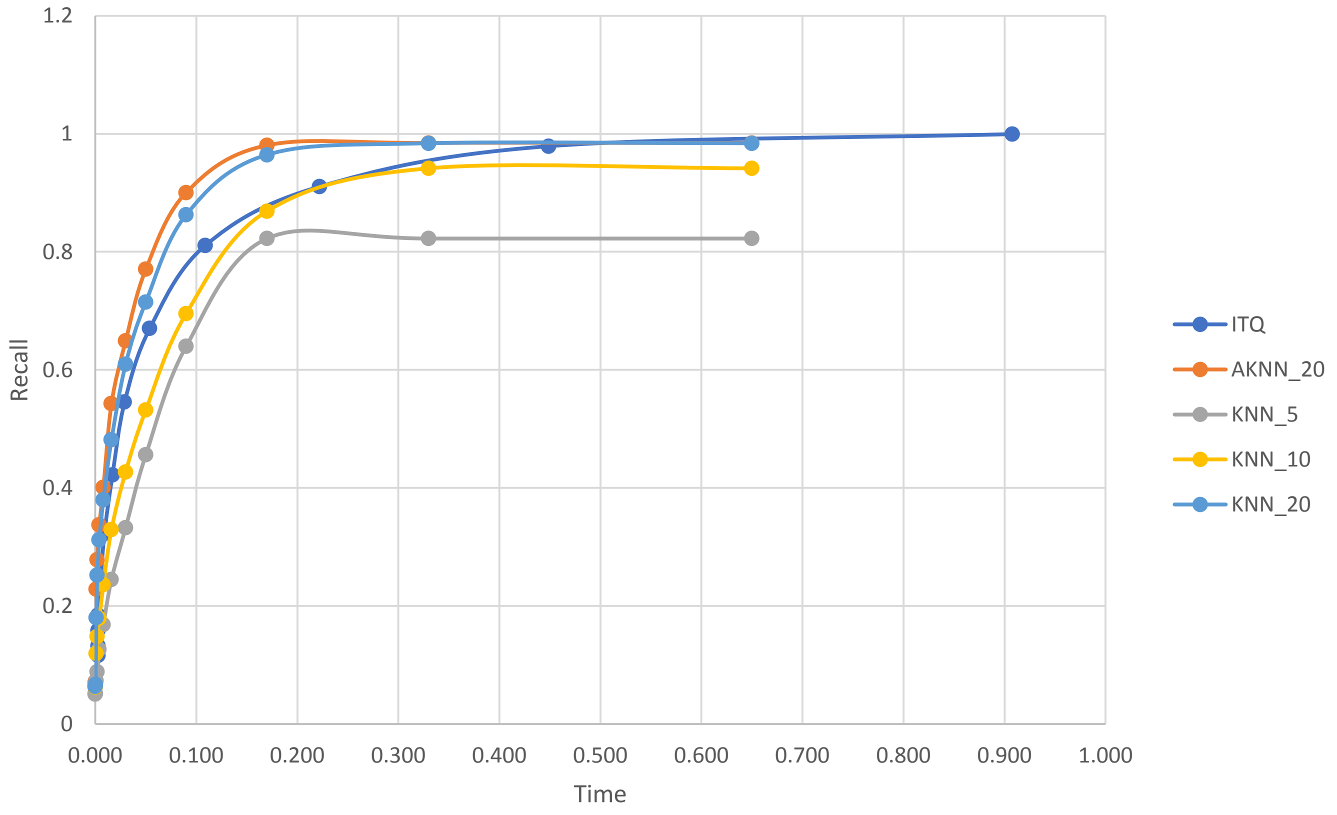
Code Len	8			Code Len	16			Code Len	32	
maxHam	Recall (%)	Scan Rate(%)		maxHam	Recall (%)	Scan Rate(%)		maxHam	Recall (%)	Scan Rate(%)
2	67.44	14.98		4	40.51	4.13		8	10.50	0.45
4	**97.46**	**63.65**		**8**	**97.39**	**59.79**		**16**	**96.22**	**56.93**

GIST_10k

Code Len	8			Code Len	16			Code Len	32	
maxHam	Recall (%)	Scan Rate(%)		maxHam	Recall (%)	Scan Rate(%)		maxHam	Recall (%)	Scan Rate(%)
2	84.17	14.64		4	69.30	4.46		8	39.40	0.65
4	**99.37**	**62.36**		**8**	**99.67**	**59.01**		**16**	**99.79**	**56.24**

SIFT1m_10k

Perform GNNS on AKNN

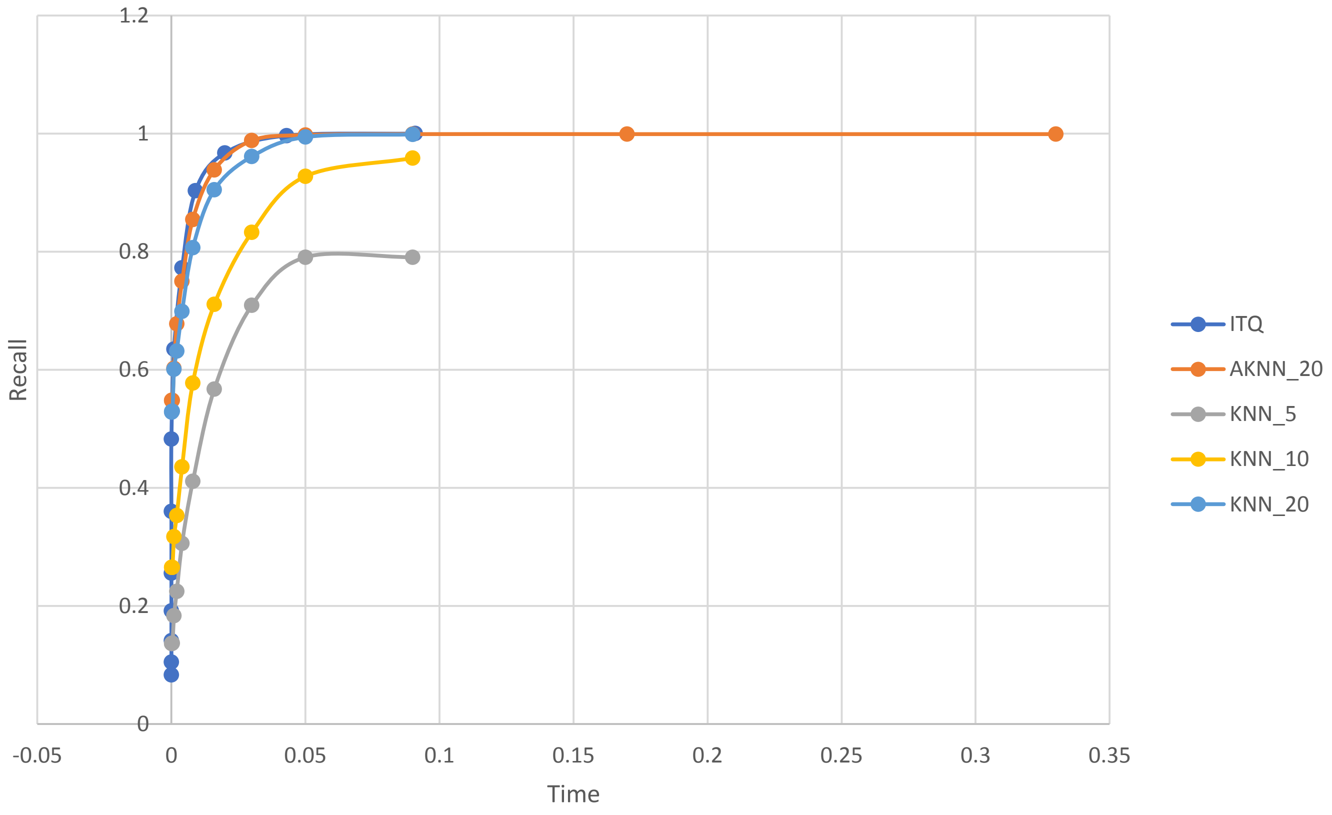


GIST_10k

Similar recall

More efficient construction

Perform GNNS on AKNN



GIST_10k

Similar recall

More efficient construction

4. Future Work

4. Future work

Work

1. Construct AKNN with more iterations of hashing
2. Use PCA hashing to generate seeds for searching
3. Small-World Graph instead on KNN

Report

1. Compare with LSH based AKNN results
2. Add CIFAR60k

THANK YOU

Q&A

Appendix

- Distance measure
- From NN Search to ANN Search
- ML Methods
- Time complexity analysis
- Hashing methods

Appendix – Distance measure

We have assumed l2-distance, which is popular

Other distance measures:

Euclidean distance other than l2, cosine similarity, Jaccard similarity

Appendix – From NN Search to ANN Search

NN Search

- Linear Scan
- Tree-based structures

Optimal guaranteed but unacceptably slow

ANN Search

- Hashing: LSH, PCAH, ITQ
- kNN graph search

Optimal guaranteed but unacceptably slow

Appendix – ML Methods

There are advantages that similarity search methods have:

- Avoid overfitting of parameters, because no parameter learning is required
- Can naturally handle a huge number of classes
- Require no training/learning phase

In the studies, a method named Naive-Bayes Nearest-Neighbor (NBNN) similarity search based methods are also be shown to perform in line with top leading learning-based image classifiers

Appendix – Time complexity analysis

Algorithm	Offline	Online
LSH	$O(cdN)$	$O(N/2^c * r(d*\log R))$
PCAH/ITQ	$O(d\hat{}2(d+N))$	$O(N/2^c * r(d*\log R))$
kNN	$O(N^2(d+\log k))$	$O(sdE * \log k)$

Appendix - Principle Component Analysis Hashing (PCAH)

Instead of randomly generated hash functions, try to obtain more meaningful ones so that the variance of each bit is maximized and the bits are pairwise independent, i.e., maximize the objective function:

$$\mathcal{L}(\boldsymbol{W}) = \sum_k var(h_k(\boldsymbol{x})) = \sum_k var\left(sgn\left(\boldsymbol{w}_k^T\boldsymbol{x}\right)\right), \ \frac{1}{N}\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}$$

where $\boldsymbol{B}$ is the binary code matrix generated by the hash functions

Appendix - Principle Component Analysis Hashing (PCAH)

The objective function is undifferentiable, hence the relaxation to maximize the variance of the projected values:

$$\hat{\mathcal{L}}(\boldsymbol{W}) = \sum_k \mathbb{E}(\| \boldsymbol{x}\boldsymbol{w}_{\boldsymbol{k}} \|^2) = \frac{1}{n} tr(\boldsymbol{W}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{W}), \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$$

The constraint requires the hashing hyperplanes to be orthogonal to each other. Essentially the relaxed objective function is the same as that of PCA

Appendix - Iterative Quantization (ITQ)

Both LSH and PCAH are hashing the items and then performing binary quantization. There is quantization error which is the error between the projected values $\boldsymbol{v}$ and the quantized binary values $sgn(\boldsymbol{v})$:

$$\mathcal{E} = \| \boldsymbol{v} - sgn(\boldsymbol{v}) \|^2$$

Smaller the error $\mathcal{E}$, the better the binary codes will preserve the original locality structure

Appendix - Iterative Quantization (ITQ)

Rotate the data to achieve the optimized error